

DAT510 - Assignment 2

Frøydís Jørgensen

October 11, 2020

Abstract

A one-paragraph summary of the entire assignment - your choices of cryptographic primitives and their parameters, procedure, test results, and analysis.

Introduction

A description of the scientific background for your project, including previous work that your project builds on. (Remember to cite your sources!) The final sentence (analogous to the thesis statement in a term paper) is the objective of your experiment.

Design and Implementation

In this section I am going to go in details on how I implemented each step of the project and explain how it works.

Part 1

The first part of this assignment required making a secure communication scenario by following 8 steps. The first step is to decide on some global parameters for a Diffie-Hellman-like key exchange. So Alice and Bob agree on a cyclic group of order p which had to be a prime as $2q + 1$. This is a so called safe prime and we get it if we take a Sophie Germain prime as q , and we get a new prime. So in this case I used the Sophie Germain prime 359. $2 \cdot 359 + 1 = 719$. So the shared prime number between Alice and Bob is 719. We can therefore say that the cyclic group is \mathbb{Z}^*_{719} , which means that after the operations done by Diffie-Hellmann key exchange we will end up with a number between $g = 2$, which was predefined as the generator, and $p = 719$. Figure 1 shows the start at my main function where the generator g and the shared prime is defined.

```
def main():
    # the generator g is pre defined as 2
    g = 2
    # The shared prime is a Safe prime, decided by 2q+1, where q is a Sophie Germain prime
    sharedPrime = 719
    print("The shared prime is ", sharedPrime, '. Which gives us the cyclic group: Z*', sharedPrime)
    print("The generator (g) is ", g)
```

Figure 1: Start of my main function, defined g and the shared prime

The second step is to follow the Diffie-Hellman's key exchange scheme and create Alice and Bobs key pair. For their private key I just chose a random number, so Alice's private key is 217 and Bob's private key is 131. The only requirements for the private keys is that they are smaller than their shared prime and a positive natural number. To create the public key from the private key I created a method, which is shown in Figure 2. To create the public key, the function does this operation:

$$g^{\text{privateKey}} \bmod \text{sharedPrime}$$

```
def publicKey(privateKey, g, prime):
    return g**privateKey % prime
```

Figure 2: The function to create the public key

Figure 3 shows how the private and public key are defined for Alice and Bob.

```
# Alice and Bob's private keys which is secret.
alicePrivate = 217
bobPrivate = 131
print("Alice's private key is ", alicePrivate)
print("Bob's private key is ", bobPrivate)
# Creates the public key which can be sent over to the other part.
alicePublic = publicKey(alicePrivate, g, sharedPrime)
bobPublic = publicKey(bobPrivate, g, sharedPrime)
print("Alice's public key is ", alicePublic)
print("Bob's public key is ", bobPublic)
```

Figure 3: Defines private key and calls a function to create the public key

The third step is to send the public key to the part you want to connect with. Since this is a staged scenario and both Alice and Bob runs on the same program this is not necessary. But we will take a closer look into this step in part 2.

The fourth step is to create a shared key. The shared key is created as shown in Figure 4 by doing this operation:

$$publicKey^{privateKey} \bmod sharedPrime$$

```
def sharedKey(publicKey, privateKey, prime):  
    return publicKey**privateKey % prime
```

Figure 4: The function to create the shared secret key

Both Alice and Bob should now have the same shared key, which is a secret and should not be shared with anyone else. As shown in Figure 5, I check that the shared keys are the same for both Alice and Bob. If they do not have the same number something wrong happened along the way and they do not have a connection.

```
aliceShared = sharedKey(bobPublic, alicePrivate, sharedPrime)  
bobShared = sharedKey(alicePublic, bobPrivate, sharedPrime)  
if aliceShared == bobShared:  
    print("Alice's shared key is ", aliceShared)  
    print("Bob's shared key is ", bobShared)  
else:  
    print("Something went wrong")
```

Figure 5: Calls the function to get the shared key, and check if Alice and Bob have the same key

At this point, we are done with the Diffie-Hellman's key exchange, but since Alice and Bob are concerned about the strength of the shared key, they want to use a cryptographically strong pseudo-random number generator (CSPRNG). I choose to use pseudo-random number generator named Blum Blum Shub (BBS) in this fifth step. Blum Blum Shub takes the form:

$$x_{n+1} = x_n^2 \bmod M$$

x_0 is called the seed, which in this case is the shared secret key. M is $p \cdot q$ where both q and p is a prime number and both are congruent to 3 mod 4. Both 7 and 11 fulfills that, and therefore I chose my M to be $7 \cdot 11 = 77$. This method is based on taking the least significant bit for each operation, as shown in Figure 6. On the way we have to check that the seed we start with or the new seeds that we creates not are 0 or 1. We do also have to check that the seed values not share factors with either p or q , Ref [1].

```
def CSPRNG_BBS(seed, size):
    p = 7
    q = 11
    M = p*q
    bits = ""
    for _ in range(size):
        if seed < 2:
            return "The seed can't be 0 or 1"
        factorials = []
        for i in range(1, seed + 1, 1):
            if seed % (i) == 0:
                if i == p or i == q:
                    return "p and q can not be factors of the seed"
        seed = (seed**2) % M
        b = seed % 2
        bits += str(b)
    return bits
```

Figure 6: The function that performs Blum Blum Shub

As shown in Figure 7 we uses the CSPRNG and creates a secret key which is of length 10 bits, but I convert it into a decimal. The reason why I chose a bit length of 10 is because In the next step we are going to encrypt a text and send it to Bob. For the encryption I chose to use SDES from the previous assignment. This is not a secure encryption, but since encryption is not the focus in this assignment I used SDES instead of importing something more secure. SDES encryption takes in a 10 bit long key and encrypts 8 bit blocks at a time. As shown in Figure 7 you get the choice of writing your own message or using a predefined message. I convert the message into bits and split it up in 8 bit blocks. Then the program goes through the list of 8

bit block and encrypt them using our 10 bit key.

```
secretKeyBit = CSPRNG_BBS(aliceShared, 10)
secretKey = int(secretKeyBit, 2)
print("Stronger secret key is ", secretKey)
yesOrNo = input("Do you want to send a predefined message? y/n: ")
if yesOrNo == 'y' or yesOrNo == 'Y' or yesOrNo == 'yes':
    message = "This is a super secret message"
else:
    message = input("Write your own message to Bob: ")
print("Alices message is: ", message)
# Converts the message into bits
messageBits = ' '.join(format(ord(x), 'b').zfill(8) for x in message).split(' ')
encrypted = []
# Encrypts the message, by using SDES from Assignment 1.
# The key used in the encryption is the secret key we got from diffie-hellman and BBS.
for i in range(0, len(message)):
    encrypted.append(SDES(messageBits[i], secretKeyBit))
print("Encrypted text from Alice to Bob: ", encrypted)
```

Figure 7: Calls Blum Blum Shub and SDES to secure the key and encrypt a text

The next step is for Bob to decrypt the message from Alice. At this point we already know that Alice and Bob got the same shared key and used the same BBS. So as shown in Figure 8, we take in 8 bit blocks from the cipher and decrypt it using the same key, which will give us the message from Alice in plaintext.

```
decrypted = ''
# Decrypts the message using the same key
for i in range(0, len(encrypted)):
    decrypted += frombits(SDES(encrypted[i], secretKeyBit, True))
print("The message that Bob gets is: ", decrypted)
```

Figure 8: Calls SDES to decrypt the cipher from Alice

I have now shown a secure communication scenario between Alice and Bob. If someone else would have gotten the cipher, they could not have deciphered it, because the key was created from both Alice's private key and Bob's private key and their shared prime, which is a secret.

Part 2

Test results

Results of testing the software, as you observed/recorded them. Note that this section is only for observations you make during testing. Your analysis belongs in the Discussion section.

Discussion

Your analysis of what your testing results mean, and your analysis.

Conclusion

A short paragraph that restates the objective from your introduction and relates it to your results and discussion, and describes any future improvements that you would recommend. Works Cited A bibliography of all of the sources you got information from in your report.

References

- [1] A Security site, *Explains how Blum Blum Shub works*, <https://www.asecuritysite.com/encryption/blum>.