Applied Cryptography

Public Key Cryptography, Assignment 5, Monday, November 12, 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, November 24, 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. **(15 points)** Consider the following signature scheme: Let $\mathbb{Z}_p$ be the cyclic group of large prime order p, with a generator g. Let H be a cryptographic hash function which outputs an element of $\mathbb{Z}_p$. We consider p, g and H to be publicly known. With these parameters, the scheme is given as follows:

   > **KeyGen**:
   >
   > (a) Choose a random $x \xleftarrow{\$} \{1, \ldots, p-2\}$
   >
   > (b) Compute $y = g^x \mod p$
   >
   > (c) Output public key $\mathsf{pk} = y$ and private key $\mathsf{sk} = x$ and public parameters (p, g)
   >
   > **Sign**: A message m with digest $H(m) = d \in \mathbb{Z}_p$ is signed as follows:
   >
   > (a) Choose a random $k \xleftarrow{\$} \mathbb{Z}_{p-1}^*$
   >
   > (b) Compute $r = g^k \mod p$, and $s = (d - xr)k^{-1} \mod (p-1)$
   >
   > (c) If $s = 0$, go back to (a)
   >
   > (d) Output the signature (r, s)
   >
   > **Verify**: The signature (r, s) for a message m with $H(m) = d$ is verified as follows:
   >
   > (a) Check if $g^d = y^r r^s \mod p$. Accept if that equality holds, otherwise reject

   (a) Show that the signature scheme is correct, i.e., that **Verify** accepts any signature which **Sign** outputs.

   (b) This scheme is probabilistic. How many possible distinct signatures are there for a given message?

   (c) Show that, by setting $r = g^e \cdot y$ for an exponent $e \in \{1, \ldots, p-2\}$, we can choose d and s such that
   $$g^d = y^r r^s \mod p$$
   without knowing the secret key.

   (d) Argue why (c) does not describe an existential forgery attack on the scheme.

2. **(20 points)** Alice has made a script that she plans to use to transfer symmetric keys with her friend Bob. It is given on Brightspace as `code-animal.py`, as well as 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
```

Like exercise 1 from last week, you can reference the file `interaction_example.py`, found on Brightspace and in Appendix A, as a reference for interacting with the server. The file should now also include an example Hash function for exercise 2g.

(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 2b (but this time you are allowed 1000 queries) and write down whether it succeeded.

3. **(15 points)** In the guest lecture, the topic of side-channel attacks was raised. Side-channel attacks are attempts to break cryptographic schemes by using additional information based on physical measurements. Suppose an attacker has some form of access to a device which is making computations with a secret key. They could measure the duration computations, changes in power usage, or the electromagnetic radiation made by the device. This information might help an attacker in breaking the scheme.

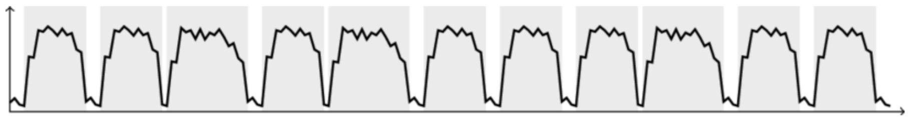
   Consider the square-and-multiply algorithm for exponentiation:

   Input: $X \in \mathbb{Z}_n$, $d \in \mathbb{N}$, with $d = d_1 d_2 \cdots d_k$ in binary.
   Output: $Y = X^d \mod n$.

   $Y \leftarrow 1$
   for $i \in \{1, \ldots, k\}$ **do**
       if $d_i = 0$ **then**
           $Y \leftarrow Y^2 \mod n$
       if $d_i = 1$ **then**
           $Y \leftarrow Y^2 \mod n$
           $Y \leftarrow Y * X \mod n$
   Return Y

(a) Argue the correctness of this algorithm. (Hint: Use induction)

(b) When tracking the power consumption using side-channel analysis, it is possible to distinguish between moments where a device is performing multiplication, and when it is squaring a value. Explain how this information can be used to derive d, and give the value of d corresponding to the power readings in the following image[1]:



(c) How could this be used to create a side-channel attack on RSA?

Such analysis of power usage requires that the adversary has physical access to a device which knows the secret key. A type of attack that is, typically, easier to achieve is a timing-attack, where the attacker measures the amount of time spent on certain computations.

(d) Suppose the attacker knows the total time spent on the square-and-multiply algorithm, as well as the time needed to perform a single multiplication or squaring step. What information about d could they uncover in this scenario?

[1] Image source: https://hal.science/tel-00733004/

A Scripts exercise 2

`code-animal.py`

```
import random
import math
from secret import secret_key, decryption_oracle

modulus = """4315620215092934399720402369023714857019
168194013257732037403680846324035088258304813340794
706455583957511287600281622775405620192951152959442
330419696661785062403781142003210472885193046469303
9613322475780195848729674846527655365879679078451495
18822775410166966816768833986107072953126359651410971
82400537288879952955223490616277325450839260389851876
9223189874607926212011496429700539958766417893026002503
8474302247096209888692518107068724001569615212092665409
7343760569607851150283737964166561880011311719583392974
3304720691814453532860881909220345497682441942893392060
1592396035248631882572207311154100883811864507"""
modulus = int(modulus)
public_key = 65533

#def animal_encoder(animal):
#     ascii_values = []
#     for char in animal:
#         ascii_values.append(str(ord(char)))
#
#     return int(''.join(ascii_values))


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


#query_string = input("What's your query: ")
#query = int(query_string)
#query = 805113737

query = random.randint(1, 100)
# print(f"{modulus = }")
# print(f"{public_key = }")
# print(f"{query = }")

print("Alice is generating a key to share with Bob...")
print("Alice is encrypting this key...")
print("Eve (you) has captured the encrypted key! Here it is:")
encrypted_query = encrypt(query, public_key, modulus)
print(encrypted_query)

decryption_query = int(input("You get ONE try! What do you want to decrypt: "))
if decryption_query == encrypted_query:
    print("Your ciphertext must be different than my ciphertext!")
else:
```

```
decrypted = decryption_oracle(decryption_query)
print("Decrypted: ", decrypted)

flag_guess = int(input("What is Alice's key? One query: "))
# print(f"{flag_guess = }")
if query == flag_guess:
    print("Win!")
else:
    print("Go fish.")
```