Lab 6: Diffie-Hellman Key Exchange

Deadline: 28 July 2023 11:59PM

Objectives

- Implement Square-Multiply for large integer exponentiation
- Implement Diffie-Hellman Key Exchange
- Implement Shanks' Baby-step Giant-step method

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` You will need `square_multiply` for the next section. (it is imported by `dhke_template.py` and `babygiant_template.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 $\alpha \in 2, 3, \ldots, p-2$, 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: $a = k_{pr,A} \in 2, \ldots, p-2$
    - Choose **another** private key: $b = k_{pr,B} \in 2, \ldots, p-2$
- 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?

Part III: Baby-Step Giant-Step Method

This section introduces one method for solving discrete logarithm problem from which DHKE is based upon, the Baby-step giant-step.

An attacker can obtain the key if he can solve the discrete logarithm problem, basically find x in $\alpha^x = \beta$, where $1<=x<=p-1$. So α, β are the inputs, we want to find x. We can also write the problem as $x = \log_\alpha \beta$.

To do this, the problem is written in a two-digit representation:

$x = x_g m - x_b$

Where $0 \le x_g, x_b$. In this way, we can write the exponentation as:

$\beta = \alpha^x = \alpha^{x_g m - x_b}$

Rearranging the terms, we get:

$\beta\alpha^{x_b} = \alpha^{x_g m}$

Task 1

Create a Python script to do the following:

- Calculate `m = ceiling(sqrt(|G|))`, i.e. size of the square root of the order of the finite cyclic group G, where group order is `p-1`.

- Baby-step phase: Compute and store into a file the values of $\alpha^{x_b}\beta$, where $0 \le x_b < m$
- Giant-step phase: Compute and store into a file the values of $\alpha^{m \times x_g}$, where $0 \le x_g < m$
- Check the two list and see if there is a match such that: $\alpha^{x_b}\beta = \alpha^{m \times x_g}$
- If a match is found, calculate $x = x_g m - x_b$

Use the provided `babygiant_template.py` as a starting point. Submit it as `babygiant.py`.
Task 2

1. Try to attack DHKE with Baby-Step Giant-Steps method. Start with a key size of 16 bits. (a key with 16 bits has a small maximum value of 2^16, or 65536) Increase the number of bits (of the key) to break slowly, to increase the difficulty of the attack. Note where the attack fails or is too hard to execute.
2. To avoid attack using Baby-Step Giant-Steps method, how many bits would you set the key be in DHKE protocol? How did you decide on this number? Answer as Q3 in `ans.txt`

Attack Procedure

As the attacker, we start with private keys (a, b) which are unknown, and the public keys (A, B) which are known. Our goal is to guess the shared keys correctly without knowledge of the private keys.

1. We take `p` and `α` (from the setup step) and a public key as inputs to the Baby-Step Giant-Steps algorithm. The inputs should be `(α, β, p)`, where `β` is the public key of interest.
2. Perform the Baby-Step Giant-Steps algorithm, allowing us to derive a guess of `x`.
3. To get the guess of shared key (of the public key and the unknown private key), we perform a square_multiply of the public key and our guess of `x`.
4. To check if this is correct, we compute the real shared key from the public and private key. (Obviously, the real attacker cannot do this, but you can to check the correctness in this exercise).

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`
- `babygiant.py`
- `ans.txt (report)`

Deadline: 28 July 2023 11:59PM