Casino

April 2, 2023

[ ]:
```
import random

rng = random.SystemRandom()

# Secure group from RFC 3526
prime = int("""
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AACAA68 FFFFFFFF FFFFFFFF""".replace('\n', '').replace(' ', ''),
16)

generator = 11
g = generator
def play():
    challenge = rng.randint(0, 1)

    a, b, z = rng.randint(1, prime-1), rng.randint(1, prime-1), rng.randint(1,␣
 ↪prime-1)

    A, B, C, Z = pow(generator, a, prime), pow(generator, b, prime),␣
 ↪pow(generator, a*b, prime), pow(generator, z, prime)

    print(f"""Guess the random bit I have coosen!
Commitment: {A}, {B}, {C if challenge == 1 else Z}""")

    guess = int(input("> ").strip())

    if guess == challenge:
        print(f"""Correct! My challenge was {challenge}
```

```
Proof: {a}, {b}""")
        return 1
    else:
        print(f"""Wrong! My challenge was {challenge}
Proof: {a}, {b}""")
        return -1


"""Welcome to the first casino with fully provable randomness using␣
 ↪cryptographically hard problems!
It uses the Decisional Diffie-Hellman Problem to provide a commitment, which␣
 ↪can be verified by the player after the answer has been given.
Your balance is 100 €.
Aquire 200 € to get one of our premium flags
"""
```

[ ]:
```
'Welcome to the first casino with fully provable randomness using
cryptographically hard problems!\nIt uses the Decisional Diffie-Hellman Problem
to provide a commitment, which can be verified by the player after the answer
has been given.\nYour balance is 100 €.\nAquire 200 € to get one of our premium
flags\n'
```

3. 2048-bit MODP Group

This group is assigned id 14.

This prime is: 2^2048 - 2^1984 - 1 + 2^64 * { [2^1918 pi] + 124476 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AACAA68 FFFFFFFF FFFFFFFF
```

The generator is: 2.

[ ]:
```
from sympy import ntheory
```

[ ]:
```
pow(11,prime-1,prime)
```

[ ]:
```
1
```

```
[ ]: (prime-1)%2
```

```
[ ]: 0
```

```
[ ]: factors = ntheory.factorint(prime-1).keys()
     factors
```

```
[ ]: dict_keys([2, 161585030356555036501694569632119141244089706205701195564210048757
     003708533171771113098447086817846735589508689548520958773029366045975144268794 93
     092811076606087706257450887260135117898039118124442123094738793820552964323049 70
     586162271331126109661527045951884026211775956283985793505850052902793882551943 09
     236401289880274517848662807630835406696808997706682382795801841589483645365891 92
     294840319835950488601097084323612935515705668214659768096735818266604858538724 11
     399429428268460432264831803862513447775296418137556058704848649903420527717979 24
     332916458210681091155394954997243262341312084860179559262535226805452 79])
```

0.1 Game

$\mathbb{Z}_p$ with $g = 11$

$a, b, c \in \mathbb{Z}_p^*$

given

g^a, g^b with options $A = g^{a \cdot b}$ and $B = g^z$

guess if

$g^{a \cdot b} == A$

0.2 Parity Oracle

This is known as the decisional diffi hellmann problem. But there is a problem at first, $g = 11$ is not an official generator. However, we can verify with *fermat's little theorem* that it is still a generator. From my crypto lectures, I remember that we also need a group that has a prime order (that's why we usually use subgroups). Also, we notice that the order is even, meaning $p-1 = 2 \cdot m$ where m is a prime in this case. Boy *Fermat* said in his theorem that

$a^{p-1} = 1 \mod p \; \forall a$

Now, we can substitute $p - 1$ with $2 \cdot m$

$a^{p-1} = a^{2 \cdot m} = a^{2^m}$

So, all we need is that m is odd. In our case, m is even prime! We can exchange every even number with 2 because 2 must be a factor. However, if we substitute 2 with an odd number, we are not able to use *Fermat* theorem to the full extend because of *Carmichael numbers* - but they are kinda rare.

$w = g^a$ with a even $\implies w^m = 1 \mod p$

$w = g^a$ with a odd $\implies$ very likely $w^m \neq 1 \mod p$

With all that knowledge, we can check g^a, g^b and the challenge for parity (0 is even, 1 is odd):

a	b	$a \cdot b$
0	0	0
1	0	1
0	1	1
1	1	1

We infer two rules:

g^z with z odd but either a or b is even $\implies$ Challenge 0

g^z with z even but a and b are even $\implies$ Challenge 0

If none of these apply, we assume that challenge 1 was taken.

[ ]:
```
a,m = min(factors),max(factors)
assert a == 2
assert m % 2 == 1
```

[ ]:
```
def oracle_is_even(val,m,prime):
    return pow(val,m,prime) == 1
```

[ ]:
```
def oracle_dh(ga,gb,c,m,prime):
    ga_even = oracle_is_even(ga,m,prime)
    gb_even = oracle_is_even(gb,m,prime)
    c_even = oracle_is_even(c,m,prime)

    if (not ga_even and not gb_even and c_even) or ((ga_even or gb_even) and␣
 ↪not c_even):
        return False

    return True
```

[ ]:
```
# some tests
assert oracle_dh(pow(g,12,prime),pow(g,12,prime),pow(g,12*12,prime),m,prime) #␣
 ↪even even == even
assert oracle_dh(pow(g,11,prime),pow(g,12,prime),pow(g,11*12,prime),m,prime) #␣
 ↪odd even  == even
assert not oracle_dh(pow(g,11,prime),pow(g,12,prime),pow(g,13,prime),m,prime)#␣
 ↪odd even  != odd
assert not oracle_dh(pow(g,11,prime),pow(g,11,prime),pow(g,12,prime),m,prime)#␣
 ↪odd odd   != even
```

[ ]:
```
from pwn import *
import re
```

[ ]:
```
r = remote("a4d62f4a3c81d6a95e7c3741-casino.challenge.master.cscg.
 ↪live",port=31337,ssl=True)
```

```
[x] Opening connection to
a4d62f4a3c81d6a95e7c3741-casino.challenge.master.cscg.live on port 31337
[x] Opening connection to
a4d62f4a3c81d6a95e7c3741-casino.challenge.master.cscg.live on port 31337: Trying
2604:1380:4602:4000::1
[x] Opening connection to
a4d62f4a3c81d6a95e7c3741-casino.challenge.master.cscg.live on port 31337: Trying
2604:1380:4602:4000::1
[+] Opening connection to
a4d62f4a3c81d6a95e7c3741-casino.challenge.master.cscg.live on port 31337: Done
```

[ ]:
```
counter = 100
while 0 <= counter < 200:
    r.recvuntil(b"\nGuess the random bit I have coosen!\n")
    data = r.recvline()
    ga,gb,c = list(map(int,re.search("Commitment: (\d+), (\d+), (\d+)",data.
 ↪decode("ASCII")).groups()))
    r.sendline(b"1" if oracle_dh(ga,gb,c,m,prime) else b"0")

    if b"Wrong" in r.recvuntil(b"!"):
        counter -=1
    else:
        counter +=1
    print(counter)

    if counter == 200:
        r.recvall()

    r.recvuntil("€\n".encode())
```

[ ]:
```
r.recvall()
```

```
[x] Receiving all data
[x] Receiving all data: 1.29KB
[+] Receiving all data: Done (1.29KB)
[*] Closed connection to
a4d62f4a3c81d6a95e7c3741-casino.challenge.master.cscg.live port 31337
```

[ ]:
```
b' My challenge was 1\nProof: 211127177783554896404050504170232092009695854588350876812828167350534862202563100751401261477527338852948793764542083279045231162346837988064187926290214827325442752900242025083989280886678844567609035382332360089882888310974907754134050889021719286343158553557272239055279701227827400044009029259263044993174083385924799127875447367601740970770160260136247790672012557204232209414924515554768365700878190775899308118714245879695452584356320946880753074181295975386259939030401690855700614789016866652531044840640786620777338919431324391732910916867317673172708613193638877917334860623318045895853449590511597961633628, 212196335364551737981443179767130997641803896967427581479193828435384778015420918938455319797008723538715097831487815222163433771426842811359392363531 3
```

```
737794201503527404973655017233941452399283942648047959432447166894949329116072567636345955805754543357502268171918791993697168767182533852960521937491294432153509134365013036286528557675271457762587657454294149361941115314849208123737109931750940359878764601463856526282292462853196633908217189991967834139272968472386324919725437575748412722525818308982882593497330759680059749874802057799539907186576454263866043329215643716016596430583909539672503434411743649260\nCSCG{I_should_have_used_prime_order_groups_instead}\n'
```

1 PSA

Use subgroups with a prime order because a normal group can never have a prime order (it's always $p-1$ then). Therefore, if the challenge would use the subgroup with order m (from above), I would never get a flag.