# Applied Cryptography

**Public Key Cryptography, Assignment 4, Monday, April 8, 2024**

**Remarks:**

- Hand in your answers through Brightspace.
- Hand in format: PDF. Either hand-written and scanned in PDF, or typeset and converted to PDF. Please, **do not** submit photos, Word files, LaTeX source files, or similar.
- Assure that the name of **each** group member is **in** the document (not just in the file name).

**Deadline:** Sunday, April 21, 23.59

**Goals:** After completing these exercises you should have understanding in the security of public key encryption and hash-and-sign digital signatures.

In exercises 1 and 3, you will be required to run a command in your terminal from within the Radboud network. You can access the Radboud network from outside of the university by either using eduVPN or connecting through lilo, by running the following command in your terminal:

```
ssh [username]@lilo.science.ru.nl
```

where your username/password combination are your Science credentials (not your "general" Radboud credentials). If you have never used lilo before, you might have to request access from the RU helpdesk. You can do so by email.

1. **(20 points)** Alice has made a script that she plans to use to transfer symmetric keys with her friend Bob. It is given in Appendix A. At a high level, she picks a random string of length 128 bits as the key, encrypts it, and sends the encryption to Bob. On the other end, Bob decrypts to obtain the key.

   Alice knows that such systems must not only be secure against eavesdropping, but also in the case that an adversary is allowed to make decryption queries. To test her cryptosystem, Alice will play the following game:

   - First, Alice shows you a valid encryption of the key she is going to send Bob.
   - Second, Alice allows you to make a decryption query of a ciphertext of your choice.

   In order to play the game, connect to the Radboud network and run the command:

   ```
   nc appliedcrypto.cs.ru.nl 4143
   ```

   The file `interaction_example.py`, which you can find on Brightspace, has useful functions that you will need. For example: a hash function for exercise 1g, or the modulus and public key used in exercise 1b.

   (a) Analyze closely the given script. Which cryptosystem is Alice using in her script?

   (b) Can you recover the key that Alice sent to Bob? Assuming you can, write down the details of how you did it.

   (c) What flaws of the cryptosystem did you exploit to find the solution?

   (d) There is one security property that Alice's cryptosystem is missing, that allows the attack to work. Name it.

   (e) Suggest how Alice can fix her cryptosystem in such a way that it remains secure, and your attack no longer works.

(f) Alice can use the FO (Fujisaki-Okamoto) second transform to turn her cryptosystem into a KEM. Explain what KEMs are and how they differ from public-key encryption?

(g) Modify Alice's script to include the FO second transform. Write the script and test it on your own computer. You will need to additionally implement key generation and the decryption oracle. For the encoding you can use your own ideas for generating deterministically a key from one's name. You should submit your script together with the rest of the answers (in the pdf or separately).

(h) Construct 1000 decryption oracle queries (more accurately, decapsulation oracle queries) at random and test for how many of them you can get an answer from the decryption oracle.

(i) Apply your attack from exercise 1b (but this time you are allowed 1000 queries) and write down whether it succeeded.

2. **(15 points)** Consider the following cryptosystem:

**KeyGen**:

(a) Choose a random prime $p$

(b) Choose a random multiplicative generator $g$ of $\mathbb{Z}_p^*$

(c) Choose a random $x \xleftarrow{\$} \mathbb{Z}_{p-1}$ coprime with $p-1$, that is, $\gcd(x, p-1) = 1$

(d) Compute $y \leftarrow g^x \pmod p$

(e) Set public key $\mathsf{pk} = y$ and private key $\mathsf{sk} = x$ and output public parameters $(p, g)$

**Encrypt**: To encrypt a message $0 < M < p$:

(a) Choose a random $k \xleftarrow{\$} \mathbb{Z}_{p-1}$

(b) Compute ciphertext pair $(C_1, C_2)$ as

$$\begin{aligned} C_1 &\leftarrow g^k \pmod p \\ C_2 &\leftarrow y^k M \pmod p \end{aligned}$$

**Decrypt**: Decrypt ciphertext as $M \leftarrow C_2 \cdot C_1^{-x} \pmod p$

(a) Show that $C_2$ is uniformly distributed over $\mathbb{Z}_p^*$.

(b) Show that the cryptosystem is OW-CPA secure if the computational Diffie-Hellman (CDH) problem is hard.

(c) In practice, in order to improve the efficiency, instead of using a generator of $\mathbb{Z}_p^*$, one can use a $g$ of much smaller order than $p$. Show that in such a case, if a message $M$ is not in the subgroup generated by $g$, the scheme is vulnerable to a meet-in-the-middle attack (see the meet-in-the-middle attack in the lecture slides).

Consider now a related digital signature scheme defined as follows:

**KeyGen**: Same as in previous cryptosystem
**Sign**: To sign a message $M$

(a) Find a pair $(r, s)$ such that $g^M = y^r \cdot r^s \pmod p$

(b) Output $(r, s)$ as the signature of $M$

**Verify**: In order to verify a signature $(r, s)$ of $M$, check that $r \in \langle g \rangle$, $s < p$ and the validity of the equation $g^M = y^r \cdot r^s \pmod p$

(d) Write down the procedure of finding $(r, s)$, i.e., the exact steps of how the signature can be generated.

(e) Show that there exists an existential forgery attack on the cryptosystem.

3. **(15 points)** Consider the function $\chi_p : x \mapsto x^{\frac{p-1}{2}}$ for an integer $x \in \mathbb{Z}_p$, where $p$ is an odd prime number.

   (a) Compute $\chi_{17}(x)$ for all $x \in \mathbb{Z}_{17}$. Show that, for any given prime number $p$ and $x_1, x_2 \in \mathbb{Z}_p$, it holds that $\chi_p(x_1) \cdot \chi_p(x_2) = \chi_p(x_1 \cdot x_2)$.

   (b) Observe the values that $\chi_p$ takes. Prove that $\chi_p(x)$ is always either $0, 1$ or $-1$ and give a characterization of $\chi_p(x)$.

   For example: $\chi_p(x) = 0$ if and only if $x = 0$. Similarly, find necessary and sufficient condition for $\chi_p(x) = 1$ as well as necessary and sufficient condition for $\chi_p(x) = -1$.

   (c) Using the answer to question (b), prove the following: Given $x, y$ in $\mathbb{Z}_p$ that are not squares (i.e. $x \neq z^2$ for any $z \in \mathbb{Z}_p$ and the same for $y$), then $x \cdot y$ is always square.

   Connect to the Radboud network or ssh into the lilo server as in exercise 1, and run in your terminal:

```
nc appliedcrypto.cs.ru.nl 4145
```

   Alice is running the script given in Appendix B. You will need to closely analyze this script to understand what cryptosystem she is using. You also need to interact with the server, so the file `interaction_example.py` can help.

   (d) Win the game, make a write-up of your solution, and explain your steps! (Hint: what is $\chi_p(g^a)$?)

   (e) If you won the game, you essentially made a distinguisher $\mathcal{D}$ for a certain problem. What decisional problem did you solve? Compute the advantage of your distinguisher $\mathcal{D}$ and show why this decisional problem is easy in this case.

   (f) State the computational variant of this decisional problem and argue if it is still hard or not.

   (g) **(Bonus 3 points.)** What modification to the code would you propose to make the decisional problem hard again? Explain how this modifies the problem in which you are working.

   (h) **(Bonus 5 points.)** In the script you are given a triple, this triple can be one of two forms: $(A, B, C)$ or $(A, B, Z)$. Assume you are given only the last element of the tuple: either $C$ or $Z$. In this case, is the decisional problem hard? If yes, show why by a reduction. If not, give a distinguisher $\mathcal{D}$, its advantage $\mathrm{Adv}(\mathcal{D})$, and compare against the advantage in (e).

   (i) **(Bonus 1 point.)** The $\chi_p$ function is a well-known function, with a well-known name. What is that name?

## A Script exercise 1

```
import random
import math
from secret import encoding, decryption_oracle, CORRECT_GUESS

modulus = int("""
155DCEBC 04BD2F5E BE9E85B3 BD3B4C00 DB8FFD13 2460EFB4
9CCC075B CE16D9DD E73F5579 0A3C2787 A4444D23 53286320
86827E70 72AC1BF7 02904EF0 25A4A6FD EEF0E482 C5F76D29
3774F3AE 9E6B5DA4 7B09FD3C D713AE6C A9AADA25 3CDB15FB
1EA14F94 6E1A5DE5 FE922A1D 3FB9B733 8F738A6D A0AEAA36
ADCAD991 DE84537C 2A399BC9 65BBEE58 689BE4C2 6DBDBF89
A2CC80D2 3A28B35D 8249B26A 870190CB E6B875DC 51FD4FE5
886AFA97 BDD98E2E FEC332B2 96E56F69 61C73BD8 BFE6CE64
DABFE02E 749004D8 D4157E5D 70FA0D1F B37F9891 0EAE1B59
3D15C924 31D8821D E683626C 3C125D84 B5E90B0D 87BA3977
2F407841 76B96E2F 42988B13 E210A0FB B
""".replace('\n', '').replace(' ', ''),
16)
public_key = 65533


def encrypt(plaintext, public_key, modulus):
       return pow(plaintext, public_key, modulus)

name = input("What is your name: ")
symm_key = secret_function(name)

encrypted_key = encrypt(symm_key, public_key, modulus)
print("The encryption of the symmetric key is:")
print(encrypted_key)

print("You can make ONE decryption query!")
decryption_query = int(input("What do you want to decrypt?:"))
print("Decrypted query: ", decryption_oracle(decryption_query))

print("Can you guess the value of the symmetric key? ")
symm_key_guess = int(input("You can guess only once: "))
if symm_key == symm_key_guess:
       print(CORRECT_GUESS)
else:
       print("Nope.")
```

## B Script exercise 3

```
from flag import FLAG, WITTYCOMMENT
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

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

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

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

    print(f"""Here is A=g^a: {A}, and B=g^b: {A}, {B}.
What is the following, g^ab or a random group element?
{C if challenge == 1 else Z}

Choose 1 if you think it’s g^ab and 0 if you think it’s random.""")


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

    if guess == challenge:
        print(f"""Correct!
It was indeed {"g^ab" if challenge == 1 else "a random element"}.
You can check for yourself with a: {a}, and b: {b}""")
        return 1
    else:
        print(f"""Wrong!
It was actually {"g^ab" if challenge == 1 else "a random element"}.
You can check for yourself with a: {a}, and b: {b}""")
        return -1



def main():
    balance = 100

    print(f"""Welcome to our Radboud Random Game:
We have picked a great group for Diffie-Hellman so that our
Decisional problem is definitely unbreakable!
Try it, we are sure you will not be able to get 120 points!
You will start with {balance} points. Good luck!
    """)

    while True:
        balance += play()

        if balance <= 0:
            print(WITTYCOMMENT)
            exit(0)

        if balance >= 120:
            print(FLAG)
            exit(0)

        print(f"Your current balance is {balance} points.\n")

if __name__ == '__main__':
    main()
```