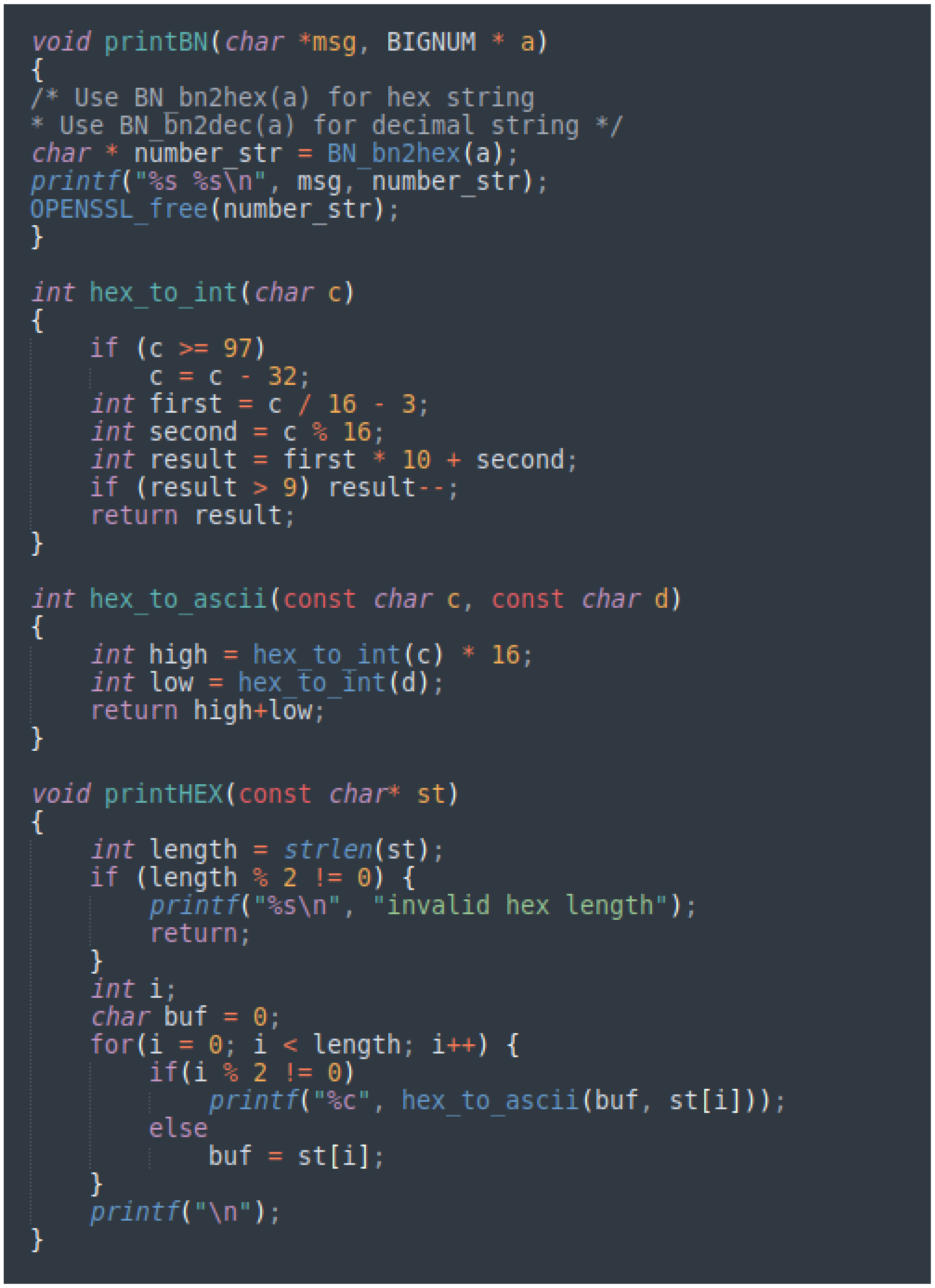
Val Robichaux
Homework 4
CSCE 465

Deliverables

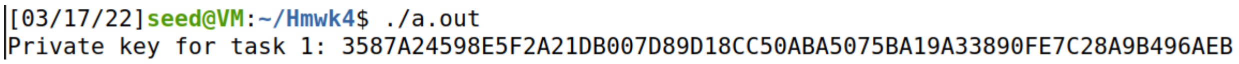
** NOTE **
I created four different function in order to produce results for printBN, hex to integer conversion, hex to ascii conversion and printing hex values as strings

I am going to leave all of them here at the top because they are important to understanding my explanations of the tasks when I call these functions.



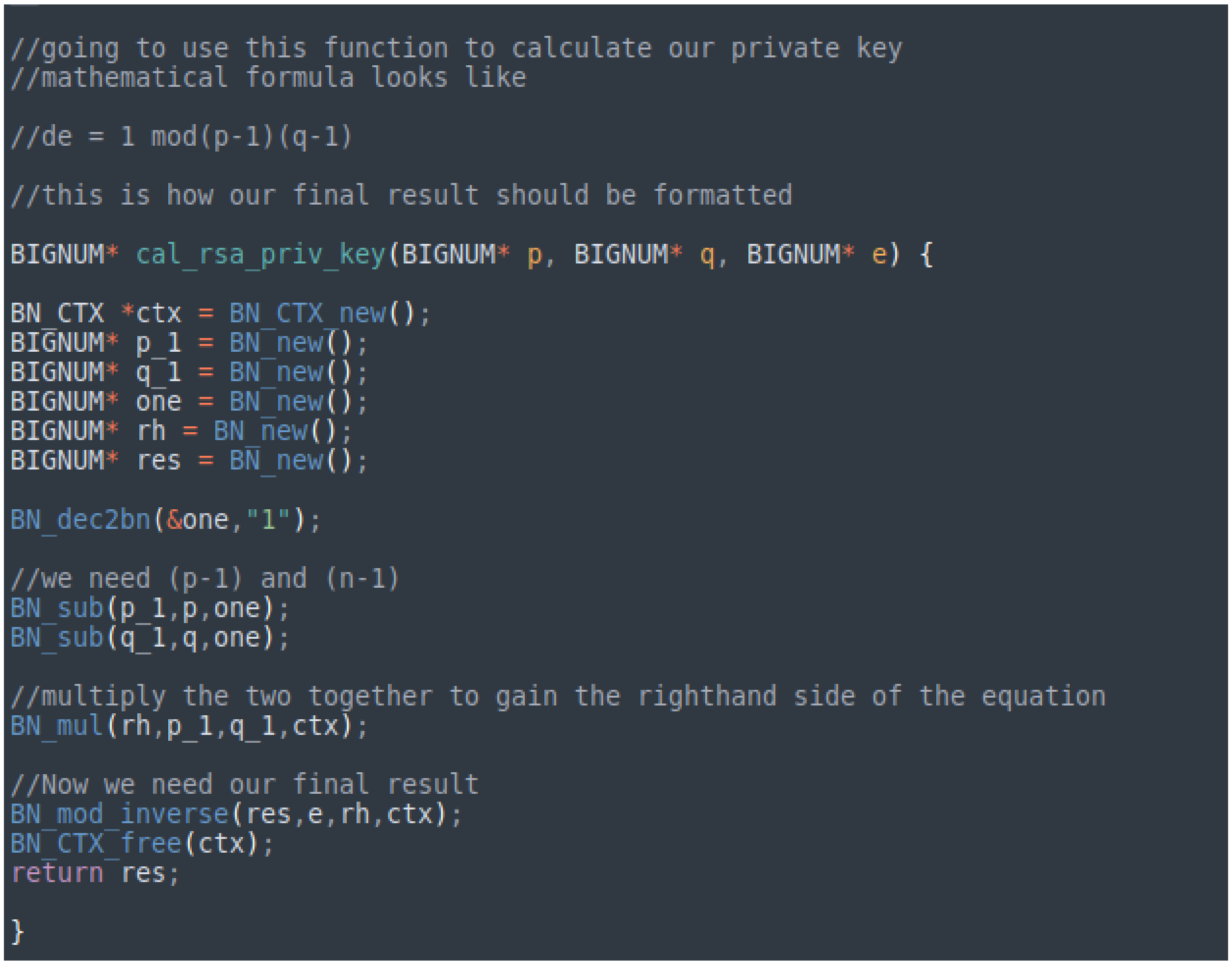
```c
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
* Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}

int hex_to_int(char c)
{
    if (c >= 97)
        c = c - 32;
    int first = c / 16 - 3;
    int second = c % 16;
    int result = first * 10 + second;
    if (result > 9) result--;
    return result;
}

int hex_to_ascii(const char c, const char d)
{
    int high = hex_to_int(c) * 16;
    int low = hex_to_int(d);
    return high+low;
}

void printHEX(const char* st)
{
    int length = strlen(st);
    if (length % 2 != 0) {
        printf("%s\n", "invalid hex length");
        return;
    }
    int i;
    char buf = 0;
    for(i = 0; i < length; i++) {
        if(i % 2 != 0)
            printf("%c", hex_to_ascii(buf, st[i]));
        else
            buf = st[i];
    }
    printf("\n");
}
```

1. Deriving the Private Key



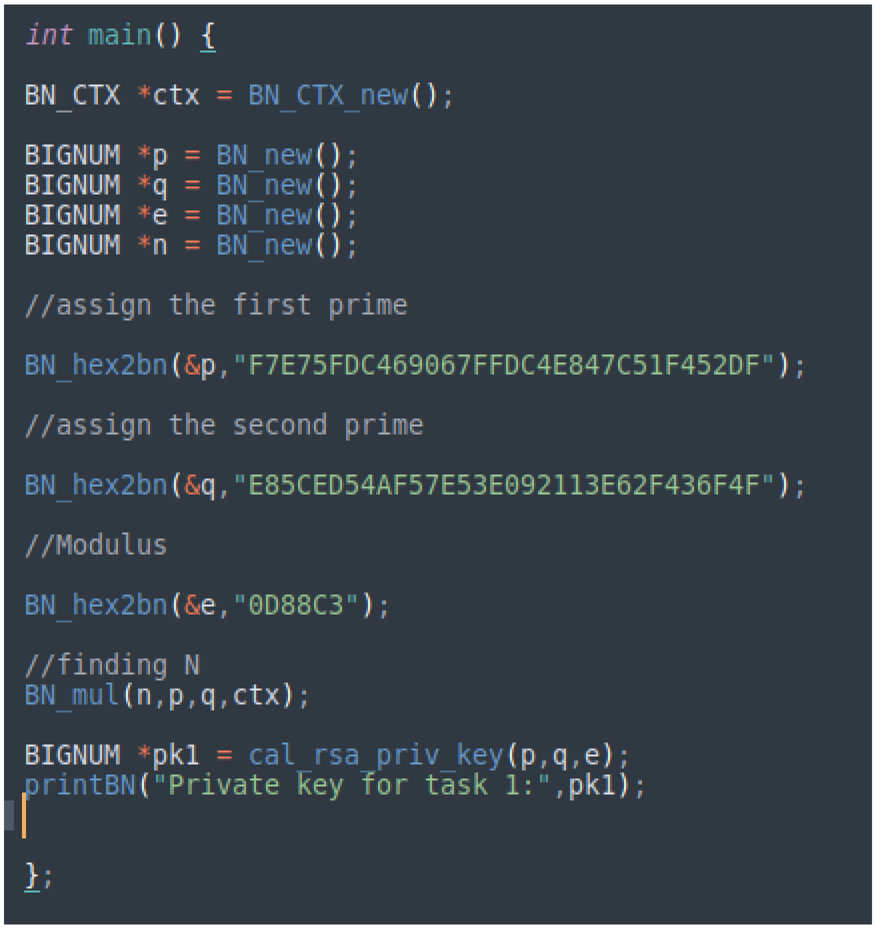
```
[03/17/22]seed@VM:~/Hmwk4$ ./a.out
Private key for task 1: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

- Calculated private key



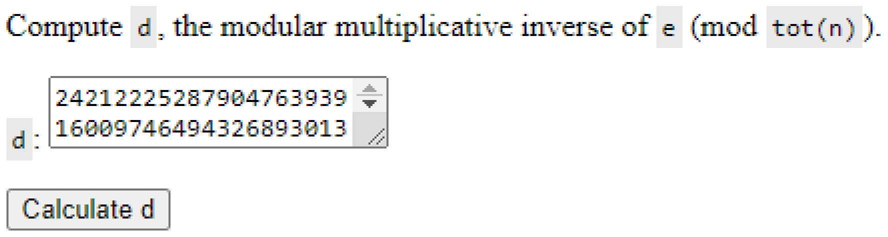
```
//going to use this function to calculate our private key
//mathematical formula looks like

//de = 1 mod(p-1)(q-1)

//this is how our final result should be formatted

BIGNUM* cal_rsa_priv_key(BIGNUM* p, BIGNUM* q, BIGNUM* e) {

BN_CTX *ctx = BN_CTX_new();
BIGNUM* p_1 = BN_new();
BIGNUM* q_1 = BN_new();
BIGNUM* one = BN_new();
BIGNUM* rh = BN_new();
BIGNUM* res = BN_new();

BN_dec2bn(&one,"1");

//we need (p-1) and (n-1)
BN_sub(p_1,p,one);
BN_sub(q_1,q,one);

//multiply the two together to gain the righthand side of the equation
BN_mul(rh,p_1,q_1,ctx);

//Now we need our final result
BN_mod_inverse(res,e,rh,ctx);
BN_CTX_free(ctx);
return res;

}
```

- The method I used where de - 1 mod(p-1)(q-1)



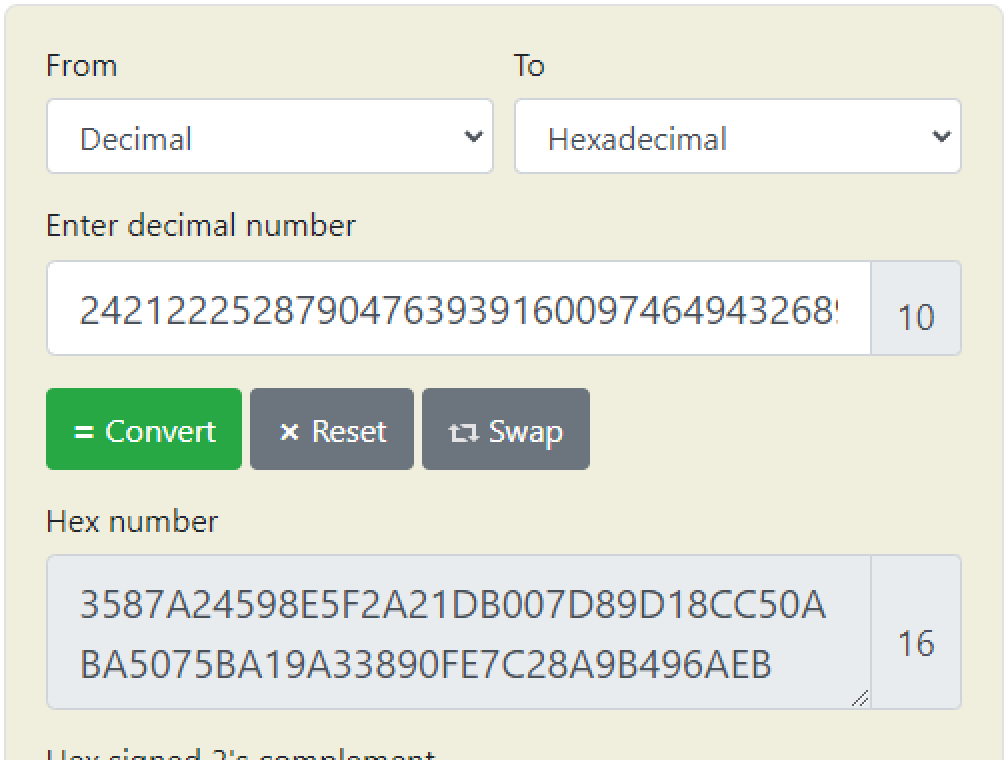
```
int main() {

BN_CTX *ctx = BN_CTX_new();

BIGNUM *p = BN_new();
BIGNUM *q = BN_new();
BIGNUM *e = BN_new();
BIGNUM *n = BN_new();

//assign the first prime

BN_hex2bn(&p,"F7E75FDC469067FFDC4E847C51F452DF");

//assign the second prime

BN_hex2bn(&q,"E85CED54AF57E53E092113E62F436F4F");

//Modulus

BN_hex2bn(&e,"0D88C3");

//finding N
BN_mul(n,p,q,ctx);

BIGNUM *pk1 = cal_rsa_priv_key(p,q,e);
printBN("Private key for task 1:",pk1);


};
```

- Setting up the final result



Compute `d`, the modular multiplicative inverse of `e` (mod `tot(n)`).

`d`:
```
2421222528790476393916009746494326893013
```


From Decimal
To Hexadecimal

Enter decimal number
24212225287904763939160097464943268 10

Hex number
3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB 16

Hex signed 2's complement

- Finally checking if my math was correct and the two keys match!
- **Success!**

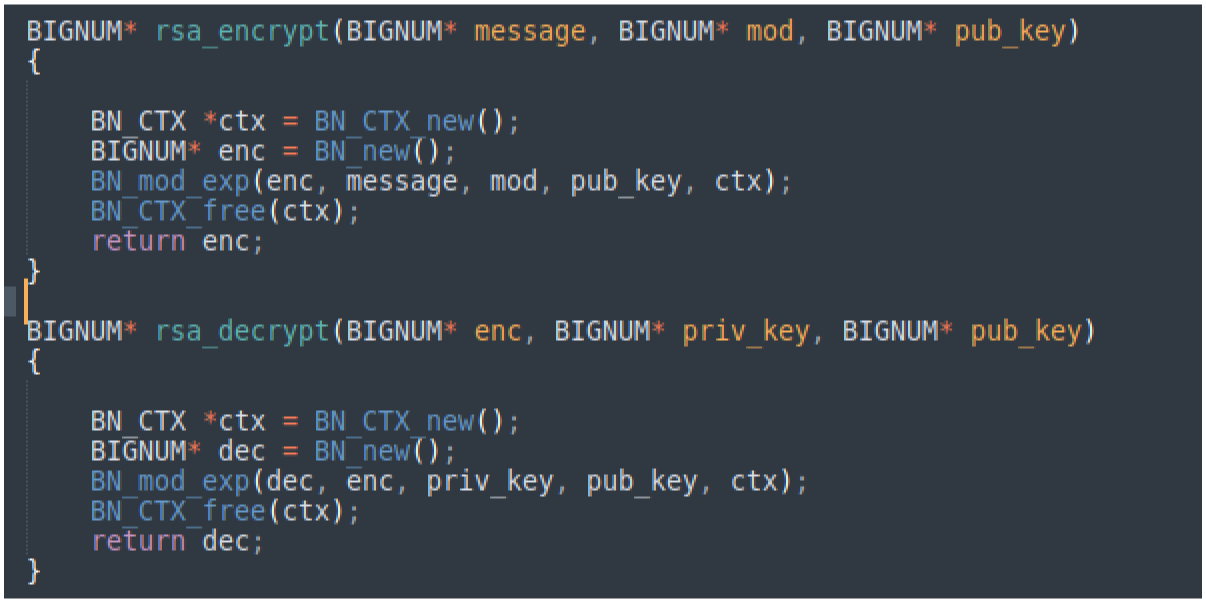
2. Encrypting a Message

```
the encrypted message for T2: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5F
ADC
```

- This is firstly showing the encrypted message for Task 2

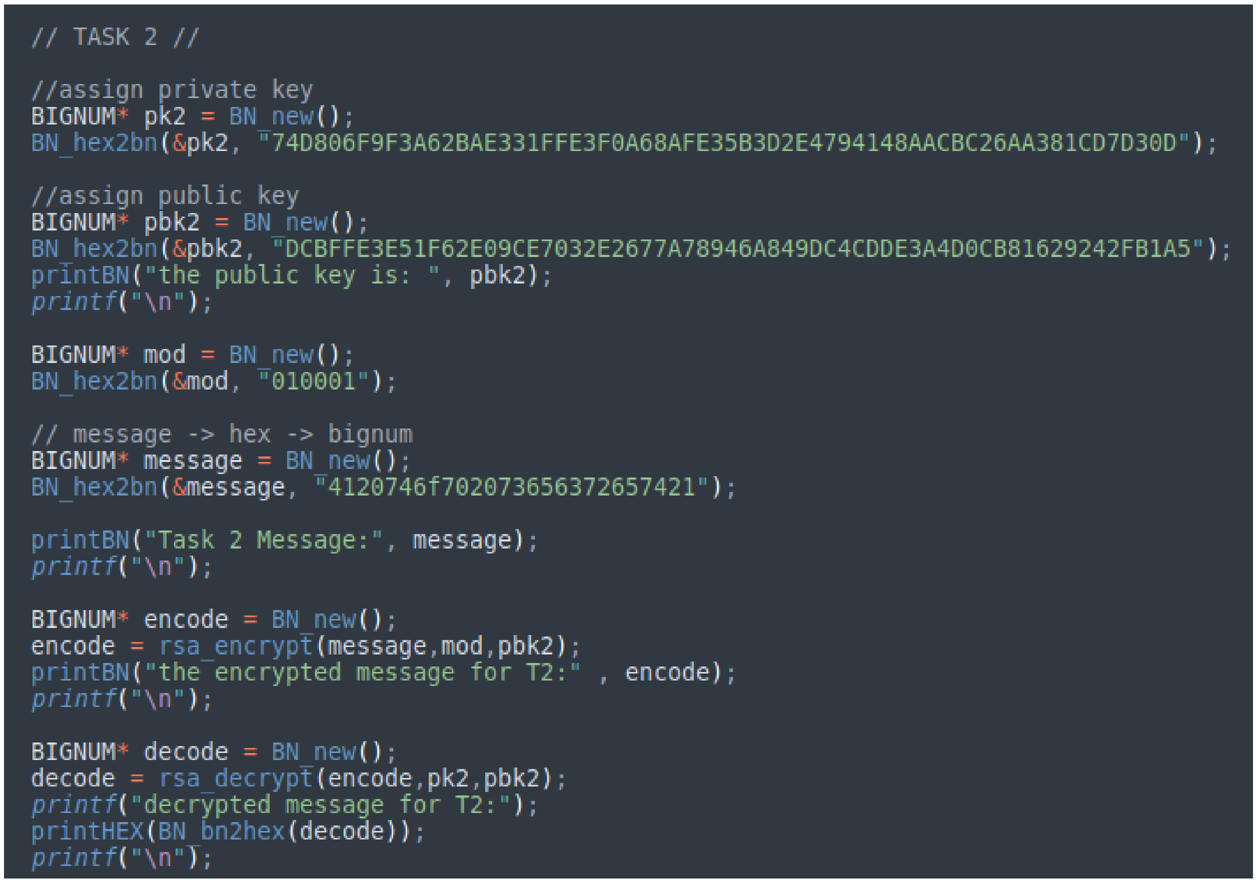
```
decrypted message for T2:A top secret!
```

- This is the fully decrypted message for Task 2



```
BIGNUM* rsa_encrypt(BIGNUM* message, BIGNUM* mod, BIGNUM* pub_key)
{

    BN_CTX *ctx = BN_CTX_new();
    BIGNUM* enc = BN_new();
    BN_mod_exp(enc, message, mod, pub_key, ctx);
    BN_CTX_free(ctx);
    return enc;
}

BIGNUM* rsa_decrypt(BIGNUM* enc, BIGNUM* priv_key, BIGNUM* pub_key)
{

    BN_CTX *ctx = BN_CTX_new();
    BIGNUM* dec = BN_new();
    BN_mod_exp(dec, enc, priv_key, pub_key, ctx);
    BN_CTX_free(ctx);
    return dec;
}
```

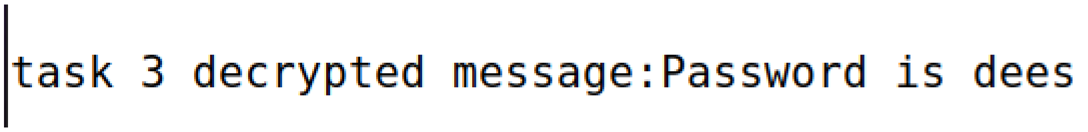
- Here my decryption and encryption and encryption functions are written
    - The encryption function accomplishes the following function using mod_exp
        - $enc \ = \ message^{modulus} mod(public\ key)$
    - The decryption function accomplishes a similar function but instead the decrypted formula is as follows
        - $dec \ = \ encoded^{privatekey} mod(public\ key)$
    - The decryption method allows the message to be decrypted if the user has access to the private key.



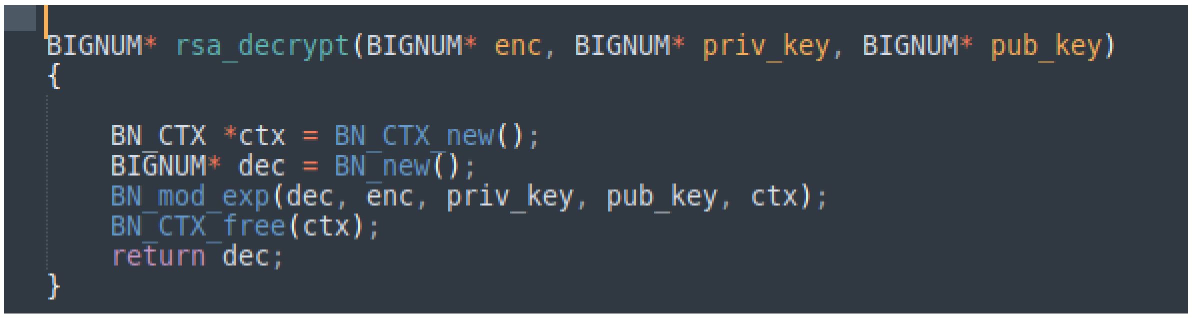
```c
// TASK 2 //

//assign private key
BIGNUM* pk2 = BN_new();
BN_hex2bn(&pk2, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

//assign public key
BIGNUM* pbk2 = BN_new();
BN_hex2bn(&pbk2, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
printBN("the public key is: ", pbk2);
printf("\n");

BIGNUM* mod = BN_new();
BN_hex2bn(&mod, "010001");

// message -> hex -> bignum
BIGNUM* message = BN_new();
BN_hex2bn(&message, "4120746f702073656372657421");

printBN("Task 2 Message:", message);
printf("\n");

BIGNUM* encode = BN_new();
encode = rsa_encrypt(message,mod,pbk2);
printBN("the encrypted message for T2:" , encode);
printf("\n");

BIGNUM* decode = BN_new();
decode = rsa_decrypt(encode,pk2,pbk2);
printf("decrypted message for T2:");
printHEX(BN_bn2hex(decode));
printf("\n");
```

- This is just the basic setup for task 2 using my functions that I created earlier.
- I encode the message, print the encoded message, then decode the message and print the original message.

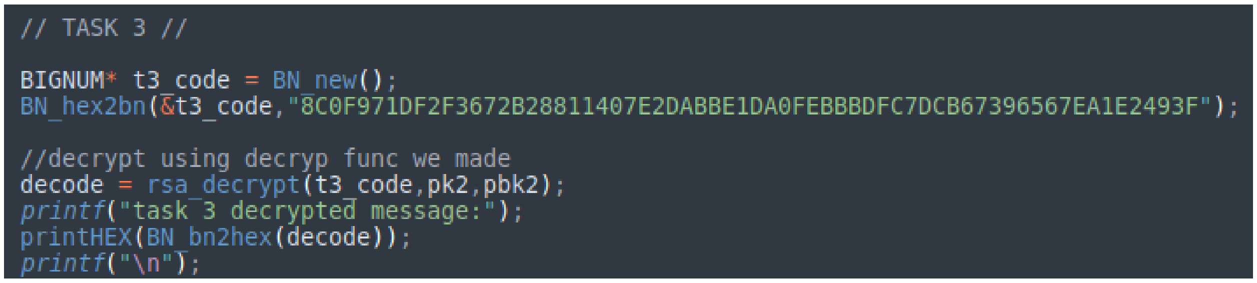
3. Decrypting a Message



```
task 3 decrypted message:Password is dees
```



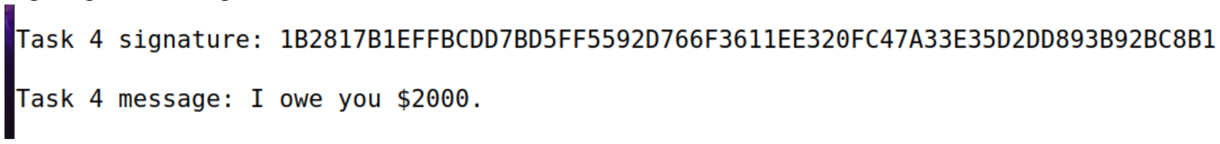
```c
BIGNUM* rsa_decrypt(BIGNUM* enc, BIGNUM* priv_key, BIGNUM* pub_key)
{

    BN_CTX *ctx = BN_CTX_new();
    BIGNUM* dec = BN_new();
    BN_mod_exp(dec, enc, priv_key, pub_key, ctx);
    BN_CTX_free(ctx);
    return dec;
}
```



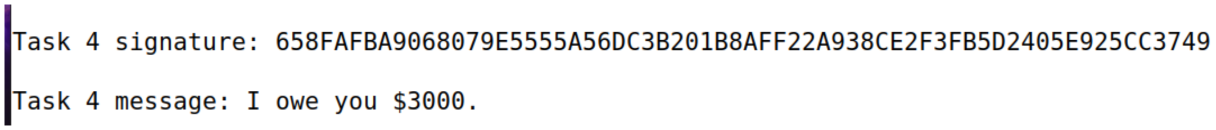
```
// TASK 3 //

BIGNUM* t3_code = BN_new();
BN_hex2bn(&t3_code,"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

//decrypt using decryp func we made
decode = rsa_decrypt(t3_code,pk2,pbk2);
printf("task 3 decrypted message:");
printHEX(BN_bn2hex(decode));
printf("\n");
```

- Since I already created a decryption function this task was relatively trivial.
- It displayed the message “Password is dees”
- This is funny because it is the password of the virtual machine that I am using.
- Yet again, since the private key and public keys are the same as in task 2, the decryption is allowed since I have access to the private key

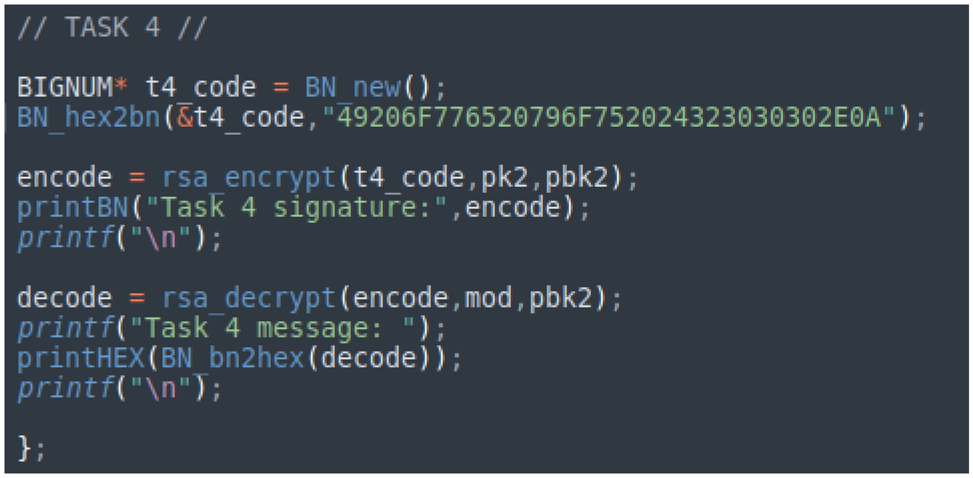
4. Signing a Message



```
Task 4 signature: 1B2817B1EFFBCDD7BD5FF5592D766F3611EE320FC47A33E35D2DD893B92BC8B1

Task 4 message: I owe you $2000.
```



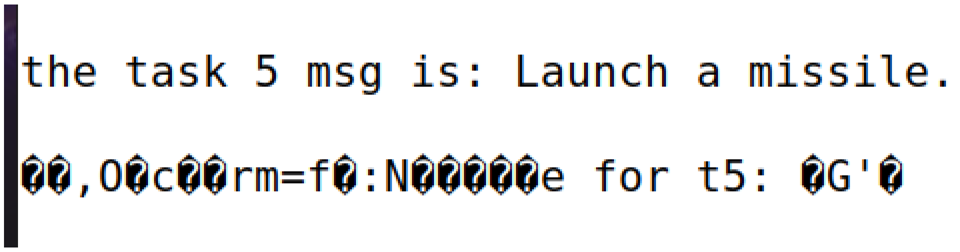
```
Task 4 signature: 658FAFBA9068079E5555A56DC3B201B8AFF22A938CE2F3FB5D2405E925CC3749

Task 4 message: I owe you $3000.
```



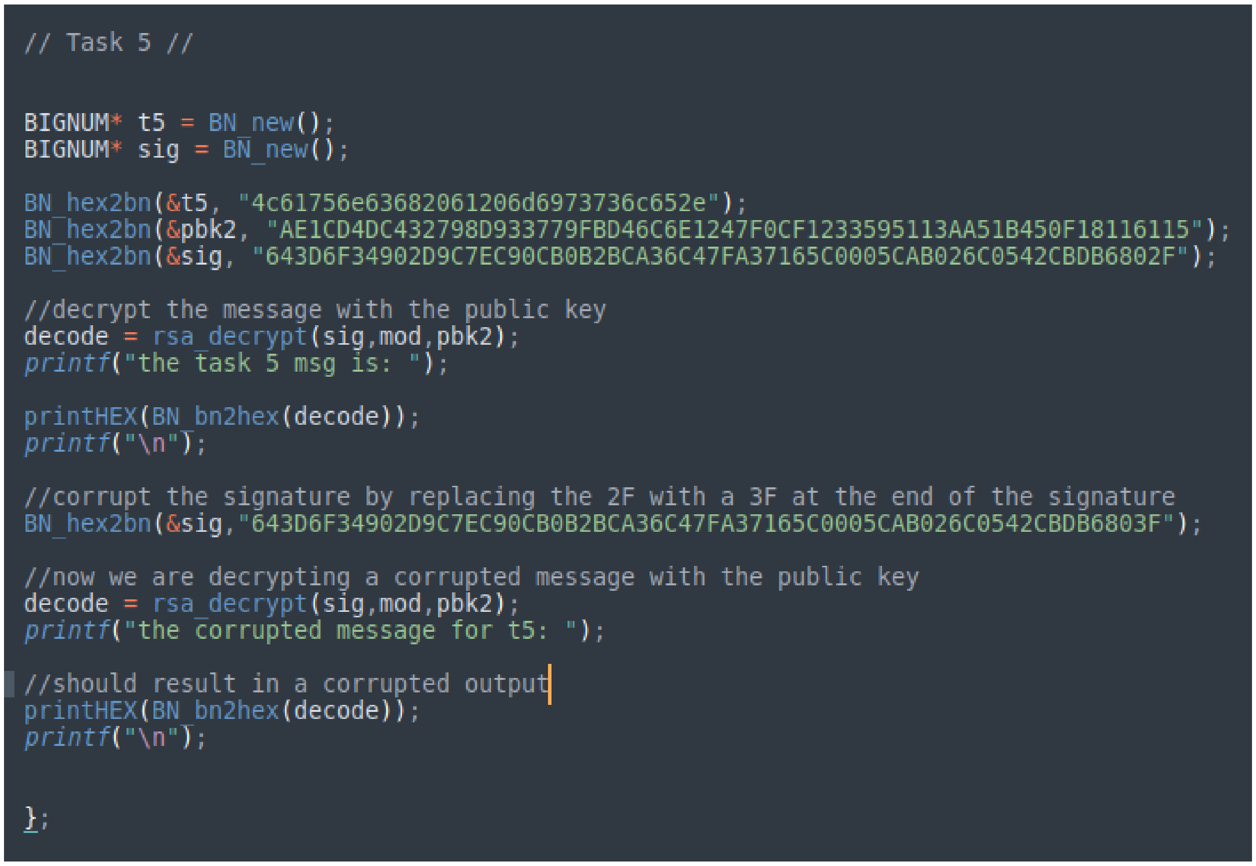
```
// TASK 4 //

BIGNUM* t4_code = BN_new();
BN_hex2bn(&t4_code,"49206F776520796F752024323030302E0A");

encode = rsa_encrypt(t4_code,pk2,pbk2);
printBN("Task 4 signature:",encode);
printf("\n");

decode = rsa_decrypt(encode,mod,pbk2);
printf("Task 4 message: ");
printHEX(BN_bn2hex(decode));
printf("\n");

};
```

- Here I show the signed message in both cases
- Case 1
    a. The calculation to get the signed message is as follows
    b. The encode variable that I use in this case signs the message with the encryption function because I am using the private key to write the message’s signature.
    c. I then use the public key to check if it is actually my message that I signed
- Case 2
    a. Case 2 is a bit similar, but changing the message will change the signature slightly but it still will function the same because I will still check the message if it is my own.

5. Verifying a Signature



```
the task 5 msg is: Launch a missile.

��,O�c��rm=f�:N�����e for t5: �G'�
```



```
// Task 5 //


BIGNUM* t5 = BN_new();
BIGNUM* sig = BN_new();

BN_hex2bn(&t5, "4c61756e63682061206d6973736c652e");
BN_hex2bn(&pbk2, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&sig, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

//decrypt the message with the public key
decode = rsa_decrypt(sig,mod,pbk2);
printf("the task 5 msg is: ");

printHEX(BN_bn2hex(decode));
printf("\n");

//corrupt the signature by replacing the 2F with a 3F at the end of the signature
BN_hex2bn(&sig,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

//now we are decrypting a corrupted message with the public key
decode = rsa_decrypt(sig,mod,pbk2);
printf("the corrupted message for t5: ");

//should result in a corrupted output
printHEX(BN_bn2hex(decode));
printf("\n");


};
```

Walking through my code

- I assign the task 5 message, public key and the signature.
- I decrypt the message first off by using Alice’s signature, the modulus provided and the public key, and I can see that the message “Launch as missile’ is correct.
- Now I corrupt Alice’s signature by replacing some bits on the end of the string and decrypting it again using the same parameters as before
- This is now interesting because when I print out the message again it becomes corrupted
- I think the reasoning for this is because Alice signed the message thus making the decryption of that message reliant upon a correct signature
- This is how the system will verify that the message is in fact Alice’s original message

- Since we change the signature in the last step, the system can not verify that it is Alice’s message correctly and displays a corrupted message.