Task 6: Diffie-Hellman Key Exchange Walkthrough

Since the task 6 was so confusing, I decided to add some documentation to walk through the three C++ programs demonstrating:

1. **Basic Diffie-Hellman (DH) key exchange:** The fundamental implementation.
2. **Man-in-the-middle (MITM) attack:** Illustrates a vulnerability in basic DH.
3. **Secured version with RSA signatures:** Shows how to secure DH against MITM attacks.

Setup

Prerequisites

- **C++ compiler:** g++ (or a compatible compiler)
- **OpenSSL library**
- **Operating System:** Windows or Ubuntu

Prepare Files

- Create `data/publickey.txt` for dh_secure: This file should contain RSA public keys for Alice and Bob. Have a text file named `publickey.txt` inside the `/data` directory, and add the following content:

```
Plaintext
3       # e1 (Alice's public exponent)
323     # n1 (Alice's modulus)
65537 # e2 (Bob's public exponent)
589     # n2 (Bob's modulus)
```

Build

Run via `make all`. The Executables `dh_basic`, `dh_mitm`, and `dh_secure` should show up in the `/bin` directory.

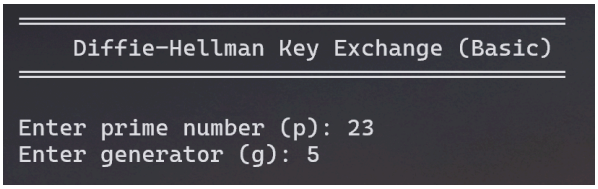
Program 1: Basic Diffie-Hellman (*dh_basic*)

Run

- **Windows:** bin\dh_basic.exe
- **Ubuntu:** ./bin/dh_basic

Walkthrough

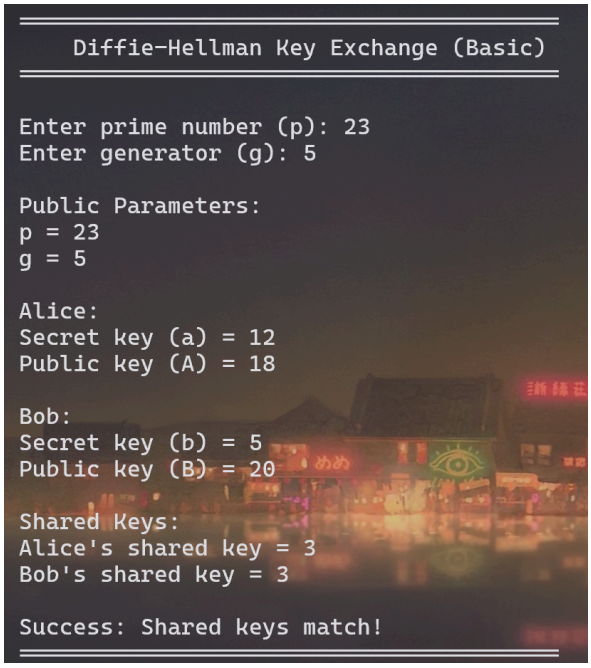
When you run the program, it will start with the following output:



What's Happening

- You are providing public parameters: a prime number **p** and a generator **g**.
- The generator **g** is chosen such that $g^x \bmod p$ produces a wide range of values before repeating.
- The program then generates random secret keys for Alice (**a**) and Bob (**b**), both of which are less than **p**.

Output (Example)



What's Happening

1. **Alice's Calculation:**
   - Alice computes her public key **$A = g^a \bmod p = 5^6 \bmod 23 = 8$.**
2. **Bob's Calculation:**
   - Bob computes his public key **$B = g^b \bmod p = 5^{15} \bmod 23 = 19$.**
3. **Key Exchange:**
   - Alice sends **A** to Bob, and Bob sends **B** to Alice.
4. **Shared Key Calculation (Alice):**
   - Alice receives **B** and computes the shared key: **$B^a \bmod p = 19^6 \bmod 23 = 2$.**
5. **Shared Key Calculation (Bob):**
   - Bob receives **A** and computes the shared key: **$A^b \bmod p = 8^{15} \bmod 23 = 2$.**
6. **Result:**
   - Both Alice and Bob arrive at the same shared key (**2**), because mathematically **$g^{(a*b)} \bmod p = g^{(b*a)} \bmod p$**. This shared secret can now be used for secure communication.

Step 2: Man-in-the-Middle Attack (*dh_mitm)*

Run

- **Windows:** bin\dh_mitm.exe
- **Ubuntu:** ./bin/dh_mitm

Walkthrough

The program starts with:

```
═══════════════════════════════════
  Diffie-Hellman with MITM Attack
═══════════════════════════════════

Enter prime number (p): 23
Enter generator (g): 5
```

What's Happening

- This program uses the same public parameters **p** and **g** as the basic DH example.
- However, it simulates a MITM attack where Mallory intercepts the key exchange between Alice and Bob.

Output (Example)

```
═══════════════════════════════════
  Diffie-Hellman with MITM Attack
═══════════════════════════════════

Enter prime number (p): 23
Enter generator (g): 5

Alice sends A = 15 (intercepted by Mallory)
Bob sends B = 7 (intercepted by Mallory)
Mallory sends M1 = 11 to Alice
Mallory sends M2 = 18 to Bob

Shared Keys:
Alice-Mallory key = 14
Mallory-Alice key = 14
Bob-Mallory key = 16
Mallory-Bob key = 16

MITM Success: Mallory has separate keys with Alice and Bob!
═══════════════════════════════════
```

Explanation

1. **Interception:**
    - Alice sends her public key **A = 8**. Mallory intercepts **A**.
    - Bob sends his public key **B = 19**. Mallory intercepts **B**.
2. **Mallory's Attack:**
    - Mallory generates her own secret keys **m1** and **m2**.
    - Mallory calculates **M1 = g^{m}_{1} mod p** (e.g., **5^{7} mod 23 = 10**) and sends **M_1** to Alice, pretending to be Bob.
    - Mallory calculates **M2 = g^{m}_{2} mod p** (e.g., **5^{3} mod 23 = 7**) and sends **M_2** to Bob, pretending to be Alice.
3. **Key Calculation (Alice & Mallory):**
    - Alice receives **M_1** and computes **M_1^{a} mod p = 10^{6} mod 23 = 12**. This is the shared key between Alice and Mallory.
    - Mallory computes **A^{m}_{1} mod p = 8^{7} mod 23 = 12**. This is also the shared key between Mallory and Alice.
4. **Key Calculation (Bob & Mallory):**
    - Bob receives **M_2** and computes **M^{2}_{b} mod p = 7^{15} mod 23 = 9**. This is the shared key between Bob and Mallory.
    - Mallory computes **B^{m}_{2} mod p = 19^{3} mod 23 = 9**. This is also the shared key between Mallory and Bob.
5. **MITM Success:**
    - Mallory has successfully established separate shared keys with both Alice and Bob.
    - Mallory can now decrypt messages from Alice meant for Bob (and vice versa), and re-encrypt them before forwarding, effectively reading and potentially modifying the communication without Alice or Bob knowing.

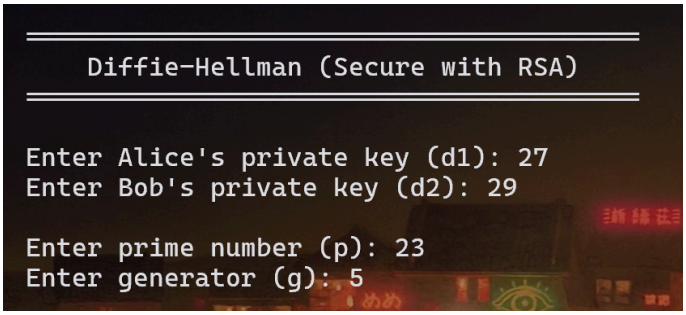
Step 3: Secure Diffie-Hellman (*dh_secure*)

Run

- **Windows:** bin\dh_secure.exe
- **Ubuntu:** ./bin/dh_secure

Walkthrough

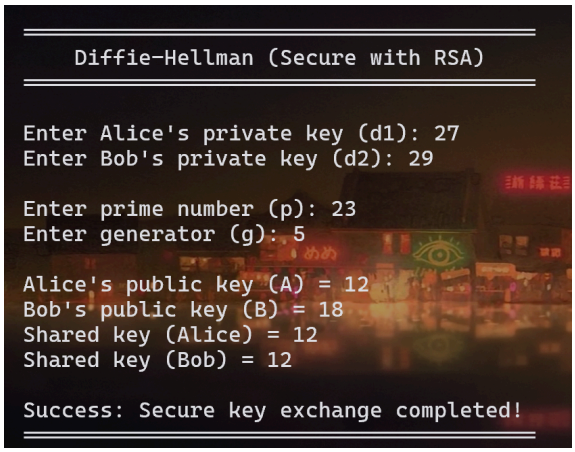
The program starts with:



What's Happening

- This program implements a secure version of Diffie-Hellman by adding RSA signatures to authenticate the public keys exchanged.
- It reads RSA public keys for Alice and Bob from the data/publickey.txt file prepared in the setup.
- You are prompted to enter the RSA private keys $\mathbf{d}_1$ (for Alice) and $\mathbf{d}_2$ (for Bob).

Output (Example)



Explanation

1. **Key Exchange and Signing:**
   - Alice computes her DH public key $\mathbf{A = 5^6 \bmod 23 = 8}$.
   - Alice signs **A** using her RSA private key $\mathbf{d_1 = 107}$ and modulus $\mathbf{n_1 = 323}$ (from `publickey.txt`). The signature is calculated as $\mathbf{signature_A = A^d{}_1 \bmod n_1}$.
   - Bob computes his DH public key $\mathbf{B = 5^{15} \bmod 23 = 19}$.
   - Bob signs **B** using his RSA private key $\mathbf{d_2 = 233}$ and modulus $\mathbf{n_2 = 589}$ (from **`publickey.txt`**). The sig is calculated as $\mathbf{signature_B = B^d{}_2 \bmod n_2}$.
2. **Verification:**
   - Alice sends **A** and $\mathbf{signature_A}$ to Bob.
   - Bob sends **B** and $\mathbf{signature_A}$ to Alice.
   - **Alice verifies Bob's signature:** Alice uses Bob's RSA public key components $\mathbf{e_2 = 65537}$ and $\mathbf{n_2 = 589}$ (from `publickey.txt`) to verify Bob's signature. She checks if $\mathbf{signature_B^{e2} \bmod n_2}$ is equal to **B**. If it is, Bob's public key is authenticated.
   - **Bob verifies Alice's signature:** Bob uses Alice's RSA public key components $\mathbf{e_1 = 3}$ and $\mathbf{n_1 = 323}$ (from `publickey.txt`) to verify Alice's signature. He checks if $\mathbf{signature_A^{e1} \bmod n_1}$ is equal to **A**. If it is, Alice's public key is authenticated.
3. **Secure Shared Key:**
   - If both signatures are verified successfully, Alice and Bob proceed to calculate the shared key as in the basic DH example. In this example, the shared key is **2**.
4. **MITM Prevention:**
   - If Mallory tries to perform a MITM attack and substitutes public keys with $\mathbf{M_1}$ and $\mathbf{M_2}$, the RSA signature verification will fail.
   - For example, if Mallory sends $\mathbf{M_1}$ to Alice pretending to be Bob, Mallory cannot create a valid signature for $\mathbf{M_1}$ that Alice can verify using Bob's RSA public key ($\mathbf{e_2}$, $\mathbf{n_2}$) because Mallory does not possess Bob's RSA private key.
   - The signature verification failure alerts Alice and Bob to the attack, preventing the establishment of a shared key with Mallory and thus securing the key exchange.

- end -