Q1 Crypto

This CTF challenge is related to the Discrete Logarithm Problem (DLP), focusing on how its security is established. The challenge tests your understanding of asymmetric encryption by using an outdated encryption bit-width. (How to generate this?)

```
// Crypto challange
// Assume only you know the password
// Prove code below is not safe
if (info.try) {
    try {
        let a = BigInt('0x' + info.try)
        let b = BigInt("0xb6d733a404d0b06e51dcf52fec53b6b9ed807b3bdc13dbe33e5e59182f66b733")
        let c = BigInt("0x3e9")
        let d = "4384742de6012452302030a8c48605374070da2f41d5847b066bcd94f32a05e0"
        let result = ((a ** c) % b).toString(16)
        if (result == d) {
            let hex = Buffer.from(info.try, 'hex')
            socket.send(JSON.stringify({ flag: hex.toString('utf8') }))
        }
        else {
            socket.send(JSON.stringify({ flag: "Zzzzzzzz...." }))
        }
```

Figure 1 DLP

POC:

1. **Factorizing large numbers**

   Tools: yafu (or any tool)

   Key in factor(0xb6d733a404d0b06e51dcf52fec53b6b9ed807b3bdc13dbe33e5e59182f66b733)

   We get

```
***factors found***

P39 = 244797265212401102686995522653336482037
P39 = 337835338562002625014208649165305613959

ans = 1
```

Figure 2 p and q

   Prime1 = 0d244797265212401102686995522653336482037

   Prime2 = 0d337835338562002625014208649165305613959

2. **Calculate φp (Euler's Totient Function)**

   We use Euler's Totient Function to determine the uniqueness of G because a (figure 1) is not a prime.

   φp = (Prime1-1) * (Prime2-1)

   =

   0d82701166972083873963502681321091904267252851881480149626751213724 1

20817858488
=
0xB6D733A404D0B06E51DCF52FEC53B6B8372D8608437F1E8DD3D71CFD03E3F7B8

3. **Start calc G (**a in figure 1**)**
   Tools: RDLP (windows only)

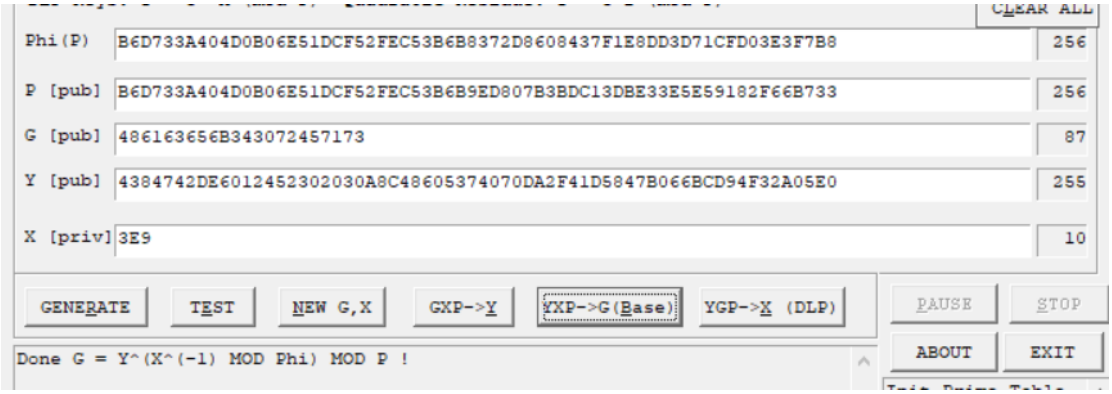

Figure 3 Calc G

Put them into the RDLP
Phi(p) <- φp in part2
P <- b in figure 1
Y <- d in figure 1
X <- c in figure 1
Click the YXP-> G
We get answer G (a in figure 1) = 0x486163656B343072457173.

4. **Verification**
   Write an Attack Script

```
const WebSocket = require("ws");


let ws = new WebSocket.WebSocket("ws://10.0.0.109:5555")
ws.on("open", () => {
    console.log("start")
    ws.send(JSON.stringify({try:"486163656b343072457173"}))
});

ws.on("message", (data) => {
    console.log(JSON.parse(data))
});

ws.on("close", () => {
    console.log("closed")
});

ws.on("error", (error) => {
    console.log("error")
});
```

Figure 4 Attack script

Run it, and now we get the correct flag.

```
{ flag: 'Hacek40rEqs' }
```

Figure 5 final flag