Group Project{ion}	Anakin Dey
Last Edited on 7/8/23 at 10:04	

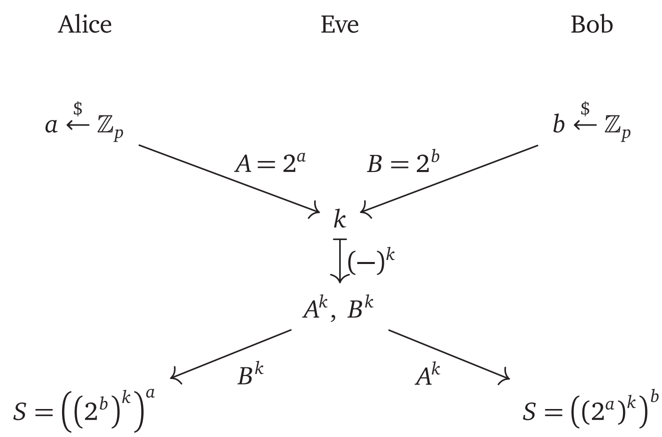

Figure 1: Diagram of public and private information

Solution

The basis of this challenge is Diffie-Hellman key exchange. Alice computes a secret value $a \xleftarrow{\$} \mathbb{Z}_p$ and Bob computes a secret value $b \xleftarrow{\$} \mathbb{Z}_p$. Then they each respectively send $A \equiv 2^a$ (mod ()p) and $B \equiv 2^b$ (mod ()p). So $A, B \in \mathbb{Z}_p$. However, instead of Alice directly receiving B and Bob directly receiving A we play the role of Eve in a Man in the Middle attack where we modify these values before passing them onto their intended recipients. We choose some k and now Alice will receive B^k and Bob will receive A^k. But what does this k do and how do we choose a good value? Note that we have

$$A^k \equiv (2^a)^k \equiv \left(2^k\right)^a \pmod p$$
$$B^k \equiv \left(2^b\right)^k \equiv \left(2^k\right)^b \pmod p$$

When 2 was the generator of our group we were working in a group $\mathbb{Z}_p$ of order $p-1$. However, now we essentially have that 2^k is the generator of our group which we will denote as $\left\langle 2^k \right\rangle \leq \mathbb{Z}_p$, the subgroup of $\mathbb{Z}_p$ generated by 2^k. A bit of group theory tells us that if we have a group G and a subgroup $H \leq G$ then we must have that the order of H *divides* the order of G. Note that we have that $S = \left(2^k\right)^{ab}$ which means that $S \in \left\langle 2^k \right\rangle$. If we can choose k such that the order of $\left\langle 2^k \right\rangle$ is small, the discrete log problem becomes significantly easier. If we factor $p-1$ and choose a factor ω, then setting $k = \frac{p-1}{\omega}$ yields that the order of $\left\langle 2^k \right\rangle = \omega$. So if ω is small enough then brute force becomes feasible. Prime factorization takes a while and we only care about reasonably small values of ω so if we don't find a small factor of $p-1$ then we can just restart our solve script.

```
from Crypto.Util.number import long_to_bytes
import itertools
import hashlib
from Crypto.Cipher import AES
import time
import pwnlib.tubes
```

```python

def handle_pow(r):
    print(r.recvuntil(b'python3 '))
    print(r.recvuntil(b' solve '))
    challenge = r.recvline().decode('ascii').strip()
    p = pwnlib.tubes.process.process(['kctf_bypass_pow', challenge])
    solution = p.readall().strip()
    r.sendline(solution)
    print(r.recvuntil(b'Correct\n'))

# Sometimes we get harder numbers, so just to save time we'll set a time limit
attempts = 1
max_attempts = 5
time_limit = 5.00
while True:
    if attempts > max_attempts:
        print("Took too long...")
        exit(1)
    start = time.time()

    r = pwnlib.tubes.remote.remote('127.0.0.1', 1337)
    print(r.recvuntil(b'== proof-of-work: '))
    if r.recvline().startswith(b'enabled'):
        handle_pow(r)

    print("Getting Public Info...")
    r.recvlineS()
    r.recvlineS()
    int(r.recvlineS().split("=")[1])
    p = int(r.recvlineS().split("=")[1])
    A = int(r.recvlineS().split("=")[1])

    print("Choosing k...")

    # unique prime factorization
    # not too efficient
    print(f"Factoring {p - 1}")
    n = p - 1
    count = 0
    w = None
    for i in itertools.chain([2], itertools.count(3, 2)):
```

```
        if n <= 1 or (i >= 100000 and w != None):
            break
        if time.time() - start > time_limit:
            break

        fact = None
        while n % i == 0:
            fact = i
            n //= i

        if fact:
            print(f"    Prime factor: {fact}")
            count += 1
            w = fact
            if (fact >= 100 or count >= 3):
                break

    if time.time() - start > time_limit:
        print("Took too long, restarting...")
        attempts += 1
        r.close()
        continue

    print(f"Subgroup size = {w}")
    k = (p - 1) // w
    print(f"Using {k = }")
    Ak = pow(A, k, p)

    print("Sending...")
    r.sendline(bytes(str(k), "utf-8"))

    print("Receiving ciphertext...")
    r.recvlineS()
    c = long_to_bytes(int(r.recvlineS().split("=")[1]))

    print("Closing...")
    r.close()

    print("Searching for secrets...")
    # Have small subgroup, enumerate secrets
    # compute powers of Ak
```

```python
    for i in range(1, w + 1):
        S_ = pow(Ak, i, p)

        # test current power
        key = hashlib.md5(long_to_bytes(S_)).digest()
        cipher = AES.new(key, AES.MODE_ECB)
        m = cipher.decrypt(c)
        if b"uiuctf" in m:
            print(m)
            exit(0)

    print("Didn't find flag")
    exit(1)
```