The discrete Log problem

Angelos Assos
Description of dLog problem

- Given the triplet \((A, B, p)\) and that \(B \equiv A^x \mod p\) for some \(x\), can you find \(x\)?
- \(A\) is a generator of \(p\)
- We want to find the "log" of \(B\)

Turns out to be very hard => Good for crypto :)
Description of dLog problem

- Given the triplet \((A, B, p)\) and that \(B \equiv A^x \mod p\) for some \(x\), can you find \(x\)?
- \(A\) is a generator of \(p\)
- We want to find the "log" of \(B\), i.e. \(\log(B) = \log(A^x) = x\)

Turns out to be very hard = Good for crypto :)

Question: How do we use it for Zero Knowledge
Zero knowledge for the dLog problem

- Alice knows $x$ such that $B \equiv A^x \mod p$ and wants to prove to Bob that she knows it without revealing it.
- Alice chooses a randomness $r$ and sends $a = A^r \mod p$ to Bob.
- Bob sends a bit $b$ in $\{0, 1\}$ to Alice.
- Alice calculates $s = (r + b \times x) \mod (p - 1)$ and sends $A^s \mod p$ to Bob.
- Bob verifies that what Alice sent is equal to $aB^b$.

Repeat $N$ times until Bob is convinced.
```python
def ZKdLogProof(x, g, p, N = 100):
    bits = os.urandom(N)
    t = os.urandom(N)
    s = [1]

    for i in range(N):
        s.append((t[i] + 1) if (bits & pow(2,i)) else 0)%(p-1))

    return [bits, s, t]

def verify(y, g, p, bits, s, t, N = 100):
    for i in range(N):
        lhs = pow(g, t[i], p)
        rhs = (y[i] if (bits & pow(2,i)) else 1)*pow(g, t[i], p) % p

    if lhs != rhs:
        return False

    return True

if __name__ == "__main__":
    p = getPrime(10)  # getting a random prime of 10 bits
    g = random.randint(2, p-1)  # generator
    x = 30  # our secret
    y = pow(g, x, p)

    # We want to prove that we know that g^x = y (mod p)
    proof = ZKdLogProof(x, g, p)

    # returns 3 lists of bits b, randomnesses r, and x^r
    bits = proof[0]
    s = proof[1]
    t = proof[2]

    Result = verify(y, g, p, bits, s, t)

    if Result:
        print("The proof is correct!")
    else:
        print("The proof is not correct")
```

A very similar configuration: Diffie Hellman

Can one distinguish between and \((g, g^x, g^y, g^{xy})\) and \((g, g^x, g^y, t)\) for random \(t\)?

If discrete log is easy, then Diffie Hellman is also easy.
Attacks on the Diffie Hellman

There are several attacks on Diffie Hellman. Several of them are:

- Pollard Rho - using the birthday paradox, good for small modulus
- Pohlig-Hellman - factorizes $\phi(p)$, computes the discrete log for the primes of it independently, and then combines using the Chinese Remainder Theorem.
Can we break through here?

def genprime():
    p = 2
    while p.bit_length() < 1020:
        p *= getPrime(16)
    while True:
        x = getPrime(16)
        if isPrime((p * x) + 1):
            return (p * x) + 1
            break
    p, q = [genprime() for _ in range(2)]

    n = p * q
    g = 0x10000
    ct| = pow(g, flag, n)

    print(f'n: \{hex(n)}')
    print(f'g: \{hex(g)}')
    print(f'ct: \{hex(ct2)}')