

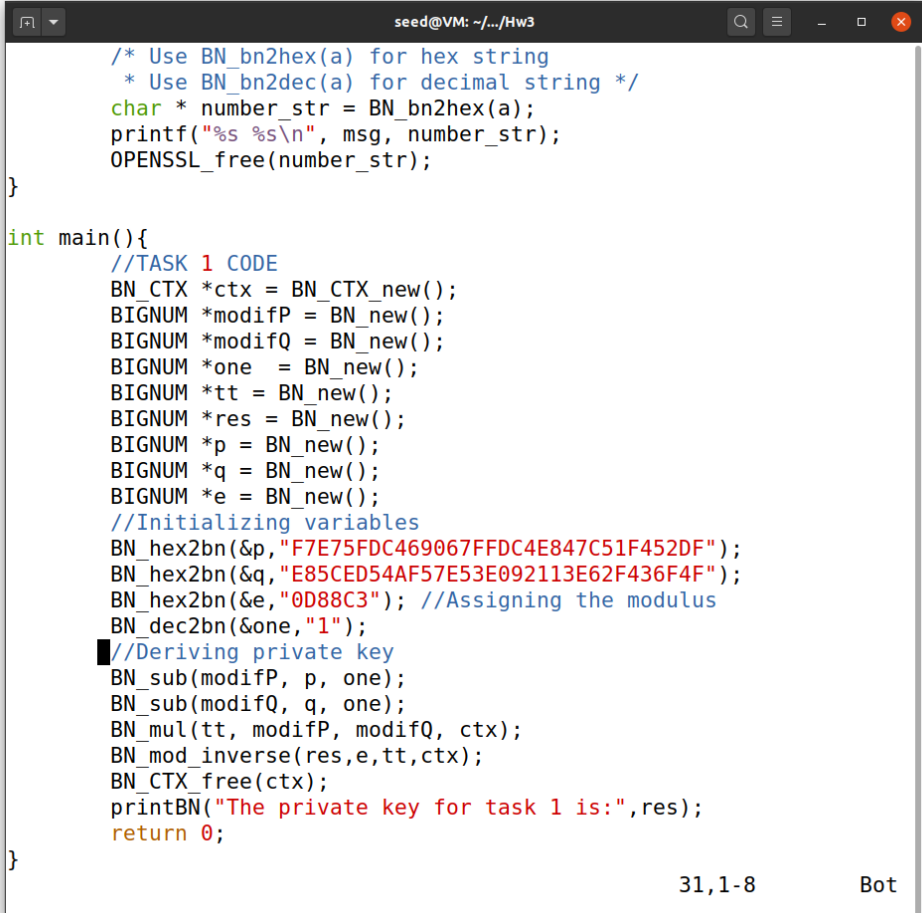
Purpose

The purpose of this lab is to practice and understand how the RSA cryptographic algorithm works. The tasks include verifying and signing messages with a personal signature. This lab also walks through how to obtain a public/private key and encode/decode messages using BIGNUM variables. BIGNUM variables that can store large numbers and be manipulated/performed arithmetic operations using APIs. These numbers are involved with RSA algorithms to help gain different values for the tasks.

Tasks**Task 1:**

For task 1, we used BIGNUM variables to derive the private key given a public key composed of the variables e and n. This was done by subtracting p and q by one, and then multiplying those values together. After that, the inverse of that value and the modulus is calculated. This result is the private key.

(Fig 1.1, Code for task 1)



```

seed@VM: ~/.../Hw3
/* 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 main(){
//TASK 1 CODE
BN_CTX *ctx = BN_CTX_new();
BIGNUM *modifP = BN_new();
BIGNUM *modifQ = BN_new();
BIGNUM *one = BN_new();
BIGNUM *tt = BN_new();
BIGNUM *res = BN_new();
BIGNUM *p = BN_new();
BIGNUM *q = BN_new();
BIGNUM *e = BN_new();
//Initializing variables
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3"); //Assigning the modulus
BN_dec2bn(&one, "1");
//Deriving private key
BN_sub(modifP, p, one);
BN_sub(modifQ, q, one);
BN_mul(tt, modifP, modifQ, ctx);
BN_mod_inverse(res, e, tt, ctx);
BN_CTX_free(ctx);
printBN("The private key for task 1 is:", res);
return 0;
}
31,1-8 Bot

```

(Fig 1.2, Output)

```
seed@VM: ~/.../Hw3
[03/14/21] seed@VM:~/.../Hw3$ gcc bn_sample.c -lcrypto
[03/14/21] seed@VM:~/.../Hw3$ ./a.out
The private key for task 1 is: 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890
FE7C28A9B496AEB
[03/14/21] seed@VM:~/.../Hw3$
```

Task 2:

Task 2 consisted of encrypting a simple message using hexadecimal. First, I converted the message to hexadecimal as directed. Next, I took the power of that message's hexadecimal value to the e-th power modulo n. This resulted in the encrypted message's hexadecimal value. In order to check my work, I decrypted that message by taking its value to the d-th power modulo n. Based on fig 2.2, I was able to successfully encrypt the message without corrupting the original message. Although the values are in hexadecimal, you can tell that they are the same values, and therefore the same message.

(Fig. 2.1, Code)

```
seed@VM: ~/.../Hw3
/* 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 main() {
    //TASK 2 CODE
    BN_CTX *ctx = BN_CTX_new();
    BN_CTX *ctx2 = BN_CTX_new();
    BIGNUM *encr = BN_new();
    BIGNUM *decr = BN_new();
    BIGNUM *d = BN_new(); //private key
    BIGNUM *n = BN_new(); //public key
    BIGNUM *e = BN_new(); //modulus
    BIGNUM *m = BN_new(); //Message in hex
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB8
1629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26
AA381CD7D30D");
    BN_hex2bn(&m, "4120746f702073656372657421");
    BN_mod_exp(encr, m, e, n, ctx);
    BN_CTX_free(ctx);
    printf("The encrypted message for task 2 is: ", encr);
    BN_mod_exp(decr, encr, d, n, ctx2);
    BN_CTX_free(ctx2);
    printf("The decrypted message for task 2 is: %s", BN_bn2hex(decr));
    return 0;
}

36,1 Bot
```

(Fig 2.2, Output)

```

seed@VM: ~/.../Hw3
[03/14/21] seed@VM:~/.../Hw3$ gcc bn_sample.c -lcrypto
[03/14/21] seed@VM:~/.../Hw3$ ./a.out
The encrypted message for task 2 is: 6FB078DA550B2650832661E14F4F8D2C
FAEF475A0DF3A75CACDC5DE5CFC5FADC
The decrypted message for task 2 is: 4120746F702073656372657421[03/14/
[03/14/21] seed@VM:~/.../Hw3$

```

Task 3:

Using the same public and private keys from task 2, I decrypted a given message, c , and converted it back to ASCII symbols in order to successfully read the message. This was done similarly to how I verified my encryption in the previous task. I raised the hexadecimal version of the encrypted message to the d -th value modulo e . Below is the code snippet and resulting output.

(Fig 3.1, Code for task 3)

```

#define NBITS 256

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);
}

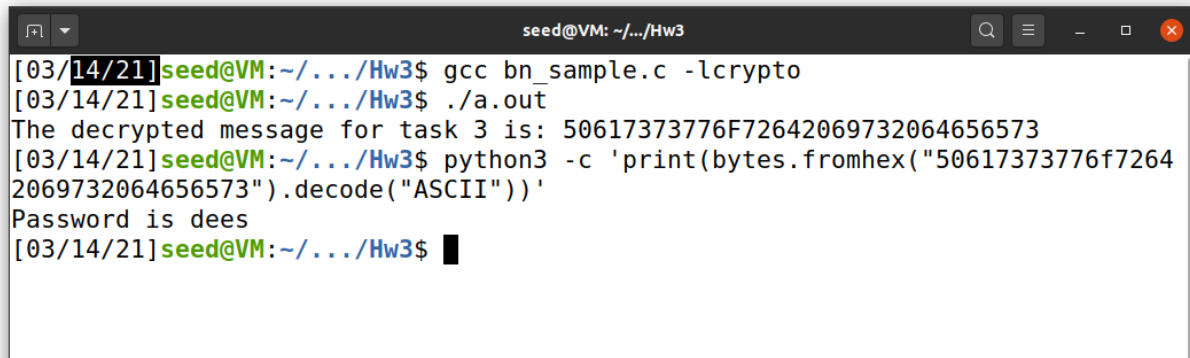
int main(){
    //TASK 3 CODE
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *decr = BN_new();
    BIGNUM *d = BN_new(); //private key
    BIGNUM *n = BN_new(); //public key
    BIGNUM *e = BN_new(); //modulus
    BIGNUM *c = BN_new(); //Encrypted Ciphertext
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629
242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA38
1CD7D30D");
    BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567E
A1E2493F");

    BN_mod_exp(decr, c, d, n, ctx);
    BN_CTX_free(ctx);
    printf("The decrypted message for task 3 is: %s\n", BN_bn2hex(decr));
    return 0;
}

```

"bn_sample.c" 32L, 940C 32,1 Bot

(Fig 3.2, output for task 3)



```
seed@VM: ~/.../Hw3
[03/14/21] seed@VM:~/.../Hw3$ gcc bn_sample.c -lcrypto
[03/14/21] seed@VM:~/.../Hw3$ ./a.out
The decrypted message for task 3 is: 50617373776F72642069732064656573
[03/14/21] seed@VM:~/.../Hw3$ python3 -c 'print(bytes.fromhex("50617373776f7264
2069732064656573").decode("ASCII"))'
Password is dees
[03/14/21] seed@VM:~/.../Hw3$
```

Task 4:

Task 4 consisted of generating a signature and using it to sign a message (provided by the lab). By putting a digital signature on a message, it helps validate that the message is coming from, uncorrupted, from the place it claims to be. Normally, a signature is placed on a message by taking the hash of the message and then encrypting that. However for simplicity, this task signed the message by using the given public key to convert the plaintext to ciphertext. This was done by taking the message to the e -th power modulo n . I also performed the same method but changed the method to \$5000 instead of \$2000 to see how it changed the signature. Below is the code and outputs for each message. Looking at Fig. 4.2 and 4.3 I noticed that although it's the same keys and modulus, the values are 5 (decimal) away from each other. This signifies that although the signatures change the value of the message, the signature keeps them in close proximity to each other. Another observation is that the 'I owe you \$5000' message is larger than the 'I owe you \$2000' message. This makes sense since 5 is greater than 3.

(Fig. 4.1, Code)

```

seed@VM: ~/.../Hw3
* 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 main(){
    //TASK 3 CODE
    BN_CTX *ctx = BN_CTX_new();
    BN_CTX *ctx2 = BN_CTX_new();
    BIGNUM *decr = BN_new();
    BIGNUM *encr = BN_new();
    BIGNUM *d = BN_new(); //private key
    BIGNUM *n = BN_new(); //public key
    BIGNUM *e = BN_new(); //modulus
    BIGNUM *m = BN_new(); //Message
    BIGNUM *m2 = BN_new(); //Modified message
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629
242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA38
1CD7D30D");
    BN_hex2bn(&m, "49206f776520796f75202432303030"); //Hex version of messa
ge

    BN_mod_exp(encr, m, e, n, ctx);
    BN_CTX_free(ctx);
    printBN("The signature for task 4 is: ", encr);
    BN_mod_exp(decr, encr, d, n, ctx2);
    BN_CTX_free(ctx2);
    printf("The decrypted message for task 4 is: %s\n", BN_bn2hex(decr));
    return 0;
}
"bn_sample.c" 37L, 1116C                                     35,31-38      Bot

```

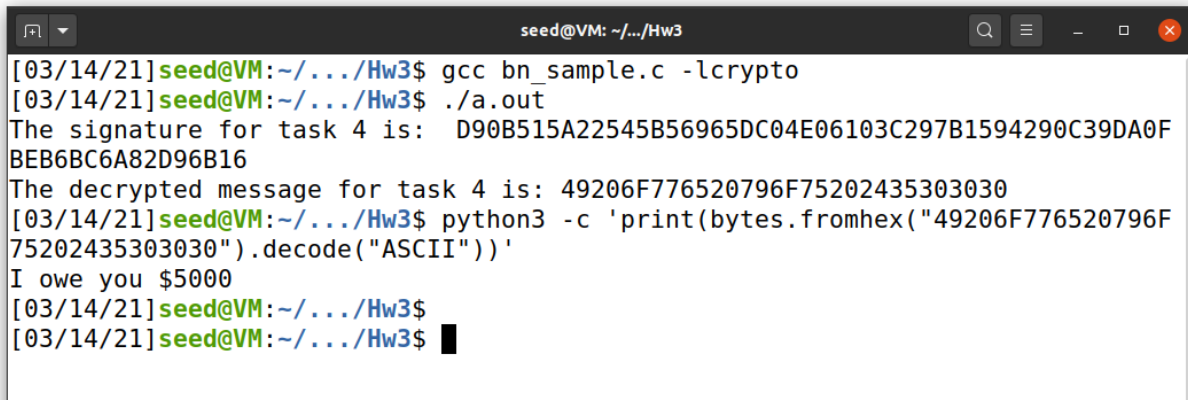
(Fig 4.2, Output for 'I owe you \$2000')

```

seed@VM: ~/.../Hw3
[03/14/21]seed@VM:~/.../Hw3$ gcc bn_sample.c -lcrypto
[03/14/21]seed@VM:~/.../Hw3$ ./a.out
The signature for task 4 is: 16CDC2D574C9FDCF64A9E387F9EF69AB8BF9D6B839ABCDBF
617EF41BA12BE37B
The decrypted message for task 4 is: 49206f776520796f75202432303030
[03/14/21]seed@VM:~/.../Hw3$ python3 -c 'print(bytes.fromhex("49206f776520796f
75202432303030").decode("ASCII"))'
I owe you $2000
[03/14/21]seed@VM:~/.../Hw3$ vim bn_sample.c
[03/14/21]seed@VM:~/.../Hw3$

```

(Fig. 4.3, output for 'I owe you \$5000')

A terminal window titled 'seed@VM: ~/.../Hw3' showing the execution of a C program and a Python script. The C program outputs a signature and a decrypted message. The Python script then decodes the message into the text 'I owe you \$5000'.

```
[03/14/21] seed@VM:~/.../Hw3$ gcc bn_sample.c -lcrypto
[03/14/21] seed@VM:~/.../Hw3$ ./a.out
The signature for task 4 is: D90B515A22545B56965DC04E06103C297B1594290C39DA0F
BEB6BC6A82D96B16
The decrypted message for task 4 is: 49206F776520796F75202435303030
[03/14/21] seed@VM:~/.../Hw3$ python3 -c 'print(bytes.fromhex("49206F776520796F
75202435303030").decode("ASCII"))'
I owe you $5000
[03/14/21] seed@VM:~/.../Hw3$
[03/14/21] seed@VM:~/.../Hw3$
```

Task 5:

Task 5 involves verifying that a signature is the expected signature. This helps authenticate the sender of the message. After verifying this message, I compared that output to the one if one byte was changed from 2F to 3F in Alice's signature. To generate Alice's message via her signature, I took the e -th power of the given signature mod n . This generated the expected message, 'Launch a missile.' I repeated the same steps but with the corrupted signature (last byte changed to 3F). This resulted in an error stating that the message was not able to be decoded by the python ASCII codec. By changing one byte in the signature, the entire message became unreadable. During the verification process the value likely changed how python decodes because it became out of range of the ASCII.

```

seed@VM: ~/.../Hw3

char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}

int main(){
    //TASK 5 CODE
    BN_CTX *ctx = BN_CTX_new();
    BN_CTX *ctx2 = BN_CTX_new();
    BIGNUM *decr = BN_new();
    BIGNUM *n = BN_new(); //public key
    BIGNUM *e = BN_new(); //modulus
    BIGNUM *S = BN_new(); //Signature
    BIGNUM *m = BN_new(); //Message
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F
18116115");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&m, "4c61756e63682061206d697373696c652e");
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542C
BDB6802F");
    BN_mod_exp(decr, S, e, n, ctx);
    BN_CTX_free(ctx);
    printBN("The message for task 5 is: ", decr);
    //Corrupted signature
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542C
BDB6803F");
    BN_mod_exp(decr, S, e, n, ctx2);
    BN_CTX_free(ctx2);
    printf("The message for task 5 is (corrupted signature): %s\n", BN_bn2h
ex(decr));
    return 0;
}

```

37,1 Bot

(Fig 5.2, Output (normal and corrupted))

```

seed@VM: ~/.../Hw3

[03/14/21] seed@VM: ~/.../Hw3$ gcc bn_sample.c -lcrypto
[03/14/21] seed@VM: ~/.../Hw3$ ./a.out
The message for task 5 is: 4C61756E63682061206D697373696C652E
The message for task 5 is (corrupted signature): 91471927C80DF1E42C154FB4638CE
8BC726D3D66C83A4EB6B7BE0203B41AC294
[03/14/21] seed@VM: ~/.../Hw3$ python3 -c 'print(bytes.fromhex("4C61756E63682061
206D697373696C652E").decode("ASCII"))'
Launch a missile.
[03/14/21] seed@VM: ~/.../Hw3$ python3 -c 'print(bytes.fromhex("91471927C80DF1E4
2C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294").decode("ASCII"))'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0x91 in position 0: ordina
l not in range(128)
[03/14/21] seed@VM: ~/.../Hw3$

```


Task 6:

Task 6 involves downloading certificates from a website and manually verifying the X.509 certificate. After following the steps outlined in the lab, I was able to find the issuer's public key, and the signature and body of the server's certificate. By getting the body of the certificate, I was able to calculate the body's hash. By doing this, I was able to determine what part of the certificate is used to generate the hash. If the hash from the given certificate and the server's certificate, the signature will be valid. I am able to get the hash from the server's signature by decrypting the signature using the issuer's public key and the modulus (signature to the e -th power mod n). Fig 6.2 shows the output. Because the last bytes and the hash match, I am able to verify the signature of the certificate. Fig 6.1 shows the code I used to generate the output (outside of the commands used from the lab).

(Fig 6.1, Code)

```

seed@VM: ~/.../Hw3
}
    OPENSSL_free(number_str);
}

int main(){
    //TASK 6 CODE
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *decr = BN_new();
    BIGNUM *nIss = BN_new(); //modulus
    BIGNUM *e = BN_new(); //public key
    BIGNUM *Sig = BN_new(); //Signature
    BN_hex2bn(&nIss, "C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4
A66461F2C0AEC6407E52EDCDB90A20EDDFE3C4D09E9AA97A1D8288E51156DB1E9F58C
251E72C340D2ED292E156CBF1795FB3BB87CA25037B9A52416610604F571349F0E8376
783DFE7D34B674C2251A6DF0E9910ED57517426E27DC7CA622E131B7F238825536FC13
458008B84FFF8BEA75849227B96ADA2889B15BCA07CDFE951A8D5B0ED37E236B4824B6
2B5499AECC767D6E33EF5E3D6125E44F1BF71427D58840380B18101FAF9CA32BBB48E2
78727C52B74D4A8D697DEC364F9CACE53A256BC78178E490329AEFB494FA415B9CEF25
C19576D6B79A72BA2272013B5D03D40D321300793EA99F5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&Sig, "a72a10305cb86b7a1bf86638f6e9a00ad5138282f86589
57a5b8eb13291d846cecfbe30511d71e315e0ee2c000e56d0648be3d556fbab71135b6
eac4cf84f1304cbb339e11172bc9d2194b2cd0ad5f172384e1df17a23ba87f69297c48
a6615f263f75e23b5ba336b31ccde30457301ffcc9fa4b8e488058279ca2c7c326dc17
02fae66cea81015c928fd3180817707ac2a34b6c3afae3cff6fe7ec956e5a54e1b144f
a9989d79b11ec3abb10d1585a946b6e5c258e85afec814286890c6b8c8947fe10f89fa
a7d60937a162b70027b5bef1b15e452806b35415e6c3c8ac8201ce86e22be17ae4bd4c
cb9c5ed062c261bd8b5a62b67630bc460fe34523c0645f");
    BN_mod_exp(decr, Sig, e, nIss, ctx);
    BN_CTX_free(ctx);
    printBN("the task 6 is: ", decr);
    return 0;
}

```

29, 1 Bot

[illegible]