Diffie-Hellman Man-In-The-Middle, level 1

Author : jul0105
Date : 13.03.2021

Challenge info

Release : Bundle 1 (01.03)

Difficulty : Medium

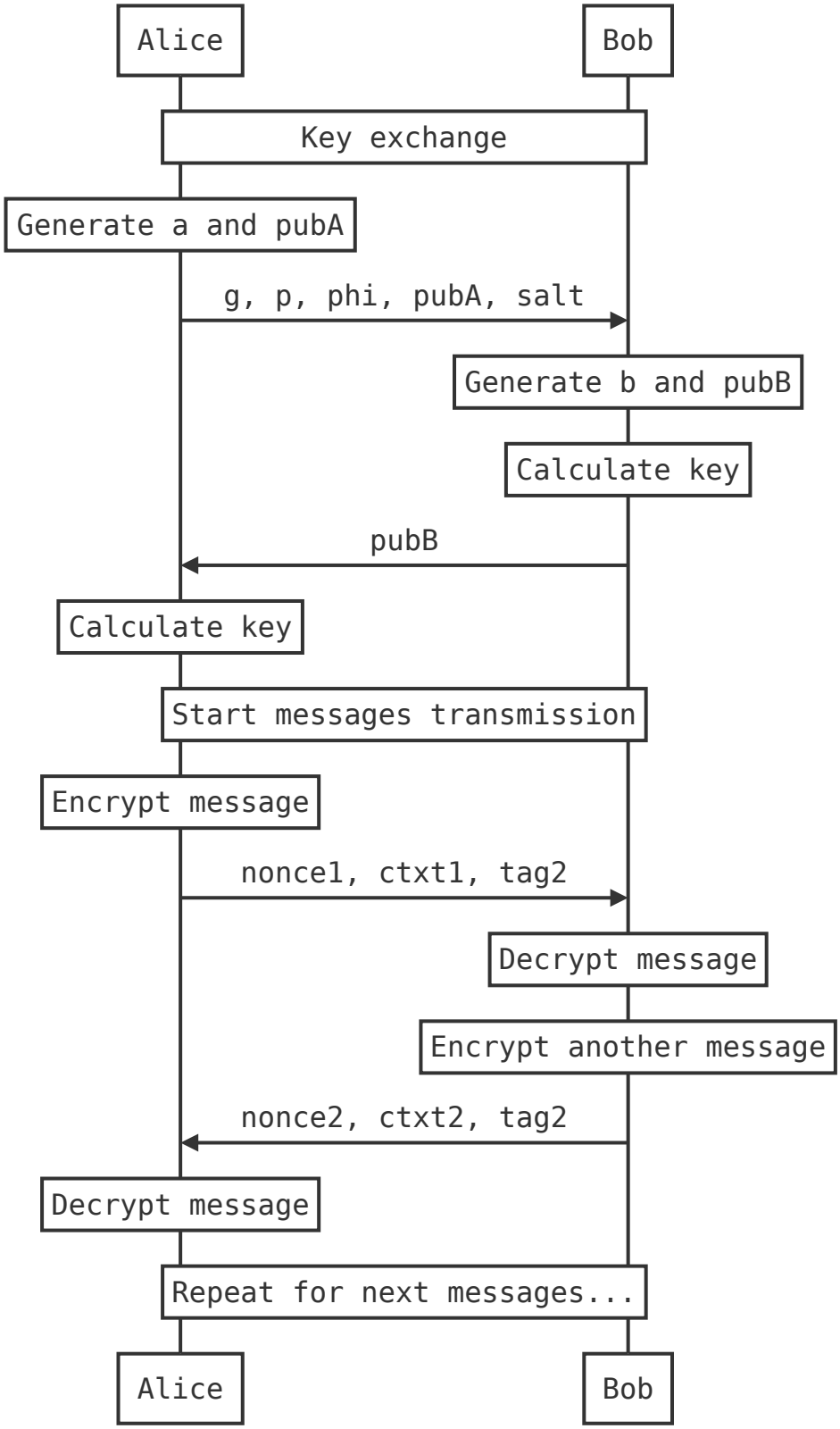
Goal :

- Compromise the key exchange between Alice and Bob.
- Decrypt the flag sent by Alice.
- Submit flag and write-up (see below)

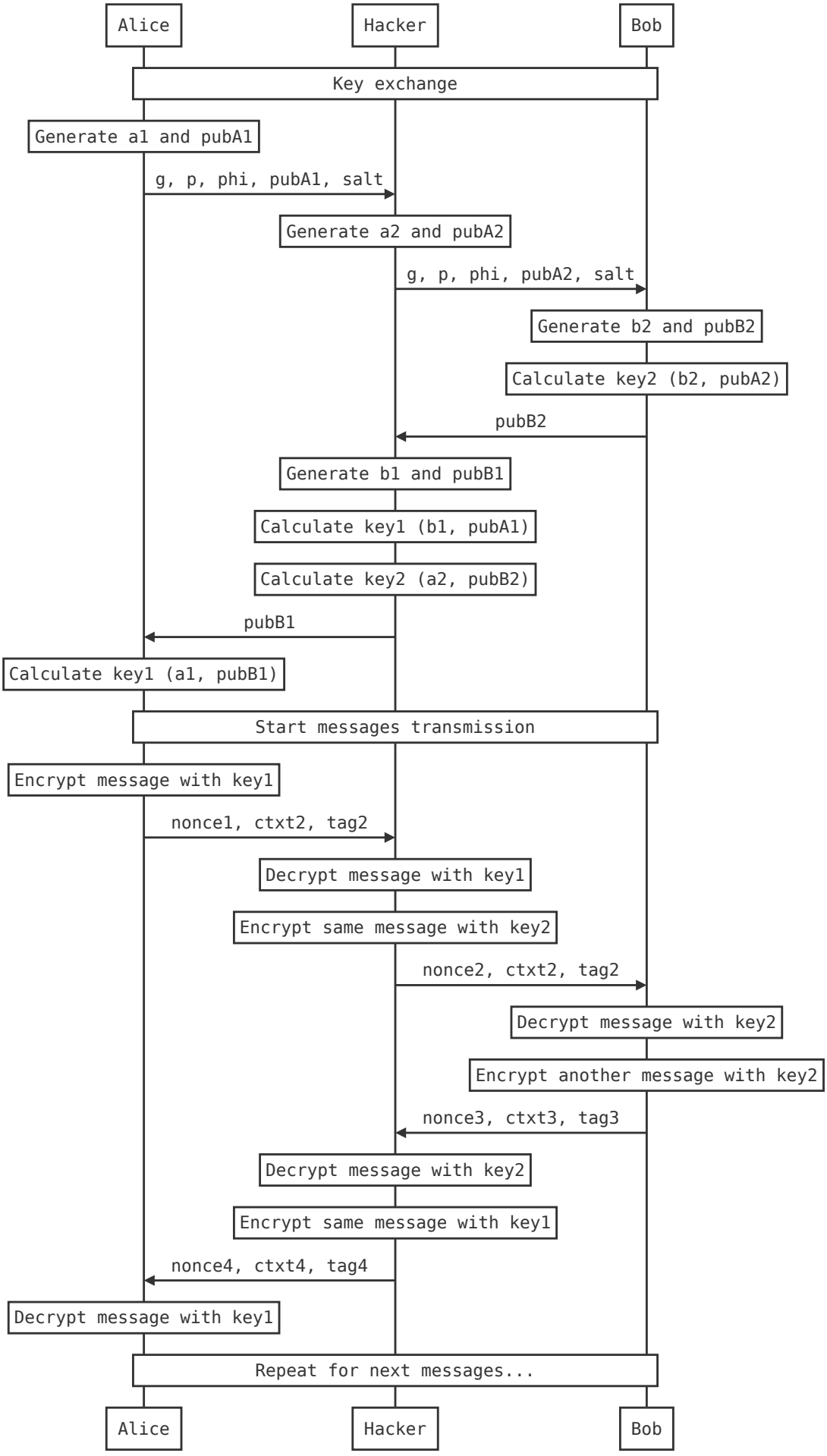
Important: This write-up was written on Typora using non-standard markdown features. Please open the PDF file if you don't view this using Typora.

Solve

1. Firstly, this is how the initial transmissions are made between Alice and Bob :



2. And this is how we, as a hacker, want to change the transmissions to be the Man In The Middle :



To summary this diagram, we need to **intercept every transmission** between Alice and Bob, **drop the package and craft a new one**.

3. To be able to craft valid package we need 4 functions :
    1. one to generate Diffie Hellman parameters a and pubA or b and pubB
    2. one to calculate the key
    3. one to encrypt
    4. and one to decrypt

```
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)
    pubA = pow(g, a, p)
    return (a, pubA)

def generate_key(priv, pub, p):
    return pow(pub, priv, 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
```

4. Then, I used the python package `websockets` to be able to communicate with the challenge from a script. For every step, a websocket is opened on the following endpoint :

```
wss://<UUID>.idocker.vuln.land/api/deploy/task?argument=<arg>
```

Where **<arg>** is the value to send to the challenge.

5. Now that we have implemented the necessary functions to craft package and communication with the challenge, it's time to script the behavior explained in the second diagram. This means intercept and modify key exchange transmission between Alice and Bob. Calculate each key pair. And then intercepts all followings messages to re-encrypt it with the appropriate key.

    For more details, see the full scripts at the end of this write-up

Questions

Which flaw in the protocol did you exploit?

The key exchange transmission doesn't guarantee authentication nor integrity. This means that when Alice send pubA to Bob, Bob has no way to be assured that pubA really originate from Alice, and that pubA has not been modified by a third party.

How can this flaw be mitigated?

It is necessary to use an authenticated canal to transmit the key exchange.

This can be done signing the key exchange package (pubA and pubB) using a public/private key pair certified by a Certificate Authority trusted by Alice and Bob.

This certificate allow each party to prove the ownership of the public key. So with the signature of the key exchange packages, Bob can be assured that the package really come from Alice and that the package has not been modified.

Full scripts

dh_function.py :

```
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)
    pubA = pow(g, a, p)
    return (a, pubA)

def generate_key(priv, pub, p):
    return pow(pub, priv, 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
```

```python
import json
import base64
from urllib.parse import quote
import dh_functions

debug = False

domain = "6338c24a-b602-4496-b75e-4e6fff0db695.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

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

    if 'been detected!' in res:
        print("FAILED")
        exit()

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

    time.sleep(1)
    return res


async def main():
    print("[+] Reset")
    await read_from_ws("") # Reset


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


    # 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)


# 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)


# 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)


# 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)


# 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(20):
        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)


        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())
```