Applied Cryptography

Public Key Cryptography, Assignment 4, Tuesday, October 15, 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, 10 November, 23.59

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

1. **(20 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 χ_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 `code-ddh.py`. You will need to closely analyse 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. Both python files are on BrightSpace, and are also given in Appendix A.

   (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 in Appendix A would you propose to make the decisional problem hard again? Explain how this modifies the problem on 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 χ_p function is a well-known function, with a well-known name. What is that name?

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

KeyGen:

(a) Choose a random prime p

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

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

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

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

Encrypt: To encrypt a message $M \in \mathbb{Z}_p$:

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

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

$$C_1 \leftarrow g^k \pmod{p}$$
$$C_2 \leftarrow y^k M \pmod{p}$$

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

(a) Show that C_2 is uniformly distributed over $\mathbb{Z}_p^*$. That is, for any fixed element z of $\mathbb{Z}_p^*$ it holds that:

$$\Pr[C_2 = z] = \frac{1}{p-1}\,.$$

(b) Suppose Malice has eavesdropped the values (C_1, C_2) sent to Alice in a previous communication. Suppose further that Malice can use Alice as a decryption oracle, such that Alice will decrypt any message sent to her that she has not seen before.

Show that by sending an appropriate ciphertext to Alice for decryption, Malice can learn the plaintext of (C_1, C_2). (Hint: See the active oracle attack on RSA in the lectures.)

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.

(c) Let r be the order of the subgroup generated by g. Show that, for any ciphertext (C_1, C_2), it holds that $C_2^r = M^r$, i.e. we can remove the random mask that hides the message in the ciphertext.

(d) Suppose that M is not in the subgroup generated by g. Use (c) to show that, in such a scenario, the scheme is vulnerable to a Meet-in-the-middle attack.

3. **(15 points)** During the guest lecture, the Paillier cryptosystem was introduced. A description is given below:

KeyGen:

(a) Choose random primes p, q such that

$$\texttt{gcd}(p \cdot q, (p-1) \cdot (q-1)) = 1$$

(b) Compute $n = p \cdot q$ and $\lambda = \texttt{lcm}(p-1, q-1)$, where `lcm` means least common multiple

(c) Select a random integer $g \xleftarrow{\$} \mathbb{Z}^*_{n^2}$

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

(e) Verify that n divides the order of g, otherwise go back to (a)

(f) Compute $\alpha = \frac{g^\lambda - 1}{n} \pmod{n^2}$, where the division is integer division

(g) Compute $\mu = \alpha^{-1} \pmod{n}$

(h) Output the public key $\mathsf{pk} = (n, g)$ and the private key $\mathsf{sk} = (\lambda, \mu)$

Encrypt: To encrypt a message $M \in \mathbb{Z}_n$:

(a) Choose a random $r \xleftarrow{\$} \mathbb{Z}_n$

(b) Output the ciphertext $C = g^M \cdot r^n \pmod{n}^2$

Decrypt: To encrypt a ciphertext in $\mathbb{Z}_{n^2}$:

(a) Compute $\gamma = \frac{c^\lambda - 1 \pmod{n^2}}{n}$

(b) Output the plaintext $M = \gamma \cdot \mu \pmod{n}$

Suppose we have generated a public key $\mathsf{pk} = (n, g)$ and private key $\mathsf{sk} = (\lambda, \mu)$ and assume that Paillier works, that is that given any $M \in \mathbb{Z}_n$ we always have:

$$\textbf{Decrypt}(\textbf{Encrypt}(M)) = M$$

(a) As was discussed in the guest lecture, Paillier is an additive homomorphic encryption scheme. This means it has the following property: Suppose we have M_1, $M_2 \in \mathbb{Z}_n$, and we take the encryptions $C_1 \leftarrow \textbf{Encrypt}(M_1)$, and $C_2 \leftarrow \textbf{Encrypt}(M_2)$. If we take their product $C = C_1 \cdot C_2 \pmod{n^2}$, it decrypts to $\textbf{Decrypt}(C) = M_1 + M_2$. Show that this encryption scheme does, indeed, have this property.

(b) Suppose that we encrypt using a fixed r, instead of a randomly chosen one. This would make **Encrypt** deterministic. In the lectures, we showed that a deterministic PKE can not be IND-CPA secure. Suppose that, in the IND-CPA experiment, we limit the adversary to submitting messages M^*_0, M^*_1 different from the adversary's queries M_i. Show that the Paillier cryptosystem with fixed r still does not have indistinguishability against such an attacker. (Hint: Use (a).)

(c) Prove that no homomorphic encryption scheme can be IND-CCA2 secure. (Hint: Recall that decription is deterministic.)

A Scripts exercise 1

code-ddh.py

```
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()
```

interaction_example.py

```
# this file shows you how to interact with a script running remotely
# especially if you want to interact multiple times, this is much easier
# than doing it by hand.

# this package is required for everything, you can install it with
# 'pip install pwntools'
from pwn import *

# we need to connect to the remote server (this requires us to be in
# the Radboud network) we do so by setting up a remote process
r = remote('appliedcrypto.cs.ru.nl', 4145)

# this allows us to read the bits we've received
output = r.recv()
print(output)

# we can also send bits just as easy. For assignment 4, exercise 1
# we want to send either '0' or '1'
r.sendline('0')
```

```
# what did we get back?
output = str(r.recv())
result = output.rfind("The challenge was")
print(output[2:result])
```