

```

{
  "id": 50,
  "name": "Peak Prime",
  "description": "> P(L)eak prime number, no one gonna guess this.\r\n",
  "max_attempts": 0,
  "value": 481,
  "category": "Cryptography",
  "type": "dynamic",
  "state": "visible",
  "requirements": null,
  "connection_info": null,
  "next_id": null,
  "attribution": "*Q\u00e2*",
  "logic": "any",
  "initial": null,
  "minimum": null,
  "decay": null,
  "function": "static"
},

```

The leak gives the top 512–25 bits of q , leaving only 25 unknown lower bits.

That reduces q to a small search window (about 33 million integers), but by iterating only primes in that window the search becomes practical.

Once q is found, factoring n is trivial, allowing RSA decryption to recover the plaintext flag.

`solve_peak_prime.py` is a solver that:

Parses the leaked values,

Iterates prime candidates for q in the constrained range,

Factors n , computes d , and decrypts the ciphertext to print the flag.

Finally, after brute forcing, we got the flag `vgucypher{f3ker_at_his_b3st_peak_prime!!!}`