Guessy writeup

Vlad Manolescu (LambdaF)

September 16, 2025

1 Abstract of challenge

This challenge consists of a quiz where we get 10 encoded strings, one after another, and we had to decode all of them consecutively in order to get to the flag.

1.1 Problem description

Don't you love guessy challenges?
nc 0.cloud.chals.io 32957

2 Solution

This solution exploits Paillier homomorphic encryption to extract the secret bit-by-bit using a zero-knowledge attack.

2.1 Core Strategy

Base-3 encoding: Represent each possible secret (0-2047) as 7 base-3 digits (trits) Zero-causing values: Create special values that make Paillier decryption return 0 only when a candidate equals the secret Trit extraction: Use 7 queries to extract one trit at a time

2.2 Per Query (extracts 1 trit)

For each candidate secret, place a zero-causing factor in LEFT/RIGHT based on its i-th trit: Trit 0 → LEFT side Trit 1 → RIGHT side Trit 2 → BOTH sides When server processes the query, whichever side(s) return ”0” reveal the secret’s i-th trit

2.3 Key Functions

make_x(n, candidate): Creates value that decrypts to 0 iff candidate == secret rerand(): Ensures all query values are distinct Base-3 decode: Converts 7 extracted trits back to the secret number

2.4 Result

After 7 queries, you know all 7 trits of the secret and can reconstruct it perfectly. The attack works because Paillier’s homomorphic properties allow crafting ciphertexts that reveal information through zero-detection.

3 Exploit Code

```
#!/usr/bin/env python3
from pwn import remote, context

# Optional: show progress
context.log_level = "info"

D = 0xD3ADC0DE  # the fixed offset added to the secret inside Paillier

# -------- Helpers ----------------------------------------------------------

def to_base3(x, digits):
    trits = []
    for _ in range(digits):
        trits.append(x % 3)
        x //= 3
    return trits  # LSB-first

```

```python
def make_x(n, candidate_r):
    """
    We want B.decrypt(e_secret * x) = secret + D - r == 0  iff candidate_r == secret.
    So choose r = D + candidate_r.
    For g = n+1, g^{-r}  1 - r*n (mod n^2). (Binomial shortcut)
    """
    n2 = n * n
    r_total = (D + candidate_r) % n  # reduce r modulo n (safe in Paillier)
    return (1 - r_total * n) % n2

def rerand(n, t):
    """
    Re-randomizer: t^n mod n^2. Keeps plaintext the same, gives a distinct ciphertext.
    Only TWO pow() calls per query (one per side).
    """
    n2 = n * n
    return pow(t, n, n2)

# -------- Per-test solve ----------------------------------------------------

def solve_one_test(io, test_idx):
    # Wait for modulus n (Paillier's n_B)
    io.recvuntil(b"n = ")
    n = int(io.recvline().strip())
    n2 = n * n

    # The prompt may not include a trailing newline
    io.recvuntil(b"You can ask 7 questions:")

    # Precompute base-3 encodings for 0..2047 as 7 trits (3^7 = 2187 > 2048)
    codes = [to_base3(r, 7) for r in range(2048)]

    for i in range(7):
        left, right = [], []

        # Two distinct re-randomizers per query
        randL = rerand(n, 2 + 2*i)
        randR = rerand(n, 3 + 2*i)

        # Place the zero-causing factors based on trit i
        for cand in range(2048):
            x = make_x(n, cand)  # decrypt(e_secret * x) == secret + D - (D + cand) == secret - cand
            trit = codes[cand][i]
            if trit == 0:
                # Zero LEFT when secret == cand
                left.append((x * randL) % n2)
            elif trit == 1:
                # Zero RIGHT when secret == cand
                right.append((x * randR) % n2)
            else:  # trit == 2 -> zero BOTH
                left.append((x * randL) % n2)
```

```
                right.append((x * randR) % n2)

        # Equalize halves (server splits the line exactly in half)
        half_len = max(len(left), len(right))

        # Neutral padding: (1 + k*n) * rand? — distinct, non-zero with overwhelming probability
        padk = 0
        while len(left) < half_len:
            left.append(((1 + padk * n) % n2) * randL % n2)
            padk += 1
        while len(right) < half_len:
            right.append(((1 + padk * n) % n2) * randR % n2)
            padk += 1

        query = left + right

        # Safety checks: even length & all distinct within this line
        assert len(query) % 2 == 0
        assert len(query) == len(set(query))

        io.sendline(" ".join(str(v) for v in query).encode())

    # Read 7 response lines: each "L R", detect where 0 appears
    trits = []
    for i in range(7):
        line = io.recvline().decode().strip()
        while not line:
            line = io.recvline().decode().strip()
        Ls, Rs = line.split()
        if Ls == "0" and Rs == "0":
            trits.append(2)
        elif Ls == "0":
            trits.append(0)
        elif Rs == "0":
            trits.append(1)
        else:
            # With the correct construction, one side MUST be 0 per query.
            # If not (extremely unlikely due to padding colliding), default to 0.
            trits.append(0)

    # Decode base-3 (LSB-first)
    secret = 0
    mul = 1
    for t in trits:
        secret += t * mul
        mul *= 3

    # Guess
    io.recvuntil(b"Can you guess my secret?")
    io.sendline(str(secret).encode())

    # Server sends empty line first, then the actual response
```

```
    verdict =
        io.recvline(timeout=10).decode(errors="ignore").strip()
    if not verdict:
        verdict =
            io.recvline(timeout=5).decode(errors="ignore").strip()

    if "Correct!" not in verdict:
        raise RuntimeError(f"[Test #{test_idx}] Wrong guess?
            Got: '{verdict}'")

# -------- Main
    ------------------------------------------------------------

def main():
    host = "0.cloud.chals.io"
    port = 32957
    io = remote(host, port)

    # There are 10 tests
    for t in range(10):
        io.recvuntil(b"Test #")
        io.recvline()  # rest of header
        solve_one_test(io, t)

    # Print the final flag
    remaining = io.recvall(timeout=5)
    if remaining:
        print(remaining.decode(errors="ignore"))

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

Listing 1: Paillier Zero-Knowledge Attack Implementation