Computer Network

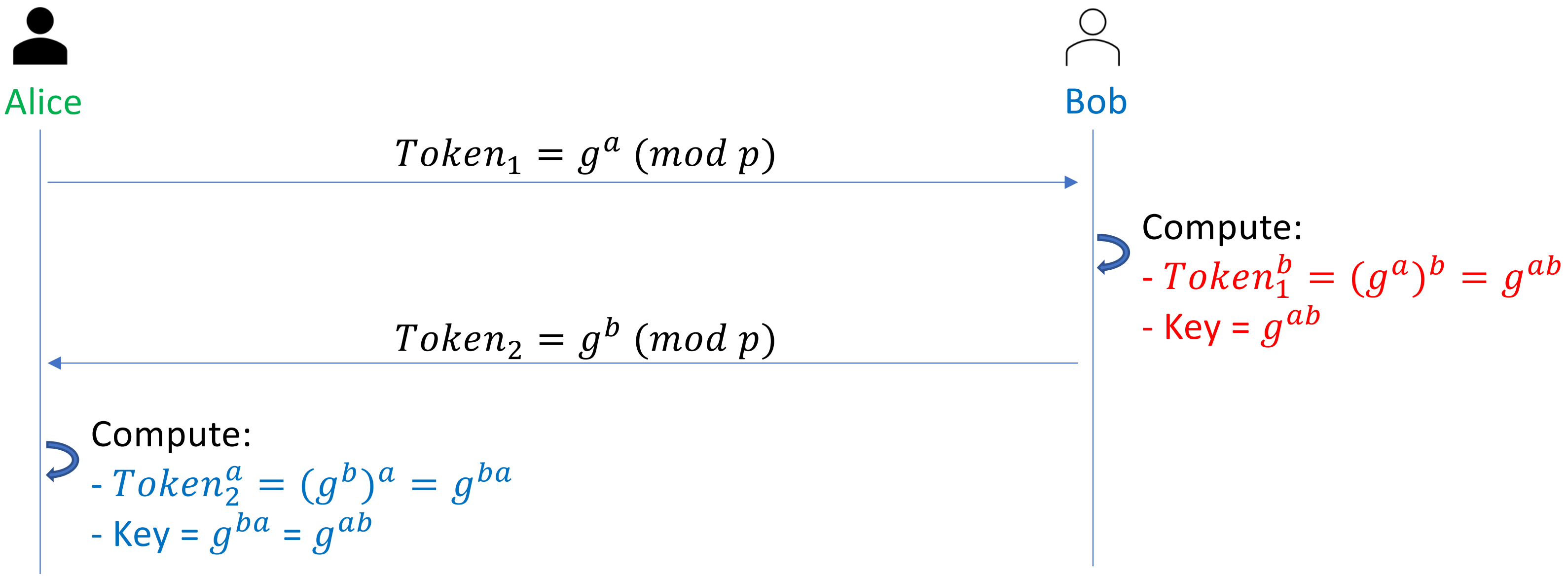
Luong Duc Anh
202131033

I. Introduction

- This project simulates the RSA key exchange algorithm.
- The key size is 64 bits.

I. Introduction (2)

- g, p are two 64-bits prime integers and they are public.
- Alice chooses a secret number (seed) a, and computes $Token_1 = g^a \ (mod\ p)$.
- Bob chooses a secret number (seed) b, and computes $Token_2 = g^b \ (mod\ p)$.
- Bob and Alice exchange $Token_1$ and $Token_2$.
- The secret key is $g^{ab} = Token_1^b = Token_2^a$
- Because a and b are secret, only Bob and Alice know g^{ab}. Bob does not know a and Alice does not know b.



II. Required tasks

- User enters a 64-bit seed and computes the Token = $g^{seed}\ (mod\ p)$.

→ There must be functions to compute the modulo power and multiplication.

- User sends Token to dest_ID.
- Token are sent via `arqLLI_sendData(arqPdu, pduSize, dest_ID)` in the form of an array of characters → There must be a function to a convert 64-bit integer to a string.
- User receive a token from dest_ID → Because the received token is a string, there must be another function to convert a string to a 64-bit integer.
- User calculate the key = received token ^ seed (mod p).

III. Parameters

These parameters are global parameters.

- *base* is *g = 9223372036854775880U*
- *order* is *p = 9223372036854775888U*
- *base* and *order* are public.

```
//my parameters
unsigned long long int seed;
uint8_t seed_8[200];
unsigned long long int base = 9223372036854775800U;
unsigned long long int order = 9223372036854775880U;
unsigned long long int token;
uint8_t token_8[200];
unsigned long long int token_in;
unsigned long long int key;
bool isKeyExchange = false;
```

III. Functions

1. mod_mul()

- This function takes as inputs three 64-bit integers *a*, *b*, and *mod*. It calculates *a***b* (mod *mod*)

```
unsigned long long int mod_mul(unsigned long long int a, unsigned long long int b, unsigned long long int mod){
    unsigned long long int res = 0; // Initialize result
    a %= mod;
    while (b) {
        // If b is odd, add a with result
        if (b & 1)
            res = (res + a) % mod;
        a = (2 * a) % mod;
        b >>= 1; // b = b / 2
    }
    return res;
}
```

III. Functions (2)

2. mod_power()

- This function takes as inputs 3 64-bit integers *x*, *y*, and *p*. It calculates *x^y* (mod *p*).

```
unsigned long long int mod_power(unsigned long long int x, unsigned long long int y, unsigned long long int p){
    // Initialize answer
    unsigned long long int res = 1;
    // Check till the number becomes zero
    while (y > 0) {
        // If y is odd, multiply x with result
        if (y % 2 == 1)
            res = mod_mul(res,x,p);
        // y = y/2
        y = y >> 1;
        // Change x to x^2
        x = mod_mul(x,x,p);
    }
    return res % p;
}
```

III. Functions (3)

3. print_menu()

- This function prints the banner.

```c
void print_menu(){
    pc.printf("=========================================================\n");
    pc.printf("=                        START!                         =\n");
    pc.printf("=            Computer Network Final Project.            =\n");
    pc.printf("=               Luong Duc Anh - 202131033               =\n");
    pc.printf("=========================================================\n");
}
```

III. Functions (4)

4. atollu()
- This function takes as input a string and convert it to a 64-bit integer.

```
unsigned long long int atollu(uint8_t *num){
    unsigned long long int buf;
    unsigned long long int total = 0;
    uint8_t l;
    uint8_t n;
    uint8_t j;
    const char* str = (const char*)num;

    l = strlen(str);
    for(int i = l; i >= 0; i--){
        j = l-i;
        buf = pow(10,i-1);
        n = num[j]-48;
        total += n*buf;
    }

    return total;
}
```

III. Functions (5)

5. llutoa()

- This function takes as inputs a 64-bit integer and a pointer. It converts the 64-bit integer to a string and stores it in the pointer.

```c
void llutoa(unsigned long long int num, uint8_t *des){
    uint8_t j;
    uint8_t l = 0;
    unsigned long long int buf;
    unsigned long long int p = 1;

    while(1){
        buf = num / p;
        if (buf == 0){
            break;
        }
        else{
            l++;
            p *= 10;
        }
    }
    for(int i = l-1; i >= 0; i--){
        j = l-1-i;
        buf = (unsigned long long int)pow(10,i);
        des[j] = num/buf + 48;
        num = num%buf;
    }
}
```

III. Functions (6)

6. get_seed()
- This function asks the user for a seed.

```
void get_seed(){
    pc.printf("Enter seed: ");
    pc.scanf("%llu", &seed);
    pc.printf("%llu\n", seed);
}
```

III. Functions (7)

7. generate_token()

- This function computes the token = $base^{seed}\ (mod\ order)$. Afterward, it converts the result into a string to be ready for sending it to dest_ID.

```c
void generate_token(){
    if(seed != 0){
        token = mod_power(base, seed, order);
        llutoa(token, token_8);
    }
    else{
        printf("------------------------------------------------------------\n");
        printf("Invalid seed (seed cannot be 0)!\n");
        printf("------------------------------------------------------------\n");
    };
}
```

III. Functions (8)

8. generate_key()

- Upon receiving the token (token_in) from dest_ID, this function calculates the secret key:

key = $token\_in$^seed (mod *order*)

```
void generate_key(){
    key = mod_power(token_in, seed, order);
    pc.printf("........................................................\n");
    pc.printf(". Key exchange success!\n");
    pc.printf(". g = %llu\n. p = %llu\n. Seed_%i = %llu\n. Token_%i = %llu\n. Received Token_%i = %llu\n. The secret key = %llu\n",
                base, order, endNode_ID, seed, endNode_ID, token, dest_ID, token_in, key);
    pc.printf("........................................................\n");
    isKeyExchange = true;

}
```

III. Functions (9)

9. get_id()
- This function asks the user for its ID and the dest_ID.

```
void get_id(){
    pc.printf(":: ID for this node : ");
    pc.scanf("%d", &endNode_ID);
    pc.printf("%i\n",endNode_ID);
    pc.printf(":: ID for the destination : ");
    pc.scanf("%d", &dest_ID);
    pc.printf("%i\n",dest_ID);
}
```

III. Functions (10)

10. init()

- This function initializes the low layer (set the sender's ID, receiver's ID, and wait for interrupts from the keyboard).

```
void init(){
    arqLLI_initLowLayer(endNode_ID);
    pc.attach(&arqMain_processInputWord, Serial::RxIrq);
}
```

III. Work flow

1. **Prepare for key exchange.**

- Print out banner, get all necessary data and compute the token.

```
//FSM operation implementation ------------------------------------------------
int main(void){
    uint8_t flag_needPrint=1;
    uint8_t prev_state = 0;

    //initialization
    pc.printf("------------------ ARQ protocol starts! --------------------------\n");
    arqEvent_clearAllEventFlag();
    arqEvent_setEventFlag(arqEvent_keyExchange);

    get_id();
    print_menu();
    get_seed();
    init();
    generate_token();
```

III. Work flow (2)

2**. Sending token**

- After computing the token, send it to dest_ID and changes the state to WAIT_ACK.
- The token will be retransmitted a few times to ensure that dest_ID receives it.

```
arqEvent_setEventFlag(arqEvent_dataToSend);
pduSize = arqMsg_encodeData(arqPdu, token_8, seqNum, strlen((const char*)token_8));
arqLLI_sendData(arqPdu, pduSize, dest_ID);
pc.printf("| [MAIN] sending token to %i (msg: %s, seq:%i)\n", dest_ID, token_8, (seqNum)%ARQMSSG_MAX_SEQNUM);
seqNum++;
main_state = WAIT_ACK;
arqEvent_clearEventFlag(arqEvent_dataToSend);
```

III. Work flow (3)

12. **WAIT_ACK**

- In this state, the user keeps listening for the ACK of the sending data.
- If ACK is received, it rotates back to MAINSTATE_IDLE.

```
case WAIT_ACK:
    arqTimer_startTimer();
    while(arqTimer_getTimerStatus()){       //while time is not out:
        if(arqEvent_checkEventFlag(arqEvent_ackRcvd)){
            // pc.printf("event received ack\n");
            uint8_t* ACKptr = arqLLI_getRcvdDataPtr();
            if(!arqMsg_checkIfAck(ACKptr)) break;
            pc.printf("| Received ACK: srcID = %i, seq = %i.\n-------------------------------------------------\n", arqLLI_getSrcId(), arqMsg_getSeq(ACKptr));
            arqTimer_stopTimer();
            wordLen = 0;
            arqEvent_clearEventFlag(arqEvent_dataToSend);
            arqEvent_clearEventFlag(arqEvent_ackRcvd);
            flag_needPrint = 1;
            main_state = MAINSTATE_TX;
            // pc.printf("state mainstate_tx\n");
        }
```

III. Work flow (4)

12. **WAIT_ACK**

- If no ACK received it retransmits the data.
- After a few times of retransmission, it goes back to MAINSTATE_IDLE as the data sending is failed.

```
if(arqEvent_checkEventFlag(arqEvent_arqTimeout)){
    // pc.printf("event timeout\n");
    if(rtm < ARQ_MAXRETRANSMISSION-6){

        arqEvent_setEventFlag(arqEvent_dataToSend);
        arqLLI_sendData(arqPdu, pduSize, dest_ID);
        arqEvent_clearEventFlag(arqEvent_dataToSend);

        pc.printf("| Retransmitting to %i\n", dest_ID);
        // arqTimer_stopTimer();
        main_state = WAIT_ACK;
        arqEvent_clearEventFlag(arqEvent_arqTimeout);
        rtm++;
        // pc.printf("rtm = %i\n", rtm);
        // pc.printf("timer status = %i", arqTimer_getTimerStatus());
    }
    else{
        arqEvent_clearEventFlag(arqEvent_dataToSend);
        pc.printf("Packet transmitting failed. No ACK received.\n");
        arqEvent_clearEventFlag(arqEvent_arqTimeout);
        wordLen = 0;
        main_state = MAINSTATE_IDLE;
        flag_needPrint = 1;
        rtm = 0;
    }
}
```

III. Work flow (5)

11. **Receiving Token**

- The token is identified by the sequence number. If the sequence number is 0, it is the token.
- These codes are written in both WAIT_ACK and MAINSTATE_IDLE states, so that the user can receive data while in both WAIT_ACK and MAINSTATE_IDLE.
- After receiving the token, it sends ACK to the token's sender.
- The received token is converted to a 64-bit integer for generating the secret key.

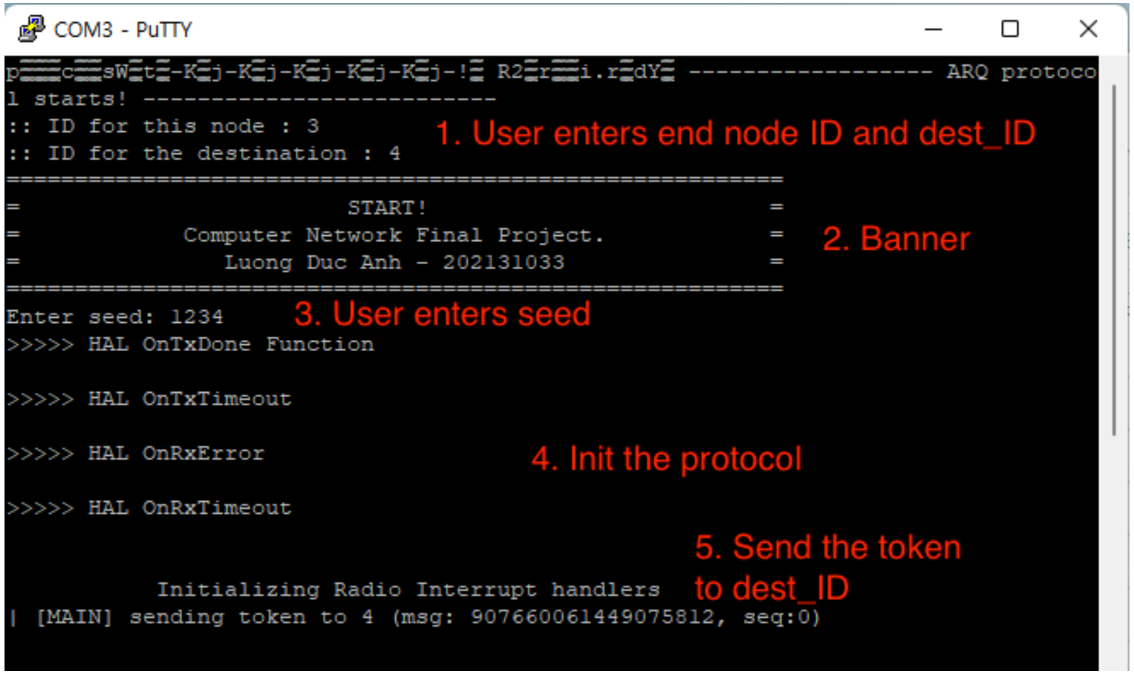
```
else if(arqEvent_checkEventFlag(arqEvent_dataRcvd)){
    uint8_t srcId = arqLLI_getSrcId();
    uint8_t* dataPtr = arqLLI_getRcvdDataPtr();
    uint8_t size = arqLLI_getSize();
    uint8_t rcd_seq = arqMsg_getSeq(dataPtr);
    if(rcd_seq == 0){
        pc.printf("\n-------------------------------------------------\n| Received Token_in from %i : %s (length:%i, seq:%i)\n",
                srcId, arqMsg_getWord(dataPtr), size, arqMsg_getSeq(dataPtr));
                token_in = atollu(arqMsg_getWord(dataPtr));
                generate_key();
    }
    else{
        pc.printf("\n-------------------------------------------------\n| RCVD from %i : %s (length:%i, seq:%i)\n",
    srcId, arqMsg_getWord(dataPtr), size, arqMsg_getSeq(dataPtr));
    }


    pduSizeAck = arqMsg_encodeAck(arqPduAck, rcd_seq);
    arqEvent_setEventFlag(arqEvent_dataToSend);
    arqLLI_sendData(arqPduAck, pduSizeAck, srcId);
    arqEvent_clearEventFlag(arqEvent_dataToSend);
    pc.printf("| ACK for seq %i is sent.\n-------------------------------------------------\n", rcd_seq);

    // main_state = MAINSTATE_IDLE;
    // flag_needPrint = 1;
    arqEvent_clearEventFlag(arqEvent_dataRcvd);
}
```

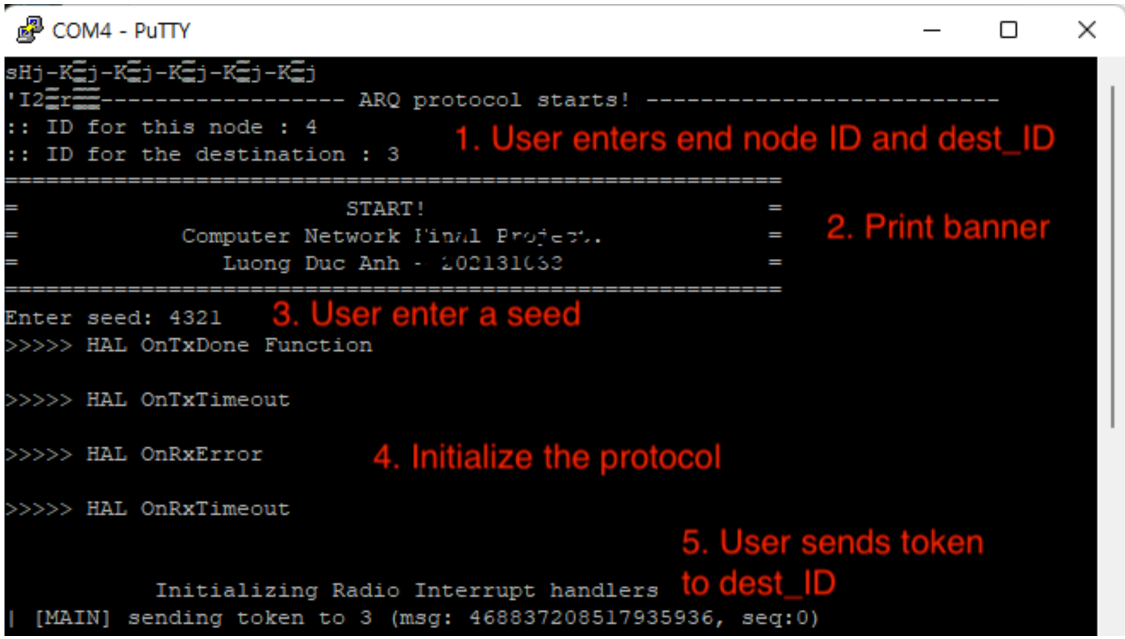
IV. Results

- The screen of COM3 (user 1).
- Seed is 1234
- Token is:
- $9223372036854775801^{1234}$ $(mod\ 9223372036854775588)$ = 907660061449075812



IV. Results (2)

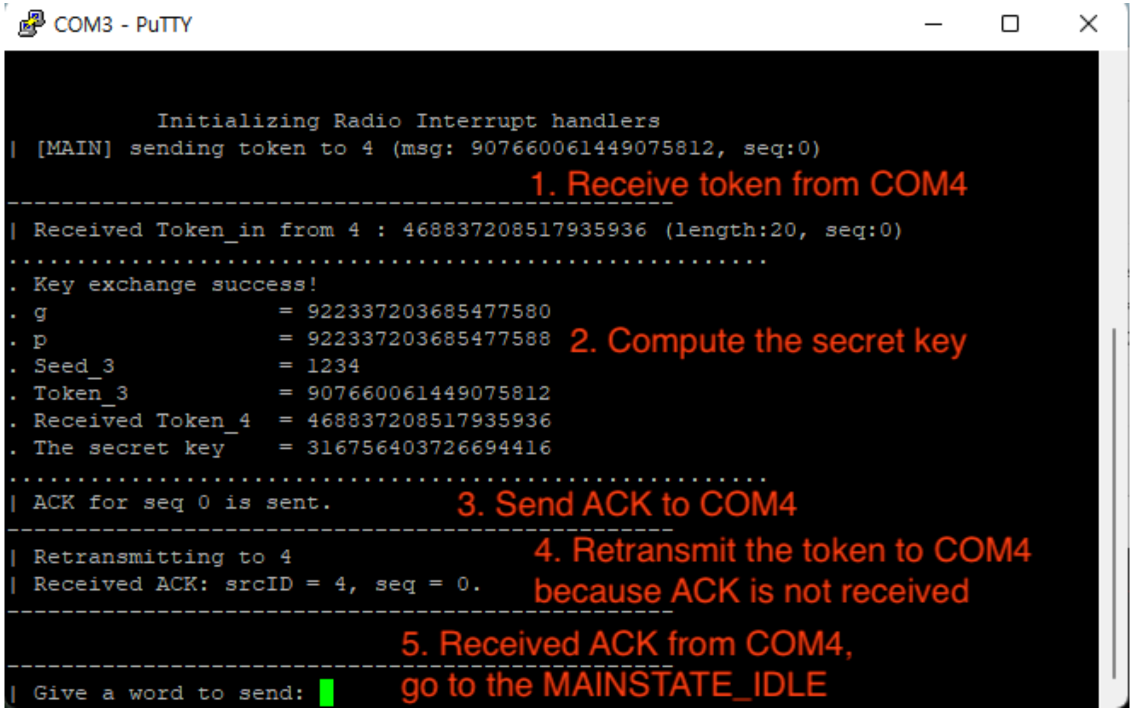
- The screen of COM4 (user 2).
- Seed is 4321
- Token is:
- $922337203685477580^{4321}$ $(mod\ 922337203685477588)$ = 468837208517935936



IV. Results (3)

- The screen of COM3 (user 1).
- The secret key = received token ^ 1234

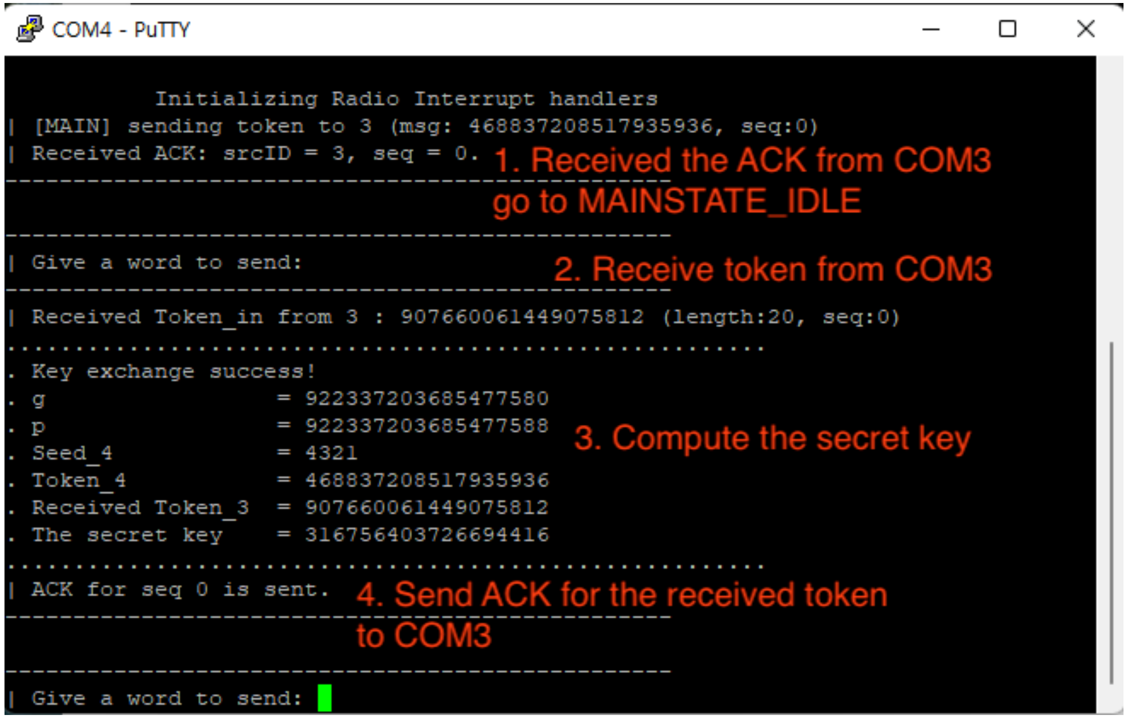
$= 468837208517935936^{1234}\ (mod\ 922337203685477588) =$ $\color{red}{316756403726694416}$



IV. Results (4)

- The screen of COM4 (user 2).
- The secret key = received token ^ 4321

$= 907660061449075812^{4321}\ (mod\ 922337203685477588) =$ 316756403726694416



IV. Results (4)

- The 64-bit secret key (3167564037266944 16) is successfully exchanged.
- COM3 and COM4 can use the secret key to encrypt and decrypt the message.
- Notably, other entities can only know tokens sent from COM3 and COM4. They cannot compute the secret key because the seeds of COM3 and COM4 (1234 and 4321) are secret.
- Therefore, the secret key is known by only COM3 and COM4.