Diffie-Hellman Man-In-The-Middle, level 2

> Author : jul0105
> Date : 06.04.2021

Challenge info

Release : Bundle 3 (27.03)

Difficulty : Hard

Goal :

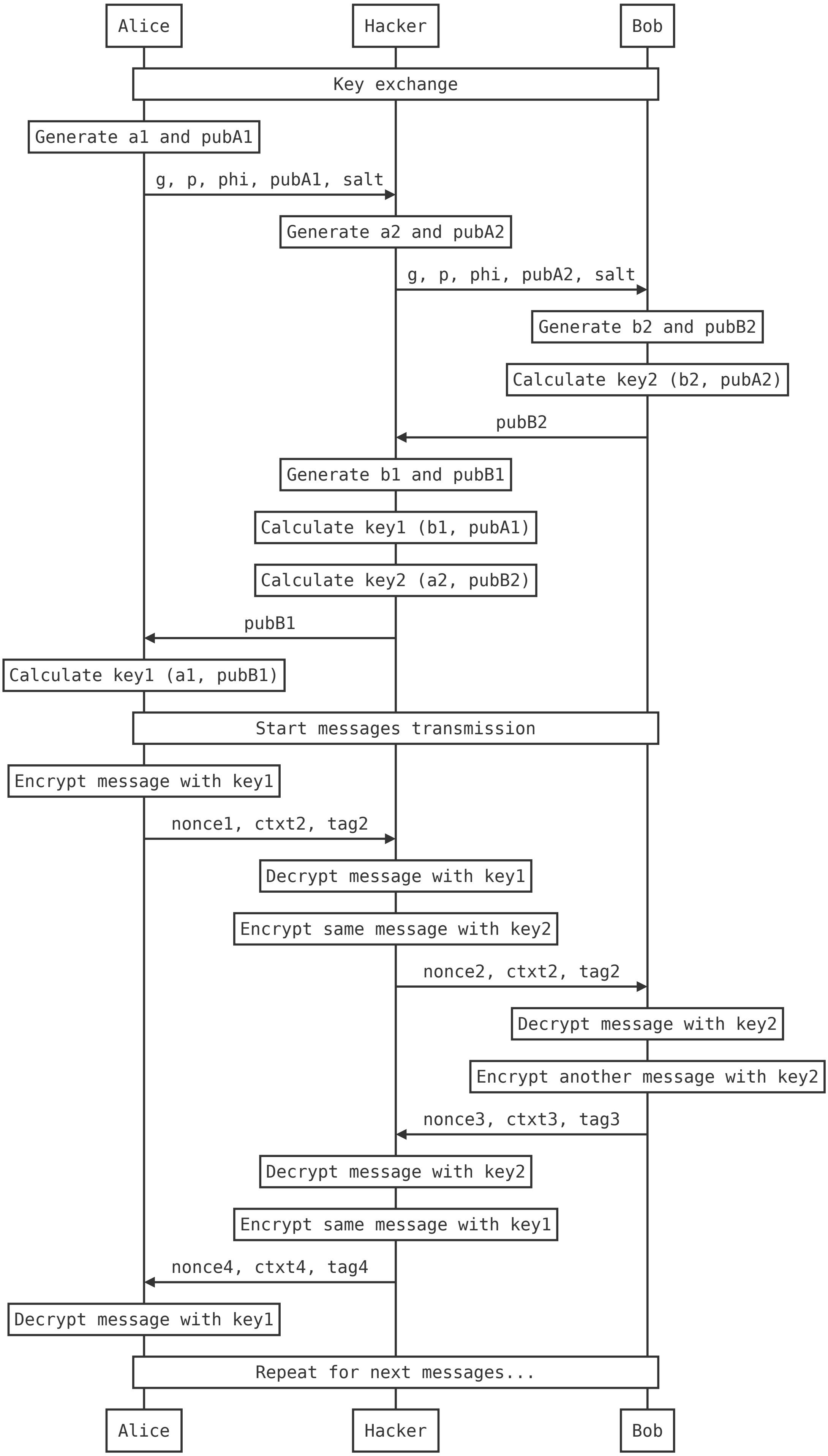
- Compromise the key exchange between Alice and Bob.
- Retrieve the flag from Alice and decrypt it.
- Submit flag and write-up (see below)

Important: This write-up was written on Typora using non-standard markdown features. Please open the PDF file if you don't view this using Typora.

This challenge was very interesting. Unfortunately, I made a mistake early on the solving that made me look to wrong paths for a long time.

Solve

1. Firstly, here is how the exchange are operated between Alice, Bob and the Hacker on the first level of this challenge (no change since last time) :


Alice
Hacker
Bob
Key exchange
Generate a1 and pubA1
g, p, phi, pubA1, salt
Generate a2 and pubA2
g, p, phi, pubA2, salt
Generate b2 and pubB2
Calculate key2 (b2, pubA2)
pubB2
Generate b1 and pubB1
Calculate key1 (b1, pubA1)
Calculate key2 (a2, pubB2)
pubB1
Calculate key1 (a1, pubB1)
Start messages transmission
Encrypt message with key1
nonce1, ctxt2, tag2
Decrypt message with key1
Encrypt same message with key2
nonce2, ctxt2, tag2
Decrypt message with key2
Encrypt another message with key2
nonce3, ctxt3, tag3
Decrypt message with key2
Encrypt same message with key1
nonce4, ctxt4, tag4
Decrypt message with key1
Repeat for next messages...
Alice
Hacker
Bob

2. Now, this second challenge adds a Trusted Third Party that is used to validated that Alice's key and Bob's key are the same.

3. Since K_A and K_B has to be the same, the simplest way to guarantee this equality is to set private parameter a' et b' to 0.
4. Here are some calculation :
    1. To communicate with Bob, we pick: $a' = 0$
    2. Then calculate and send to Bob: $A' \equiv g^{a'} \equiv g^0 \equiv 1 \pmod p$
    3. Bob pick b and send B back.
    4. Our shared key between Hacker and Bob is $K_B \equiv g^{a'b} \equiv g^{0b} \equiv 1 \pmod p$
5. Same thing between Alice and Hacker :
    1. To communicate with Alice, we pick: $b' = 0$
    2. Then calculate and send to Alice: $B' \equiv g^{b'} \equiv g^0 \equiv 1 \pmod p$
    3. Our shared key between Hacker and Bob is $K_A \equiv g^{ab'} \equiv g^{0a} \equiv 1 \pmod p$
6. Now we have $K_A = K_B = 1$ so the key exchange is validated by the trusted third party
7. Finally, we just need to send `Please` to Alice for her to send the UUID :

```
74c34938-959b-4c0d-9dbb-c20f8ef4f38a
```

Questions

Which flaw in the protocol did you exploit?

As explained before, by picking 0 as a private parameter, we can completely void the other party's private parameter, due to the property of 0 in multiplication.

This is the only value in $\mathbb{Z}_p$ that can be used to calculate K without knowing the other party's private parameter.

How can this flaw be mitigated?

It shouldn't be allowed to pick 0 as a private parameter because it completely break the system. Luckily, it is easily avoidable because the public parameter associated will always be equal to 1.

So when a party receive a public parameter that is equal to 1, it should refuse it.

Another solution would be to refuse K = 1.

Full scripts

Here are the full python scripts. They don't have changed much since the last level. For more details on the scripts, refer to first level's write up.

dh_functions.py :

```python
import base64
from Crypto.Protocol.KDF import scrypt
from Crypto.Cipher import AES
from random import randrange

AES_KEY_LEN = 16

def generate_param(g, p, q):
    #a = randrange(q)
    a = 0 # Lvl2
    pubA = pow(g, a, p)
    return (a, pubA)

def generate_key(a, pubB, p):
    return pow(pubB, a, p)

def encrypt(dh_key, salt, msg, p):
    key_bytes   = dh_key.to_bytes((p.bit_length() + 7) // 8, "big")
    aes_enc_key = scrypt(key_bytes, salt, AES_KEY_LEN, N=2**14, r=8, p=1)
    cipher      = AES.new(aes_enc_key, AES.MODE_GCM)
    ctxt, tag   = cipher.encrypt_and_digest(msg)
    return ctxt, tag, cipher.nonce

def decrypt(dh_key, salt, ciphertext, tag, nonce, p):
    key_bytes   = dh_key.to_bytes((p.bit_length() + 7) // 8, "big")
    aes_enc_key = scrypt(key_bytes, salt, AES_KEY_LEN, N=2**14, r=8, p=1)
    cipher      = AES.new(aes_enc_key, AES.MODE_GCM, nonce=nonce)
    plaintext   = cipher.decrypt_and_verify(ciphertext, tag)
    return plaintext
```

solve.py :

```
# pip install websockets

import asyncio
import websockets
import time
import json
import base64
from urllib.parse import quote
import dh_functions

debug = False

domain = "c02e7e85-ed56-410b-92e3-466a49b9d01c.idocker.vuln.land"
base_url = "wss://" + domain + "/api/deploy/"

async def read_from_ws_exec(task, force_debug=False):
    uri = base_url + task
    if debug or force_debug:
        print(uri)
    res = ''
    async with websockets.connect(uri) as websocket:
        try:
            while True:
                val = await websocket.recv()
                res += val + '\n'
                if debug or force_debug:
                    print(f"< {val}")
        except websockets.exceptions.ConnectionClosed:
            if debug or force_debug:
                print("CLOSED")
            return res
```

```python
async def read_from_ws(task, force_debug=False):
    res = await read_from_ws_exec(task, force_debug)

    if 'been detected!' in res or 'noticed suspicious behaviour' in res:
        print("FAILED")
        exit()

    if debug:
        print("\n\n")

    time.sleep(0.01)
    return res


async def main():
    print("[+] Reset")
    await read_from_ws("") # Reset
    time.sleep(1)


    print("[+] Init...")
    await read_from_ws("task") # Init
    time.sleep(1)


    # Intercept Alice -> Bob (g, p, q, A, salt)
    print("[+] Intercept Alice ---(g, p, phi, pubA, salt)--> Bob")

    res = await read_from_ws("task?argument=1") # Intercept
    obj = extract_data(res)
    g = obj['g']
    p = obj['p']
    phi = obj['phi']
    pubA = obj['pubA']
    salt = obj['salt']
    g_dec = base64_to_int(g)
    p_dec = base64_to_int(p)
    phi_dec = base64_to_int(phi)
    pubA_dec = base64_to_int(pubA)
    salt_dec = base64.b64decode(salt)
    print('g_dec:', g_dec)
    print('p_dec:', p_dec)
    print('phi_dec:', phi_dec)
    print('pubA_dec:', pubA_dec)
    print('salt_dec:', salt_dec)


    # Generate ax and Ax
    print("[+] Generate corrupted ax and pubAx")
    ax_dec, pubAx_dec = dh_functions.generate_param(g_dec, p_dec, phi_dec)
    pubAx = int_to_base64(pubAx_dec, 256)
    print('ax_dec:', ax_dec)
    print('pubAx_dec:', pubAx_dec)
    print('pubAx:', pubAx)


    # Hacker -> Bob (g, p, q, Ax, salt)
    print("[+] Drop package and craft a new package for Bob")
    await read_from_ws("task?argument=2") # Drop package
    await read_from_ws("task?argument=2") # Insert package

    await read_from_ws("task?argument=Alice") # Sender
    await read_from_ws("task?argument=Bob") # Receiver


    print("[+] Hacker ---(g, p, phi, pubAx, salt)---> Bob")
    o = json.dumps({'g': g, 'p': p, 'phi': phi, 'pubA': pubAx, 'salt': salt})
    await read_from_ws("task?argument=" + quote(o)) # Content


    # Hacker <- Bob (B)
    print("[+] Intercept Alice <---(pubB)--- Bob")
    res2 = await read_from_ws("task?argument=1") # Intercept package
    obj2 = extract_data(res2)
    pubB = obj2['pubB']
    pubB_dec = base64_to_int(pubB)
    print('pubB_dec:', pubB_dec)


    # Generate bx and Bx
    print("[+] Generate corrupted bx and pubBx")
    bx_dec, pubBx_dec = dh_functions.generate_param(g_dec, p_dec, phi_dec)
    pubBx = int_to_base64(pubBx_dec, 256)
    print('bx_dec:', bx_dec)
    print('pubBx_dec:', pubBx_dec)


    # Alice <- Hacker (Bx)
    print("[+] Drop package and craft a new package for Alice")
    await read_from_ws("task?argument=2") # Drop package
    await read_from_ws("task?argument=2") # Insert package

    await read_from_ws("task?argument=Bob") # Sender
    await read_from_ws("task?argument=Alice") # Receiver


    print("[+] Alice <---(pubBx)--- Hacker")
    o = json.dumps({'pubB': pubBx})
    await read_from_ws("task?argument=" + quote(o)) # Content


    print("[+] Calculate Alice's key")
    key_alice = dh_functions.generate_key(bx_dec, pubA_dec, p_dec)


    print("[+] Calculate Bob's key")
    key_bob = dh_functions.generate_key(ax_dec, pubB_dec, p_dec)

    for j in range(3):
        print("\n[#] Starting loop " + str(j) +"\n")
        from_alice = True
        if j % 2:
            from_alice = False

        if from_alice:
            decryption_key = key_alice
            encryption_key = key_bob
        else:
            decryption_key = key_bob
            encryption_key = key_alice


        # Intercept
        if from_alice:
            print("[+] Intercept Alice ---(nonce, ctxt, tag)--> Bob")
        else:
            print("[+] Intercept Alice <--(nonce, ctxt, tag)--- Bob")
        res3 = await read_from_ws("task?argument=1")
        obj3 = extract_data(res3)
        nonce = obj3['nonce']
        ctxt = obj3['ctxt']
        tag = obj3['tag']
        nonce_dec = base64.b64decode(nonce)
        ctxt_dec = base64.b64decode(ctxt)
        tag_dec = base64.b64decode(tag)


        # Decrypt message
        print("[+] Decrypt message")
        msg1 = dh_functions.decrypt(decryption_key, salt_dec, ctxt_dec, tag_dec,
nonce_dec, p_dec)
        print('   [*] message: ', msg1)


        if j == 1:
            msg1 = b'Please' # Lvl2
            print('   [*] changed message: ', msg1)


        print("[+] Encrypt message")
        ctxt_dec, tag_dec, nonce_dec = dh_functions.encrypt(encryption_key,
salt_dec, msg1, p_dec)
        ctxt = base64.b64encode(ctxt_dec).decode('ascii')
        tag = base64.b64encode(tag_dec).decode('ascii')
        nonce = base64.b64encode(nonce_dec).decode('ascii')

        if from_alice:
            print("[+] Drop package and craft a new package for Bob")
        else:
            print("[+] Drop package and craft a new package for Alice")
        await read_from_ws("task?argument=2") # Drop package
        await read_from_ws("task?argument=2") # Insert package

        if from_alice:
            await read_from_ws("task?argument=Alice") # Sender
            await read_from_ws("task?argument=Bob") # Receiver
        else:
            await read_from_ws("task?argument=Bob") # Sender
            await read_from_ws("task?argument=Alice") # Receiver


        if from_alice:
            print("[+] Hacker ---(nonce, ctxt, tag)--> Bob")
        else:
            print("[+] Hacker <--(nonce, ctxt, tag)--- Bob")
        o = json.dumps({'nonce': nonce, 'ctxt': ctxt, 'tag': tag})
        await read_from_ws("task?argument=" + quote(o)) # Content



def extract_data(string):
    lines = string.splitlines()
    for line in lines:
        if line[:6] == 'Data: ':
            return json.loads(line[6:])

def base64_to_int(b):
    return int.from_bytes(base64.b64decode(b), byteorder='big', signed=False)

def int_to_base64(i, bit_length):
    return base64.b64encode(i.to_bytes(bit_length, byteorder='big',
signed=False)).decode('ascii')

asyncio.get_event_loop().run_until_complete(main())
```