Final code?

April 2, 2025

[1]:
```
# As a Test:

# Parameters
a = 2
b = 3
e = 7
p = 191
# d = 163 check below

# verify that gcd conditions are checked and that inverses exist.
```

[2]:
```
# Python code to demonstrate naive
# method to compute gcd ( Euclidean algo )


def computeGCD(x, y):
    while(y):
        x, y = y, x % y
    return abs(x)

# a = 60
# b = 48

# prints 12
# print ("The gcd of 60 and 48 is : ",end="")
```

[3]:
```
print(computeGCD(2,191))
print(computeGCD(7, 190))
```

```
1
1
```

[4]:
```
d = pow(7,-1,190)
d
```

[4]: 163

[5]:
```
plaintext =␣
  ↪'HELLOWORLDTHEQUICKBROWNFOXJUMPSOVERTHELAZYDOGMANYTIMESINTHEDARKNESSWHILESTARS'
plaintext +=␣
  ↪'GLINTABOVETREESANDWINDWHISPERSOFTHEMIDNIGHTSKYTHESHADOWSMOVEQUIETLYACROSS'
plaintext +=␣
  ↪'THEFLOORASMOONLIGHTFILTERSTHROUGHTHEWINDOWPANEINTHEDISTANCEANOWLHOOTSCALLING'
plaintext +=␣
  ↪'TOTHEDEEPFORESTBEYONDTHETREESTHEAIRISSILENTBUTFULLOFSECRETSWHICHONLYTHENIGHTKNOW'
plaintext +=␣
  ↪'SHOWTOKEEPSOMETIMESTHEQUIETTELLSMORETHANWORDSCOULDEVERDOANDINTHESECRETPRESENCEOFDARKNESS'
plaintext +=␣
  ↪'EVERYTHINGFEELSMAGNIFIEDEVERYWHISPEREVERYSTEPANDINTHEMOMENTOFSUSPENSETHEWORLDHOLDSITSBREAT

len(plaintext)
```

[5]:
```
485
```

[6]:
```
# convert letters to numbers
def char_to_num(c):
    return 26 if c == " " else ord(c.upper()) - ord('A')

plaintext_nums = [char_to_num(c) for c in plaintext]

# encrypt
def encrypt_affine_exponent(m, a, b, e, p):
    return (a * pow(m, e, p) + b) % p

cipher_nums = [encrypt_affine_exponent(m, a, b, e, p) for m in plaintext_nums]

# outputs
print("Plaintext:", plaintext)
print("\nPlaintext nums:", plaintext_nums)
print("\nSize of cipher text", len(cipher_nums))
print("\nCiphertext nums:", cipher_nums)
```

```
Plaintext: HELLOWORLDTHEQUICKBROWNFOXJUMPSOVERTHELAZYDOGMANYTIMESINTHEDARKNESSWH
ILESTARSGLINTABOVETREESANDWINDWHISPERSOFTHEMIDNIGHTSKYTHESHADOWSMOVEQUIETLYACROS
STHEFLOORASMOONLIGHTFILTERSTHROUGHTHEWINDOWPANEINTHEDISTANCEANOWLHOOTSCALLINGTOT
HEDEEPFORESTBEYONDTHETREESTHEAIRISSILENTBUTFULLOFSECRETSWHICHONLYTHENIGHTKNOWSHO
WTOKEEPSOMETIMESTHEQUIETTELLSMORETHANWORDSCOULDEVERDOANDINTHESECRETPRESENCEOFDAR
KNESSEVERYTHINGFEELSMAGNIFIEDEVERYWHISPEREVERYSTEPANDINTHEMOMENTOFSUSPENSETHEWOR
LDHOLDSITSBREATH

Plaintext nums: [7, 4, 11, 11, 14, 22, 14, 17, 11, 3, 19, 7, 4, 16, 20, 8, 2,
10, 1, 17, 14, 22, 13, 5, 14, 23, 9, 20, 12, 15, 18, 14, 21, 4, 17, 19, 7, 4,
11, 0, 25, 24, 3, 14, 6, 12, 0, 13, 24, 19, 8, 12, 4, 18, 8, 13, 19, 7, 4, 3, 0,
17, 10, 13, 4, 18, 18, 22, 7, 8, 11, 4, 18, 19, 0, 17, 18, 6, 11, 8, 13, 19, 0,
```

1, 14, 21, 4, 19, 17, 4, 4, 18, 0, 13, 3, 22, 8, 13, 3, 22, 7, 8, 18, 15, 4, 17,
18, 14, 5, 19, 7, 4, 12, 8, 3, 13, 8, 6, 7, 19, 18, 10, 24, 19, 7, 4, 18, 7, 0,
3, 14, 22, 18, 12, 14, 21, 4, 16, 20, 8, 4, 19, 11, 24, 0, 2, 17, 14, 18, 18,
19, 7, 4, 5, 11, 14, 14, 17, 0, 18, 12, 14, 14, 13, 11, 8, 6, 7, 19, 5, 8, 11,
19, 4, 17, 18, 19, 7, 17, 14, 20, 6, 7, 19, 7, 4, 22, 8, 13, 3, 14, 22, 15, 0,
13, 4, 8, 13, 19, 7, 4, 3, 8, 18, 19, 0, 13, 2, 4, 0, 13, 14, 22, 11, 7, 14, 14,
19, 18, 2, 0, 11, 11, 8, 13, 6, 19, 14, 19, 7, 4, 3, 4, 4, 15, 5, 14, 17, 4, 18,
19, 1, 4, 24, 14, 13, 3, 19, 7, 4, 19, 17, 4, 4, 18, 19, 7, 4, 0, 8, 17, 8, 18,
18, 8, 11, 4, 13, 19, 1, 20, 19, 5, 20, 11, 11, 14, 5, 18, 4, 2, 17, 4, 19, 18,
22, 7, 8, 2, 7, 14, 13, 11, 24, 19, 7, 4, 13, 8, 6, 7, 19, 10, 13, 14, 22, 18,
7, 14, 22, 19, 14, 10, 4, 4, 15, 18, 14, 12, 4, 19, 8, 12, 4, 18, 19, 7, 4, 16,
20, 8, 4, 19, 19, 4, 11, 11, 18, 12, 14, 17, 4, 19, 7, 0, 13, 22, 14, 17, 3, 18,
2, 14, 20, 11, 3, 4, 21, 4, 17, 3, 14, 0, 13, 3, 8, 13, 19, 7, 4, 18, 4, 2, 17,
4, 19, 15, 17, 4, 18, 4, 13, 2, 4, 14, 5, 3, 0, 17, 10, 13, 4, 18, 18, 4, 21, 4,
17, 24, 19, 7, 8, 13, 6, 5, 4, 4, 11, 18, 12, 0, 6, 13, 8, 5, 8, 4, 3, 4, 21, 4,
17, 24, 22, 7, 8, 18, 15, 4, 17, 4, 21, 4, 17, 24, 18, 19, 4, 15, 0, 13, 3, 8,
13, 19, 7, 4, 12, 14, 12, 4, 13, 19, 14, 5, 18, 20, 18, 15, 4, 13, 18, 4, 19, 7,
4, 22, 14, 17, 11, 3, 7, 14, 11, 3, 18, 8, 19, 18, 1, 17, 4, 0, 19, 7]

Size of cipher text 485

Ciphertext nums: [96, 110, 31, 31, 65, 149, 65, 9, 31, 175, 98, 96, 110, 93, 72,
138, 68, 11, 5, 9, 65, 149, 105, 15, 65, 95, 88, 72, 37, 80, 187, 65, 170, 110,
9, 98, 96, 110, 31, 3, 75, 153, 175, 65, 54, 37, 3, 105, 153, 98, 138, 37, 110,
187, 138, 105, 98, 96, 110, 175, 3, 9, 11, 105, 110, 187, 187, 149, 96, 138, 31,
110, 187, 98, 3, 9, 187, 54, 31, 138, 105, 98, 3, 5, 65, 170, 110, 98, 9, 110,
110, 187, 3, 105, 175, 149, 138, 105, 175, 149, 96, 138, 187, 80, 110, 9, 187,
65, 15, 98, 96, 110, 37, 138, 175, 105, 138, 54, 96, 98, 187, 11, 153, 98, 96,
110, 187, 96, 3, 175, 65, 149, 187, 37, 65, 170, 110, 93, 72, 138, 110, 98, 31,
153, 3, 68, 9, 65, 187, 187, 98, 96, 110, 15, 31, 65, 65, 9, 3, 187, 37, 65, 65,
105, 31, 138, 54, 96, 98, 15, 138, 31, 98, 110, 9, 187, 98, 96, 9, 65, 72, 54,
96, 98, 96, 110, 149, 138, 105, 175, 65, 149, 80, 3, 105, 110, 138, 105, 98, 96,
110, 175, 138, 187, 98, 3, 105, 68, 110, 3, 105, 65, 149, 31, 96, 65, 65, 98,
187, 68, 3, 31, 31, 138, 105, 54, 98, 65, 98, 96, 110, 175, 110, 110, 80, 15,
65, 9, 110, 187, 98, 5, 110, 153, 65, 105, 175, 98, 96, 110, 98, 9, 110, 110,
187, 98, 96, 110, 3, 138, 9, 138, 187, 187, 138, 31, 110, 105, 98, 5, 72, 98,
15, 72, 31, 31, 65, 15, 187, 110, 68, 9, 110, 98, 187, 149, 96, 138, 68, 96, 65,
105, 31, 153, 98, 96, 110, 105, 138, 54, 96, 98, 11, 105, 65, 149, 187, 96, 65,
149, 98, 65, 11, 110, 110, 80, 187, 65, 37, 110, 98, 138, 37, 110, 187, 98, 96,
110, 93, 72, 138, 110, 98, 98, 110, 31, 31, 187, 37, 65, 9, 110, 98, 96, 3, 105,
149, 65, 9, 175, 187, 68, 65, 72, 31, 175, 110, 170, 110, 9, 175, 65, 3, 105,
175, 138, 105, 98, 96, 110, 187, 110, 68, 9, 110, 98, 80, 9, 110, 187, 110, 105,
68, 110, 65, 15, 175, 3, 9, 11, 105, 110, 187, 187, 110, 170, 110, 9, 153, 98,
96, 138, 105, 54, 15, 110, 110, 31, 187, 37, 3, 54, 105, 138, 15, 138, 110, 175,
110, 170, 110, 9, 153, 149, 96, 138, 187, 80, 110, 9, 110, 170, 110, 9, 153,
187, 98, 110, 80, 3, 105, 175, 138, 105, 98, 96, 110, 37, 65, 37, 110, 105, 98,
65, 15, 187, 72, 187, 80, 110, 105, 187, 110, 98, 96, 110, 149, 65, 9, 31, 175,
96, 65, 31, 175, 187, 138, 98, 187, 5, 9, 110, 3, 98, 96]

Now, suppose an Attacker intercepted the Ciphertext Numbers, now the process of attacking begins here, suppose they dont know a, b, e (and d) or p yet.

[7]:
```
import math

def is_prime(n):
    """Note: check if n is a prime number."""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(math.isqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True

def next_prime_from_max(lst):
    """Note: return the next prime candidate  max value in the list.
    (This is a guess, may not always work, You may have to try the prime after␣
 ↪if
    at the end of the code the decrypted text doesnt make sense in english) """
    if not lst:
        return None

    candidate = max(lst)
    while not is_prime(candidate):
        candidate += 1

    return candidate
```

[8]:
```
# we picked a prime that will make the guess work to make this example, may not␣
 ↪always work
# (so it's possible you may have to also try couple of primes after as well)

ciphertext_nums = cipher_nums
# # UNCOMMENT IF NEEDED for VISUALIZATION
# print(ciphertext_nums, '\n')


next_prime = next_prime_from_max(ciphertext_nums)
print(f"Next prime >= {next_prime}")
```

```
Next prime >= 191
```

[9]:
```
from collections import Counter

cipher_counts = Counter(ciphertext_nums)
most_common = cipher_counts.most_common(10) # can increase this if the end␣
 ↪results dont make sense in english.

print("Top 10 most common ciphertext values:")
vals = []
for val, freq in most_common:
    print(f"Ciphertext {val} appears {freq} times")
    vals.append(val)

vals
```

```
Top 10 most common ciphertext values:
Ciphertext 110 appears 69 times
Ciphertext 98 appears 45 times
Ciphertext 187 appears 41 times
Ciphertext 65 appears 38 times
Ciphertext 96 appears 34 times
Ciphertext 105 appears 31 times
Ciphertext 138 appears 30 times
Ciphertext 9 appears 27 times
Ciphertext 31 appears 23 times
Ciphertext 175 appears 20 times
```

[9]:
```
[110, 98, 187, 65, 96, 105, 138, 9, 31, 175]
```

[10]:
```
from collections import Counter

def modinv(a, p):
    return pow(a, -1, p)

def find_root_exponent(m1, m2, m3, c1, c2, c3, p, max_e=20): # e is fixed here␣
 ↪to avoid large e's for efficiency, but you can increase it if the end␣
 ↪results dont make sense.
    numerator = (c1 - c2) % p
    denominator = (c2 - c3) % p
    if denominator == 0:
        return None
    try:
        R = (numerator * modinv(denominator, p)) % p
    except ValueError:
        return None
    one_plus_R = (1 + R) % p

    for e in range(1, max_e + 1):
```

```
        m1e = pow(m1, e, p)
        m2e = pow(m2, e, p)
        m3e = pow(m3, e, p)
        f = (m1e - one_plus_R * m2e + R * m3e) % p
        if f == 0:
            return e
    return None
```

[11]:
```
# Frequency analysis
counter = Counter(ciphertext_nums)
top_cipher = [val for val, _ in counter.most_common(10)]

print(top_cipher)
```

```
[110, 98, 187, 65, 96, 105, 138, 9, 31, 175]
```

[12]:
```
# Common English letters and their numeric mappings
common_plain = {
    'E': 4, 'T': 19, 'A': 0, 'O': 14, 'I': 8, 'N': 13, 'S': 18, 'R': 17
}
common_plain_vals = list(common_plain.values())

# Try all combinations of plaintext and ciphertext triplets
results = []

for i in range(len(common_plain_vals)):
    for j in range(i + 1, len(common_plain_vals)):
        for k in range(len(common_plain_vals)):
            m1, m2, m3 = common_plain_vals[i], common_plain_vals[j],␣
 ↪common_plain_vals[k]
            for ci in range(len(top_cipher)):
                for cj in range(len(top_cipher)):
                    for ck in range(len(top_cipher)):
                        if len({ci, cj, ck}) < 3:
                            continue
                        c1, c2, c3 = top_cipher[ci], top_cipher[cj],␣
 ↪top_cipher[ck]
                        e = find_root_exponent(m1, m2, m3, c1, c2, c3, p)
                        if e is not None:
                            results.append({
                                "plaintext": (m1, m2, m3),
                                "ciphertext": (c1, c2, c3),
                                "e": e
                            })

print("size of results:", len(results))
print("\nCandidates:")
```

```
# uncomment to get list of all possible combinations of triplets of cipher and␣
 ↪plaintexts with their exponent e

# # Visualization: First 10 triplets (feel free to increase size or print all)
# # UNCOMMMENT IF NEEDED
# for i in range(10):
#     print(results[i])
```

size of results: 15159

Candidates:

[13]:
```
# Frequency analysis: get top 10 most frequent numbers/letters in the ciphertext
counter = Counter(ciphertext_nums)
top_cipher = [val for val, _ in counter.most_common(10)]

# Common English letters: E, T, A, O, I, N, S, R
common_plain = {'E': 4, 'T': 19, 'A': 0, 'O': 14, 'I': 8, 'N': 13, 'S': 18, 'R':
 ↪ 17}
common_plain_vals = list(common_plain.values())

# Modulus
p = 191
phi_p = p - 1

# Test all triplets of
results = []

for i in range(len(common_plain_vals)):
    for j in range(i + 1, len(common_plain_vals)):
        for k in range(len(common_plain_vals)):
            m1, m2, m3 = common_plain_vals[i], common_plain_vals[j],␣
 ↪common_plain_vals[k]
            for ci in range(len(top_cipher)):
                for cj in range(len(top_cipher)):
                    for ck in range(len(top_cipher)):
                        if len({ci, cj, ck}) < 3:
                            continue
                        c1, c2, c3 = top_cipher[ci], top_cipher[cj],␣
 ↪top_cipher[ck]
                        e = find_root_exponent(m1, m2, m3, c1, c2, c3, p)
                        if e is not None:
                            try:
                                d = pow(e, -1, phi_p)
                                results.append({
                                    "plaintext": (m1, m2, m3),
                                    "ciphertext": (c1, c2, c3),
```

```
                                    "e": e,
                                    "d": d
                                })
                            except ValueError:
                                continue

print("size of results:", len(results))
print("\nCandidates:")

# # Visualization: First 10 triplets (feel free to increase size or print all)
# # UNCOMMENT IF NEEDED
# for i in range(10):
#     print(results[i])
```

```
size of results: 4410

Candidates:
```

[14]:
```
top_cipher = vals
top_cipher
```

[14]:
```
[110, 98, 187, 65, 96, 105, 138, 9, 31, 175]
```

Once you find a possible exponent e, we then solve for a and b. ideally exponent should be small for efficient use of the system, so let's try the ones with smaller exponent e first, say e<10

Then figure out if the decrypted plaintext maks sense in english. Otherwise, try another possible combination of triplet of characters to find another e.

[15]:
```
# Filter the triplets/matches with their keys by filtering e < 10 (for␣
 ↪efficiency, can increase this if needed) and compute a and b for each triplet
def solve_a_b(m1, m2, c1, c2, e, p):
    m1e = pow(m1, e, p)
    m2e = pow(m2, e, p)
    numerator = (c1 - c2) % p
    denominator = (m1e - m2e) % p
    try:
        den_inv = pow(denominator, -1, p)
        a = (numerator * den_inv) % p
        b = (c1 - a * m1e) % p
        return a, b
    except ValueError:
        return None, None
```

[16]:
```
# Filter triplets/matches with e < 10
filtered_matches = [match for match in results if match['e'] < 10]

# Compute a and b for each match
results_with_ab = []
```

```
for match in filtered_matches:
    m1, m2, m3 = match['plaintext']
    c1, c2, c3 = match['ciphertext']
    e = match['e']
    a, b = solve_a_b(m1, m2, c1, c2, e, p)
    if a is not None:
        results_with_ab.append({
            "plaintext": (m1, m2, m3),
            "ciphertext": (c1, c2, c3),
            "e": e,
            "a": a,
            "b": b
        })

print(len(results_with_ab), '\n') # even shorter than last time, but still much␣
 ↪larger than we want.

# # For Visualization: can increase size if needed.
# # UNCOMMENT IF NEEDED
# for i in range(10):
#     print(results_with_ab[i])
```

2616

[17]:
```
# Updated decryption logic using direct modular root and improved num_to_char

def decrypt_affine_exponent_direct(cipher_nums, a, b, d, p, length):
    try:
        a_inv = pow(a, -1, p)
    except ValueError:
        return None  # skip if a is not invertible

    plaintext_nums = []
    for c in cipher_nums[:length]:  # only decrypt first 20 for visibility
        y = (a_inv * (c - b)) % p
        m = pow(y, d, p)
        plaintext_nums.append(m)

    return plaintext_nums
```

[18]:
```
# Updated num_to_char: wrap around A-Z (0-25)
def num_to_char(n):
    return chr((n % 26) + ord('A')) if n is not None else '?'
```

[19]:
```
# Rerun decryption using improved logic
decryption_results_fixed = []
```

```
for match in results:
    if match['e'] < 10:
        m1, m2, m3 = match["plaintext"]
        c1, c2, c3 = match["ciphertext"]
        e = match["e"]
        d = match["d"]
        a, b = solve_a_b(m1, m2, c1, c2, e, p)
        if a is None:
            continue
        decrypted_nums = decrypt_affine_exponent_direct(ciphertext_nums, a, b,␣
 ↪d, p, length=100)
        decrypted_chars = ''.join(num_to_char(n) for n in decrypted_nums)
        decryption_results_fixed.append({
            "e": e, "d": d, "a": a, "b": b,
            "plaintext_preview": decrypted_chars
        })
```

[20]:
```
# Filter out plaintexts previews that contain '?'
filtered_decryption_results = [
    r for r in decryption_results_fixed
    if '?' not in r['plaintext_preview']
]
```

[21]:
```
# # UNCOMMENT if needed for VISUALIZATION
# print(len(filtered_decryption_results), '\n')
# for i in range(20):
#     print(filtered_decryption_results[i])
```

[22]:
```
# If you can't manually scan for english words, THEN We define a simple list of␣
 ↪most common English words (can be expanded if needed)

common_words = common_words = {
    "HELLO", "WORLD", "THE", "BE", "TO", "OF", "AND", "A", "IN", "THAT",
    "HAVE", "I", "IT", "FOR", "NOT", "ON", "WITH", "HE", "AS", "YOU",
    "DO", "AT", "THIS", "BUT", "HIS", "BY", "FROM", "THEY", "WE", "SAY",
    "HER", "SHE", "OR", "AN", "WILL", "MY", "ONE", "ALL", "WOULD", "THERE",
    "THEIR", "WHAT", "SO", "UP", "OUT", "IF", "ABOUT", "WHO", "GET", "WHICH",
    "GO", "ME", "WHEN", "MAKE", "CAN", "LIKE", "TIME", "NO", "JUST", "HIM",
    "KNOW", "TAKE", "PEOPLE", "INTO", "YEAR", "YOUR", "GOOD", "SOME", "COULD",
    "THEM", "SEE", "OTHER", "THAN", "THEN", "NOW", "LOOK", "ONLY", "COME",
    "ITS", "OVER", "THINK", "ALSO", "BACK", "AFTER", "USE", "TWO", "HOW",
    "OUR", "WORK", "FIRST", "WELL", "WAY", "EVEN", "NEW", "WANT", "BECAUSE",
    "ANY", "THESE", "GIVE", "DAY", "MOST", "US"
}

# # Force include the word "the" in list of possible plaintexts, since it's the␣
 ↪most common word in english.
```

```
# # Scan decrypted previews for 2+ matches (since "the" is included forcibly␣
 ↪from the common words list

seen = set() # used for ensuring no duplicates
the_matches_only_unique = []

for entry in filtered_decryption_results: # Iterates over all decrypted preview␣
 ↪results that had no '?' in them.
    preview = entry['plaintext_preview']
    matches = [w for w in common_words if w in preview]
    if "THE" not in matches: # force include "the", otherwise continue on to␣
 ↪the next set of texts
        continue

    key = (entry['a'], entry['b'], entry['e'], entry['d'], preview)
    if key not in seen:
        seen.add(key)
        the_matches_only_unique.append({**entry, "matched_words": matches})

# # Print unique entries (UNCOMMENT IF NEEDED FOR VISUALIZATION)
# for i, entry in enumerate(the_matches_only_unique, 1):
#     print(f"[{i}] a={entry['a']}, b={entry['b']}, e={entry['e']},␣
 ↪d={entry['d']}")
#     print(f"    Preview: {entry['plaintext_preview']}")
#     print(f"    Matched words: {entry['matched_words']}")
#     print("-" * 50)
```

[23]:
```
# Find the maximum number of matched words in the list of "THE"-containing␣
 ↪unique results

max_match_count = 0
max_matched_entries = []

for entry in the_matches_only_unique:
    num_matches = len(entry['matched_words'])
    if num_matches > max_match_count:
        max_match_count = num_matches
        max_matched_entries = [entry]
    elif num_matches == max_match_count:
        max_matched_entries.append(entry)

print(f"Maximum matched words: {max_match_count}")
for i, entry in enumerate(max_matched_entries, 1):
    print(f"[{i}] a={entry['a']}, b={entry['b']}, e={entry['e']},␣
 ↪d={entry['d']}")
    print(f"    Preview: {entry['plaintext_preview']}")
```

```
    print(f"    Matched words: {entry['matched_words']}")
    print("-" * 50)
```

```
Maximum matched words: 16
[1] a=2, b=3, e=7, d=163
    Preview: HELLOWORLDTHEQUICKBROWNFOXJUMPSOVERTHELAZYDOGMANYTIMESINTHEDARKNESS
WHILESTARSGLINTABOVETREESANDWINDW
    Matched words: ['ANY', 'I', 'TIME', 'DO', 'THE', 'OR', 'A', 'SO', 'WORLD',
'AND', 'HELLO', 'AN', 'OVER', 'IN', 'HE', 'ME']
--------------------------------------------------
```

extract the a, b, e, and d from that maximum match plain text. then apply the decryption function and then the num to char functions we defined to turn our list of cipher text nums into list of characters, then combine the characters in the list to a string at the end

[24]:
```
# Extract the best match key values
best_match = max_matched_entries[0]
a_best = best_match['a']
b_best = best_match['b']
e_best = best_match['e']
d_best = best_match['d']

# Full decryption function using updated logic
def decrypt_affine_full(cipher_nums, a, b, d, p):
    try:
        a_inv = pow(a, -1, p)
    except ValueError:
        return None

    return [pow((a_inv * (c - b)) % p, d, p) for c in cipher_nums]

# Decrypt full ciphertext
full_decrypted_nums = decrypt_affine_full(ciphertext_nums, a_best, b_best,␣
 ↪d_best, p)

# Convert decrypted numbers to characters (A-Z wrapping)
def num_to_char(n):
    return chr((n % 26) + ord('A')) if n is not None else '?'

# Combine characters into a single string
decrypted_message = ''.join(num_to_char(n) for n in full_decrypted_nums)

decrypted_message[:len(cipher_nums)]  # Show all characters for visibility
```

[24]:
```
'HELLOWORLDTHEQUICKBROWNFOXJUMPSOVERTHELAZYDOGMANYTIMESINTHEDARKNESSWHILESTARSGL
INTABOVETREESANDWINDWHISPERSOFTHEMIDNIGHTSKYTHESHADOWSMOVEQUIETLYACROSSTHEFLOORA
SMOONLIGHTFILTERSTHROUGHTHEWINDOWPANEINTHEDISTANCEANOWLHOOTSCALLINGTOTHEDEEPFORE
STBEYONDTHETREESTHEAIRISSILENTBUTFULLOFSECRETSWHICHONLYTHENIGHTKNOWSHOWTOKEEPSOM
```

ETIMESTHEQUIETTELLSMORETHANWORDSCOULDEVERDOANDINTHESECRETPRESENCEOFDARKNESSEVERY
THINGFEELSMAGNIFIEDEVERYWHISPEREVERYSTEPANDINTHEMOMENTOFSUSPENSETHEWORLDHOLDSITS
BREATH'