Diffie-Hellman - Static Client 2

Dawid Karpiński

04.12.2024 r.

Challenge działa podobnie do Static Client 1, natomiast tym razem Bob weryfikuje wartości `g`. Jedynym parametrem, który można wykorzystać do przeprowadzenia ataku jest `p`. W tym celu, można wygenerować taką liczbę, aby była łatwa do rozwiązania w problemie logarytmu dyskretnego.

W przypadku ataku Pohlig-Hellman, będzie to liczba gładka. Definiuje się je jako liczbę pierwszą, która ma dzielniki mniejsze od danej liczby k.

Zatem, rozwiązanie challenge'u sprowadza się do następujących kroków:

1. Wygenerowanie gładkiej liczby pierwszej, o tej samej długości co oryginalny `p`
2. Wysłanie odpowiedzi do Boba z nowym parametrem `p'`
3. Obliczenie logarytmu dyskretnego poprzez Pohlig-Hellman

Generowanie gładkiej liczby pierwszej

Algorytm, który znajdzie taką gładką liczbę, o zbliżonej długości bitów do tej użytej przez Alice:

```
def smooth_p(length: int):
    mul = 1
    i = 1
    while True:
        mul *= i
        if (mul + 1).bit_length() >= length and isprime(mul + 1):
            return mul + 1
        i += 1
```

Podmienienie parametru `p`

W wiadomości zwrotnej do Bob'a wysłano oryginalne `g` i `A`, natomiast podmieniono parametr `p` na ten wygenerowany.

```
s_p = smooth_p(p.bit_length())

msg = json.dumps({"p": hex(s_p), "g": hex(g), "A": hex(A)})
sock.send(msg.encode("utf-8"))
```

Obliczenie logarytmu dyskretnego

Ostatnim krokiem jest rozwiązanie problemu dyskretnego logarytmu. Ze względu na to, że użyty `p` jest liczbą gładką, nie wymaga on dużo czasu by znaleźć wynik.

```
# obliczenie klucza prywatnego
b = discrete_log(s_p, B, g)
```

Na koniec pozostało użyć znaleziony klucz prywatny do otrzymania flagi.

```
sha1 = hashlib.sha1()
sha1.update(str(pow(A, b, p)).encode("ascii"))
key = sha1.digest()[:16]

cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = cipher.decrypt(encrypted)

print(plaintext.decode("utf-8"))  # crypto{uns4f3_pr1m3_sm4ll_oRd3r}
```