Lab 6: Diffie-Hellman Key Exchange

Deadline: 20 July 2024 11:59PM

Objectives

- Implement Square-Multiply for large integer exponentiation
- Implement Diffie-Hellman Key Exchange

Part I: Implementing Square and Multiply

Implement square and multiply algorithm to exponentiate large integers efficiently in a computer.

To do exponentiation of $y = a^x \bmod n$, we can use the following algorithm.

```
square_multiply(a, x, n)
{
  y = 1
  # n_b is the number of bits in x
  for( i = n_b-1 downto 0 )
  {
    # square
    y = y^2 mod n
    # multiply only if the bit of x at i is 1
    if (x_i == 1) y = a*y mod n
  }
  return y
}
```

```
square_multiply(a, x, n)
{
  y = 1
  # n_b is the number of bits in x
  for( i = n_b-1 downto 0 )
  {
    # square
    y = y^2 mod n
    # multiply only if the bit of x at i is 1
    if (x_i == 1) y = a*y mod n
  }
  return y
}
```

You are provided `primes_template.py` to use as a starting point. Save it as `primes.py`

Part II: Diffie-Hellman Key Exchange (DHKE)

DHKE can be used to exchange keys between two parties through insecure channel. For this exercise, you will simulate a exchange of keys using the DHKE protocol, following by sending an encrypted message.

Create a Python script that enables you to do this. Use the provided `dhke_template.py` as a starting point. Submit it as `dhke.py`.

Set-up

- Choose a large prime `p`. You can go to Wolfram Alpha and type in "prime closest to 2^80", or any other large number.
- Choose an integer $\alpha \in 2, 3, . . . , p-2$, which is a primitive element or generator in the group
- Save the values of `p` and `α`

Key Generation

- We need two sets of public and private keys to simulate the two parties in the key exchange
- Private keys (a, b)
  - Choose a random private key: $a = k_{pr,A} \in 2, \dots, p-2$
  - Choose **another** private key: $b = k_{pr,B} \in 2, \dots, p-2$
- Public keys (A, B)
  - Compute the first public key: $A = k_{pub,A} \equiv \alpha^a \bmod p$
  - Compute the second public key: $B = k_{pub,B} \equiv \alpha^b \bmod p$

Compute Shared Keys

Exchange your public keys and compute the shared keys: $k_{AB} \equiv B^a \bmod p$, or $k_{AB} = k \equiv A^b \bmod p$.

Message

Encrypt a message from you (A) using the shared key. You can use any method of encryption with the key. A suggestion is to use PRESENT with ECB mode from the previous lab, and encrypt a text message. Show that the other party (B) can decrypt the message.

Extend the end of the provided template file to print the plaintext, ciphertext to be sent by A, and then the decrypted plaintext by B.

Answer the following questions as Q1 and Q2 in a text file `ans.txt`:

1. How could we perform the exchange of keys in the real world? Do we need a secure channel? Why or why not?
2. What is an advantage and a disadvantage of DHKE?

Submission

eDimension Submission

Lab 6 submission:

Upload a zip file with the following:

file name format: lab6_name_studentid

Upload a **zip file** with the following:

- `primes.py`
- `dhke.py`
- `ans.txt (report)`

Deadline: 10 July 2024 11:59PM