RSA Encryption and Signature Lab Report

Emre Can Tüzer

Date: November 4, 2024

BGK-503

Kemal Bıçakcı

Task-1: Deriving the Private Key

First, we select three large prime numbers, p, q, and e, and let n = p ∗ q. The public key that will use is (e, n). In this project, 128-bit numbers are utilized, while 512-bit numbers are often employed. For p, q, and e, we employ the following hexadecimal values. The private key d must be determined.

$$n = p \times q$$

$$\phi(n) = (p-1) \times (q-1)$$

$$d = e^{-1} \times \bmod \phi(n)$$

```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *p = BN_new(); BIGNUM *q = BN_new(); BIGNUM *e = BN_new();
BIGNUM *n = BN_new(); BIGNUM *phi_n = BN_new();
BIGNUM *d = BN_new();
BIGNUM *p_m1 = BN_new(); BIGNUM *q_m1 = BN_new();
BIGNUM *one = BN_new();

// Initialize p, q, e and one
BN_hex2bn(&one, "1");
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3");


// n = p*q
BN_mul(n, p, q, ctx);
printBN("(p * q) = ", n);

// p-1, q-1
BN_sub(p_m1,p,one);
BN_sub(q_m1,q,one);

// phi(n) = (p-1)(q-1)
BN_mul(phi_n, p_m1, q_m1, ctx);
printBN("(p-1) * (q-1) = ", phi_n);

// d = e^-1 mod phi_n
BN_mod_inverse(d, e, phi_n, ctx);
printBN("Key = ", d);

bn2free_func(p, q, e, n, phi_n, d, p_m1, q_m1);
BN_CTX_free(ctx);

return 0;
```

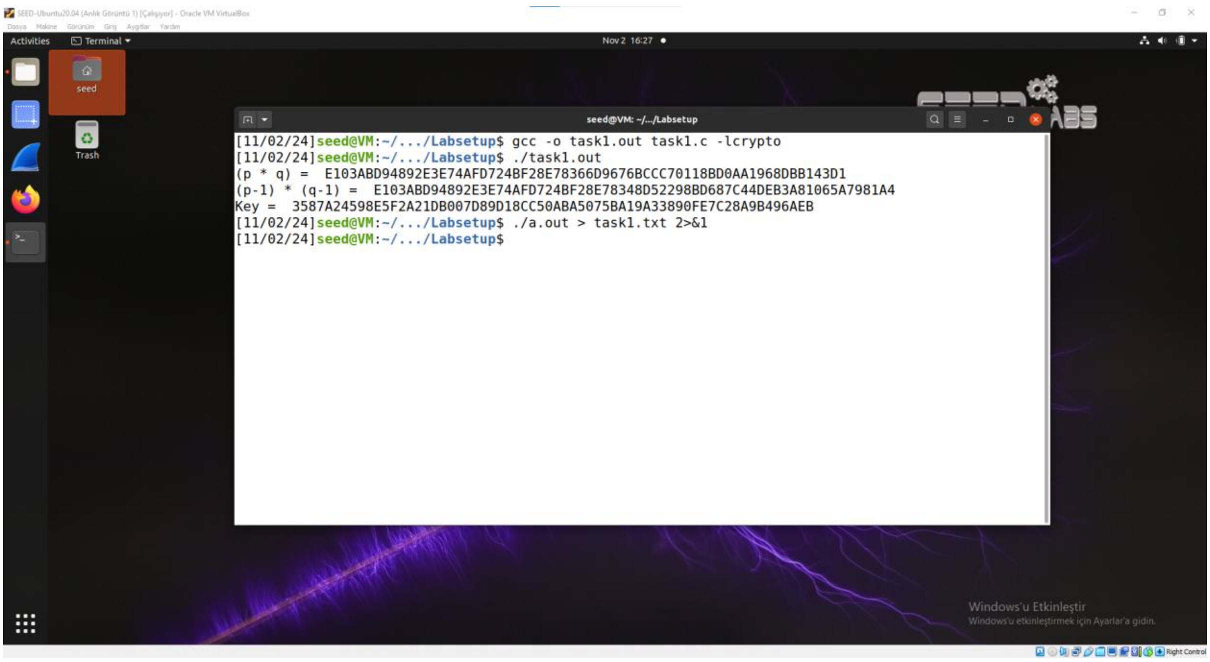
The product inverse is computed using the BN_mod_inverse function. For the function to successfully infer, the BN_CTX structure must be used.

Additionally, this function was added to prevent memory leaks for all tasks.

```
void bn2free_func(BIGNUM *bn1, BIGNUM *bn2, BIGNUM *bn3, BIGNUM *bn4, BIGNUM *bn5, BIGNUM *bn6, BIGNUM *bn7, BIGNUM *bn8)
{
BN_free(bn1);
BN_free(bn2);
BN_free(bn3);
BN_free(bn4);
BN_free(bn5);
BN_free(bn6);
BN_free(bn7);
BN_free(bn8);
}
```

Output:

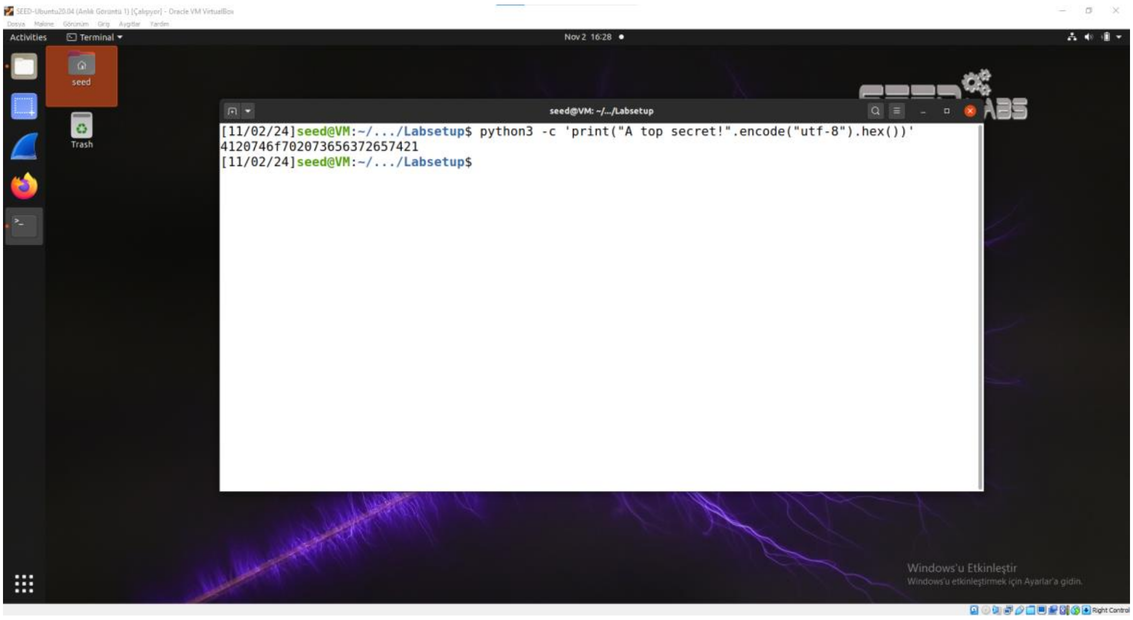
Running the file task1.c allows us to see the calculated values clearly, and the value of d is 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB.



Task-2: Encrypting a Message

Here, we encrypt the message "A top secret!" using the public-key (e, n). Since a computer program may represent a string using ASCII characters, we can convert these characters to hexadecimal and then back again.

$$C = M^e \times mod(n)$$



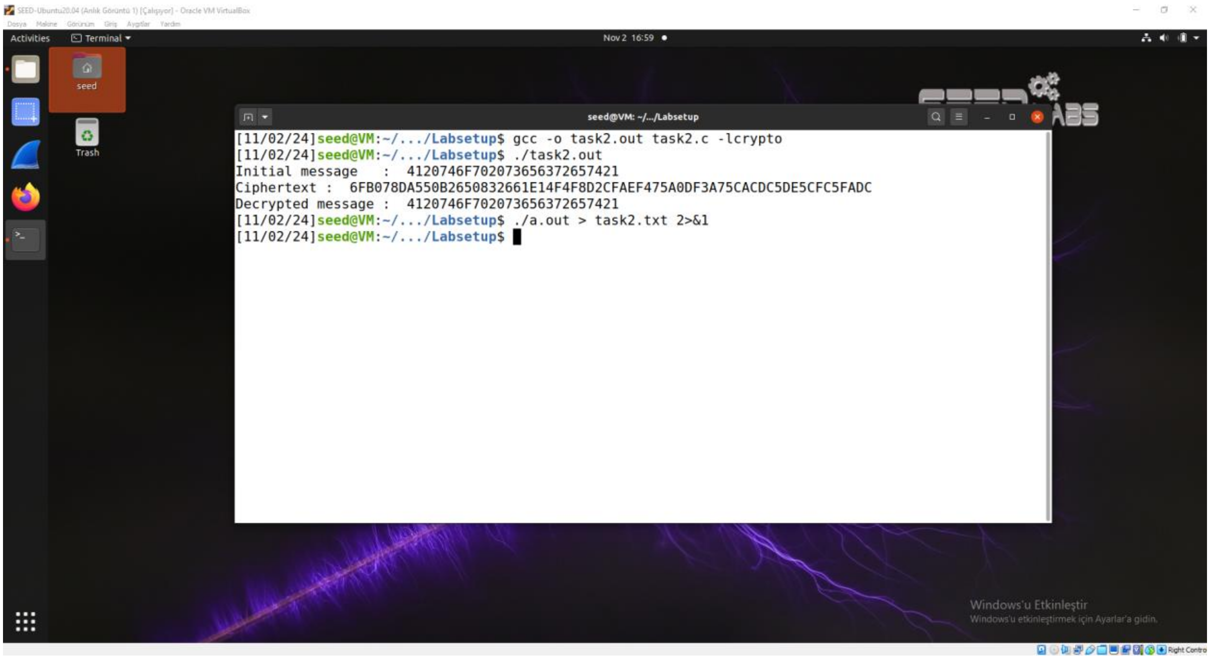
The message's ASCII codes are transformed into the following hex string: "A top secret!" Each of the three characters' ASCII values is represented in hexadecimal as 4120746f702073656372657421.

The encryption of the plain text M, or cipher text C, must now be calculated.

```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new(); BIGNUM *e = BN_new();
BIGNUM *Message = BN_new();
BIGNUM *Dec_message = BN_new();
BIGNUM *d = BN_new();
BIGNUM *Ciphertext = BN_new();

// Initialize n, M, e and d
BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&Message, "4120746f702073656372657421");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

printBN("Message   : ",Message);

// Encryption
BN_mod_exp(Ciphertext, Message, e, n, ctx);
printBN("Ciphertext : ", Ciphertext);

// Decryption
BN_mod_exp(Dec_message, Ciphertext, d, n, ctx);
printBN("Decrypted message : ", Dec_message);
bn2free_func(n, e, Message, Dec_message, d, Ciphertext);
BN_CTX_free(ctx);
```

The BN_hex2bn function converts hex string to BIGNUM. Large numbers are processed and stored using the BIGNUM data type. The BN_mod_exp function is used to accomplish encryption.

Output:



Task-3: Decrypting a Message

Here, Using a supplied cipher text C to decrypt a separate message using the same public/private keys as task 2. This is how we typically use RSA encryption to communicate over a network, first create a connection by automatically generating very large prime integers p and q, and then, use the same set of keys to encrypt and decrypt various messages. The following is the provided cipher text called C.

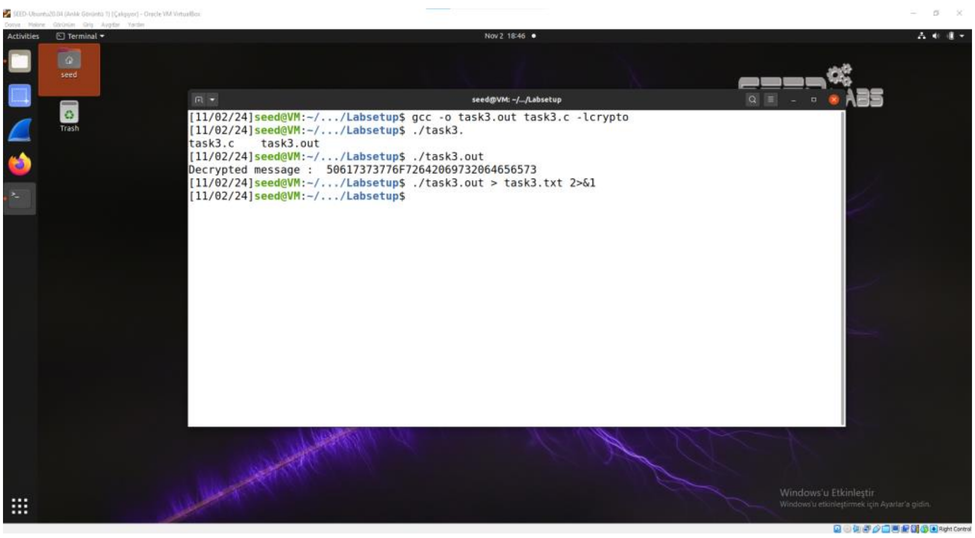
C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F

$$\boldsymbol{M} = \boldsymbol{C}^{\boldsymbol{d}} \times \boldsymbol{mod}(\boldsymbol{n})$$

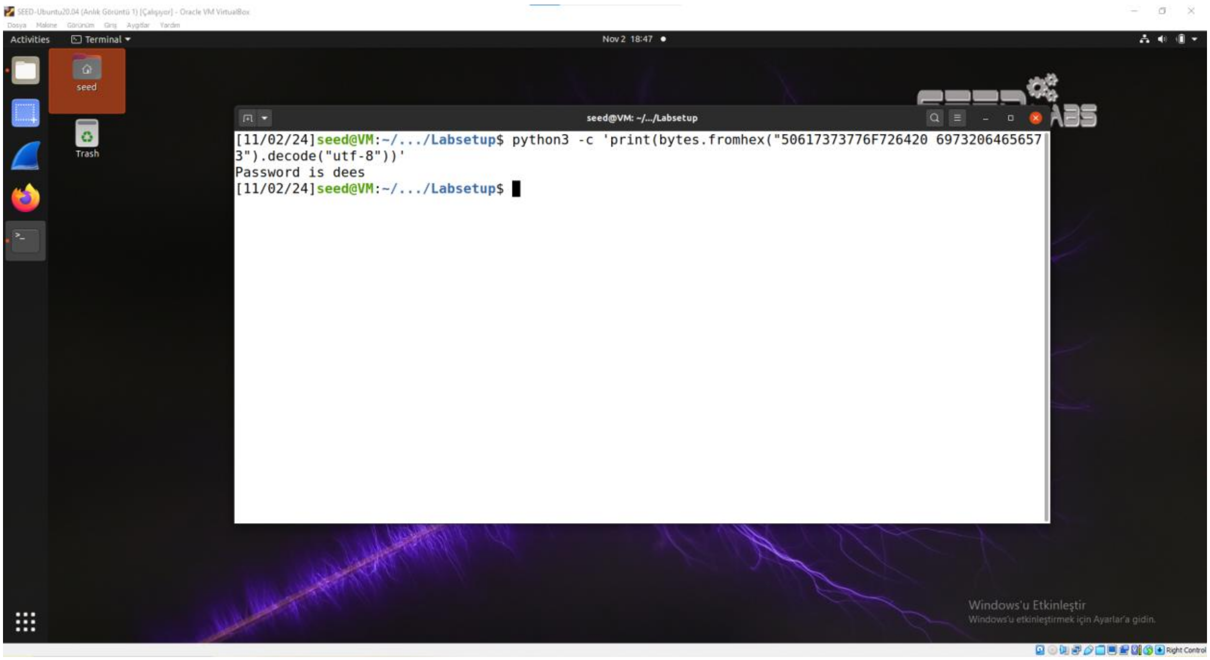
```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *Message = BN_new();
BIGNUM *d = BN_new();
BIGNUM *Ciphertext = BN_new();

// Initialize n, e, d and C
BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&Ciphertext, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

// Decryption: M = C^d mod n
BN_mod_exp(Message, Ciphertext, d, n, ctx);
printBN("Decrypted message : ", Message);
bn2free_func(n, e, Message, d, Ciphertext);
BN_CTX_free(ctx);
```

To conduct decryption, the BN_mod_exp function is utilized.

We obtain the decoded message in the output below.



The output of the ciphertext's decoding is a hex string. After converting a hexadecimal text to byte format and then to a UTF-8 string, we are able to extract our message.



Task-4: Signing a Message

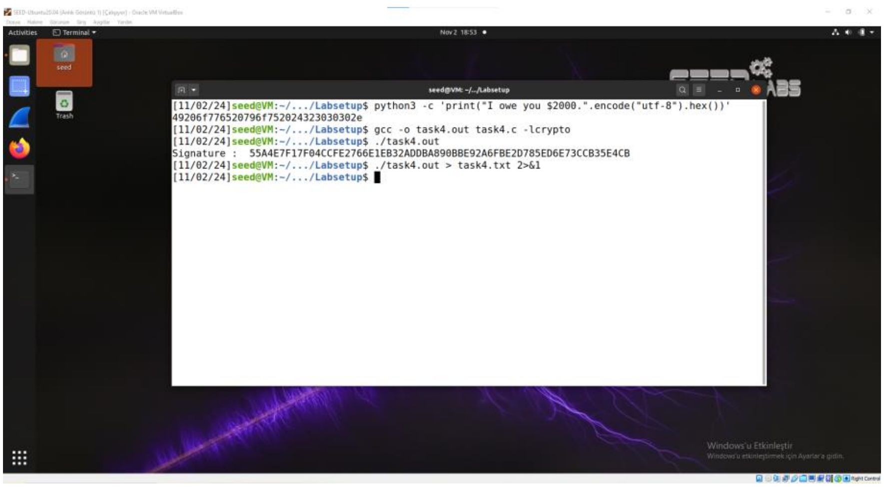
We must sign the following message: "I owe you $2000." RSA encryption typically involves using the RSA function to sign a message with the private key (d, n).

$$S = M^d \times mod(n)$$

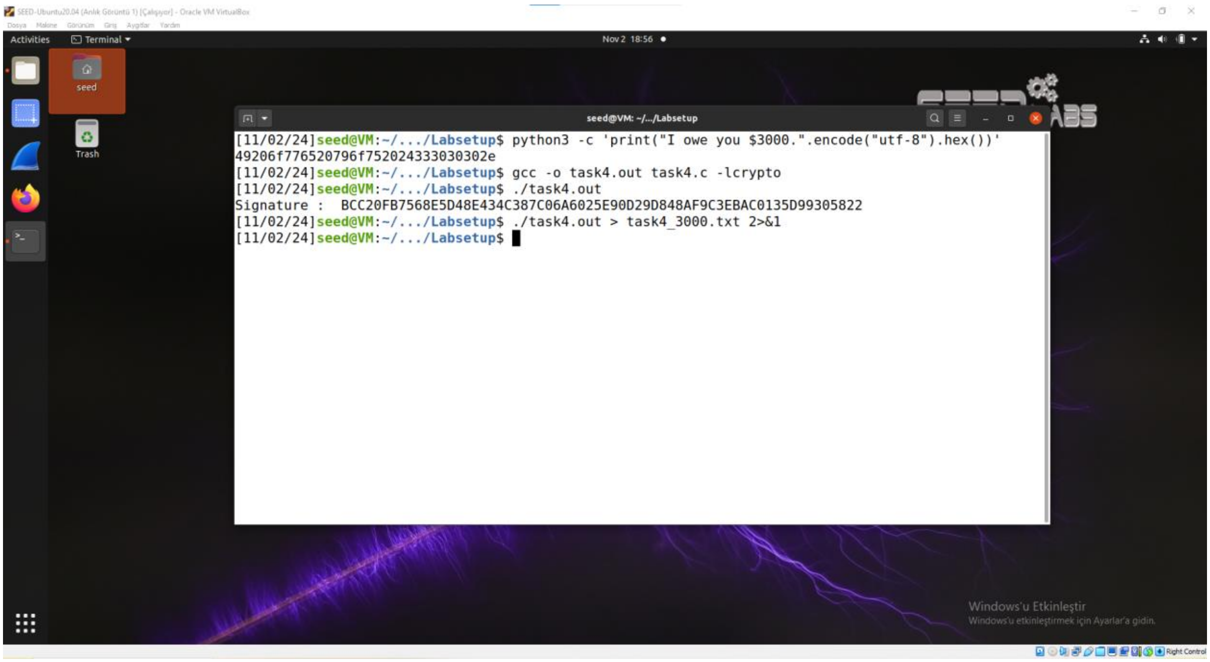
```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *Message = BN_new();
BIGNUM *d = BN_new();
BIGNUM *Signature = BN_new();

// Initialize n, e, d and C
BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
//BN_hex2bn(&Message, "49206f776520796f752024323030302e"); //I owe you $2000.
BN_hex2bn(&Message, "49206f776520796f752024333030302e"); //I owe you $3000.

// Digital signature: S = E(PR_a, H(M)) = M^d mod n
BN_mod_exp(Signature, Message, d, n, ctx);
printBN("Signature : ", Signature);
bn2free_func(n, e, Message, d, Signature);
BN_CTX_free(ctx);
return 0;
```

The BN_hex2bn function converts hex string to BIGNUM. A signature is created using the BN_mod_exp function.

Generated the signature S. For “ I owe you $2000” message.



We use the same Python command as before to convert the provided string to hexadecimal format.



Signature (2000) :
55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB

Signature (3000) :
BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822

It is easy to see that a single bit modification in the message results in a signature that is entirely different from the original message. This demonstrates that the signature is created entirely at random and does not omit any patterns while processing a sequence.

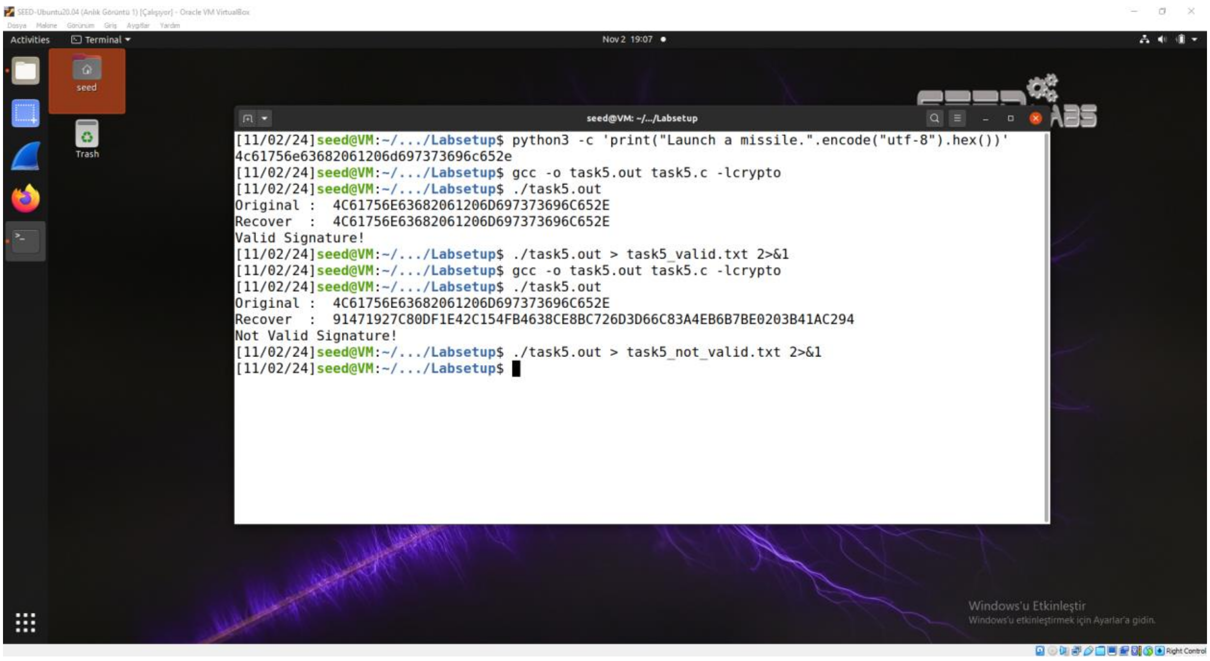
Task-5 Verifying a Signature

The BN_hex2bn function converts hex string to BIGNUM. A signature recover is created using the BN_mod_exp function.

$$M' = S^e \times mod(n)$$

```
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *e = BN_new();
BIGNUM *Message = BN_new();
BIGNUM *Recover = BN_new();
BIGNUM *Signature = BN_new();


BN_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&e, "010001");
//BN_hex2bn(&Signature, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
BN_hex2bn(&Signature, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F"); //modified S
BN_hex2bn(&Message, "4c61756e63682061206d697373696c652e"); //Launch a missile.


BN_mod_exp(Recover, Signature, e, n, ctx);
printBN("Original : ", Message);
printBN("Recover  : ", Recover);

if (BN_cmp(Message, Recover)==0){
    printf("Valid Signature!\n");
}
else {
    printf("Not Valid Signature!\n");
}
bn2free_func(n, e, Message, Recover, Signature);
BN_CTX_free(ctx);
```

We use the same Python command as before to convert the provided string to hexadecimal format.



When it comes to RSA digital signatures, altering even a single bit of the message or signature results in a whole different verification outcome. This is because of the cryptographic algorithm's mathematical characteristics, particularly its use of modular arithmetic and exponential operations.