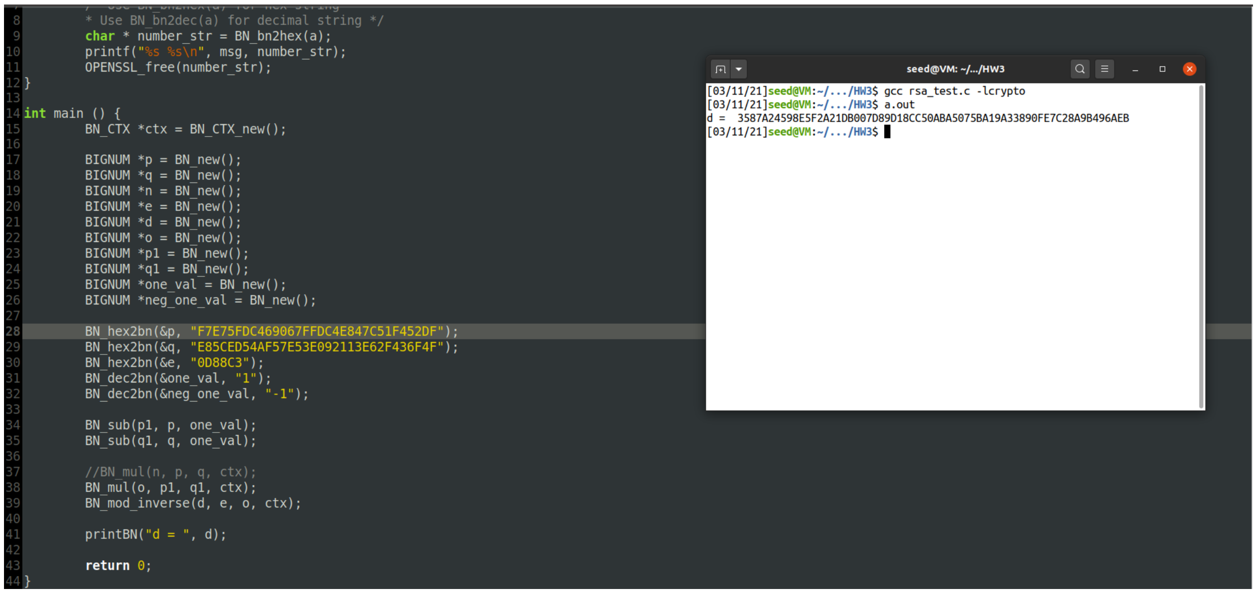
John Sikes

CSCE 465 - HW 3

<u>Task 1:</u>

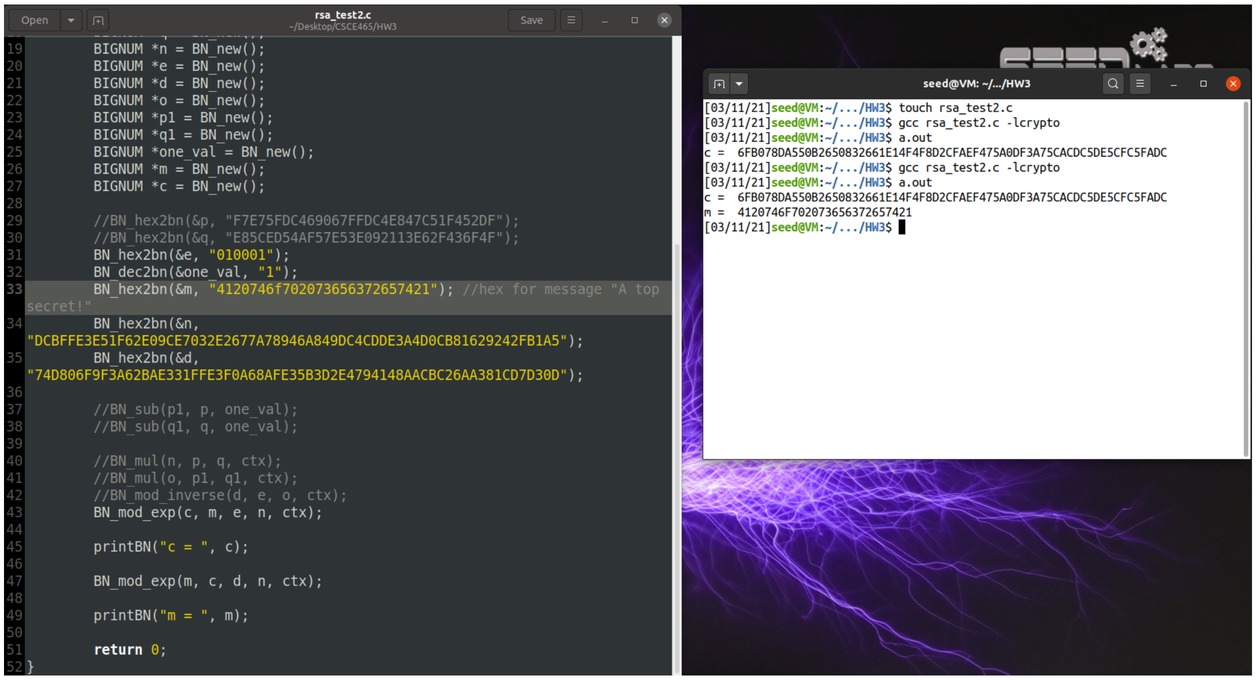


```c
         * Use BN_bn2dec(a) for decimal string */
        char * number_str = BN_bn2hex(a);
        printf("%s %s\n", msg, number_str);
        OPENSSL_free(number_str);
}

int main () {
        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *p = BN_new();
        BIGNUM *q = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *o = BN_new();
        BIGNUM *p1 = BN_new();
        BIGNUM *q1 = BN_new();
        BIGNUM *one_val = BN_new();
        BIGNUM *neg_one_val = BN_new();

        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&e, "0D88C3");
        BN_dec2bn(&one_val, "1");
        BN_dec2bn(&neg_one_val, "-1");

        BN_sub(p1, p, one_val);
        BN_sub(q1, q, one_val);

        //BN_mul(n, p, q, ctx);
        BN_mul(o, p1, q1, ctx);
        BN_mod_inverse(d, e, o, ctx);

        printBN("d = ", d);

        return 0;
}
```

```
[03/11/21]seed@VM:~/.../HW3$ gcc rsa_test.c -lcrypto
[03/11/21]seed@VM:~/.../HW3$ a.out
d =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[03/11/21]seed@VM:~/.../HW3$ 
```

In this task, I calculated the private key value by setting up the determination of (p-1)*(q-1) as the modulus for the following equation. This required first declaring the number 1 as a BIGNUM variable, then applying it to the values of p and q with the subtraction function. From here, it was as simple as multiplying their subtracted values as the modulus, which was then used in the inverse modulo function to determine d as a result of the formula d*e = 1 mod((p-1)*(q-1)). The value of this function was output to the terminal shown.

Task 2:

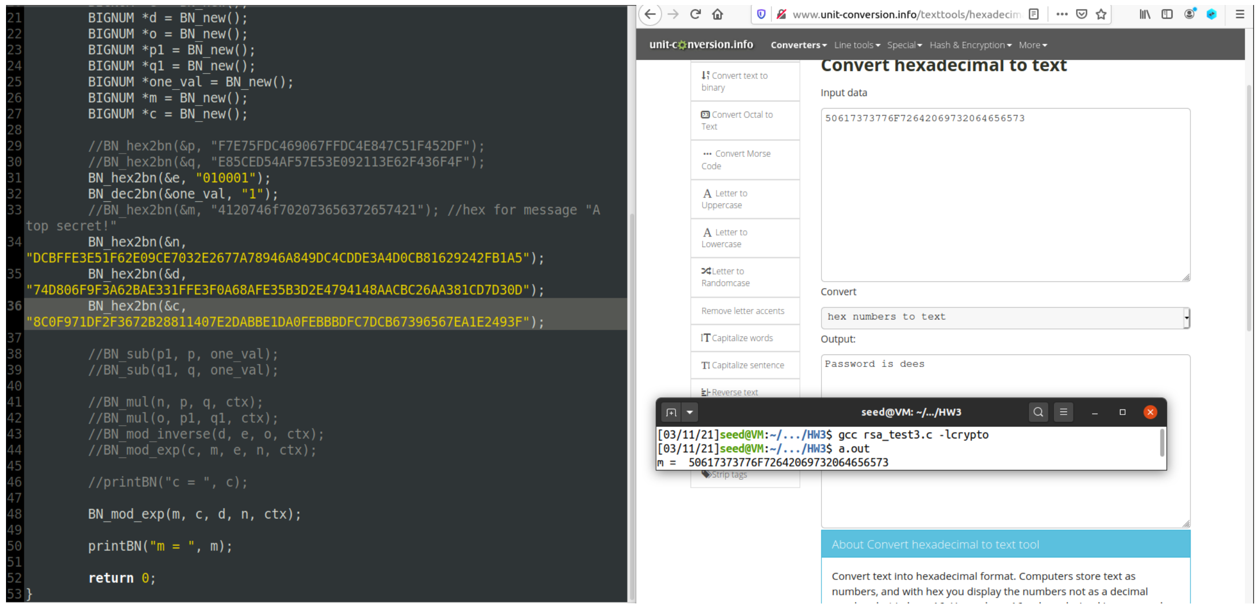


```
        BIGNUM *n = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *o = BN_new();
        BIGNUM *p1 = BN_new();
        BIGNUM *q1 = BN_new();
        BIGNUM *one_val = BN_new();
        BIGNUM *m = BN_new();
        BIGNUM *c = BN_new();

        //BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        //BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&e, "010001");
        BN_dec2bn(&one_val, "1");
        BN_hex2bn(&m, "4120746f702073656372657421"); //hex for message "A top secret!"
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

        //BN_sub(p1, p, one_val);
        //BN_sub(q1, q, one_val);

        //BN_mul(n, p, q, ctx);
        //BN_mul(o, p1, q1, ctx);
        //BN_mod_inverse(d, e, o, ctx);
        BN_mod_exp(c, m, e, n, ctx);

        printBN("c = ", c);

        BN_mod_exp(m, c, d, n, ctx);

        printBN("m = ", m);

        return 0;
}
```

```
[03/11/21]seed@VM:~/.../HW3$ touch rsa_test2.c
[03/11/21]seed@VM:~/.../HW3$ gcc rsa_test2.c -lcrypto
[03/11/21]seed@VM:~/.../HW3$ a.out
c =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
[03/11/21]seed@VM:~/.../HW3$ gcc rsa_test2.c -lcrypto
[03/11/21]seed@VM:~/.../HW3$ a.out
c =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
m =  4120746F702073656372657421
[03/11/21]seed@VM:~/.../HW3$ 
```

This task required me to encrypt a message, which first got converted to a hex string and used as input for encryption. To do this, I used the provided values to perform the encryption using the modular exponent function to determine the encrypted message, c. To confirm that this performed properly, I also performed decryption of the message using the same function but in reverse, using d as the exponential value instead of e, and determined the encrypted string yielded the same message when decrypted.

Task 3:



```c
        BIGNUM *d = BN_new();
        BIGNUM *o = BN_new();
        BIGNUM *p1 = BN_new();
        BIGNUM *q1 = BN_new();
        BIGNUM *one_val = BN_new();
        BIGNUM *m = BN_new();
        BIGNUM *c = BN_new();

        //BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        //BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&e, "010001");
        BN_dec2bn(&one_val, "1");
        //BN_hex2bn(&m, "4120746f702073656372657421"); //hex for message "A
top secret!"
        BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&c,
"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

        //BN_sub(p1, p, one_val);
        //BN_sub(q1, q, one_val);

        //BN_mul(n, p, q, ctx);
        //BN_mul(o, p1, q1, ctx);
        //BN_mod_inverse(d, e, o, ctx);
        //BN_mod_exp(c, m, e, n, ctx);

        //printBN("c = ", c);

        BN_mod_exp(m, c, d, n, ctx);

        printBN("m = ", m);

        return 0;
}
```

Convert hexadecimal to text

Input data

50617373776F72642069732064656573

Convert: hex numbers to text

Output:

Password is dees

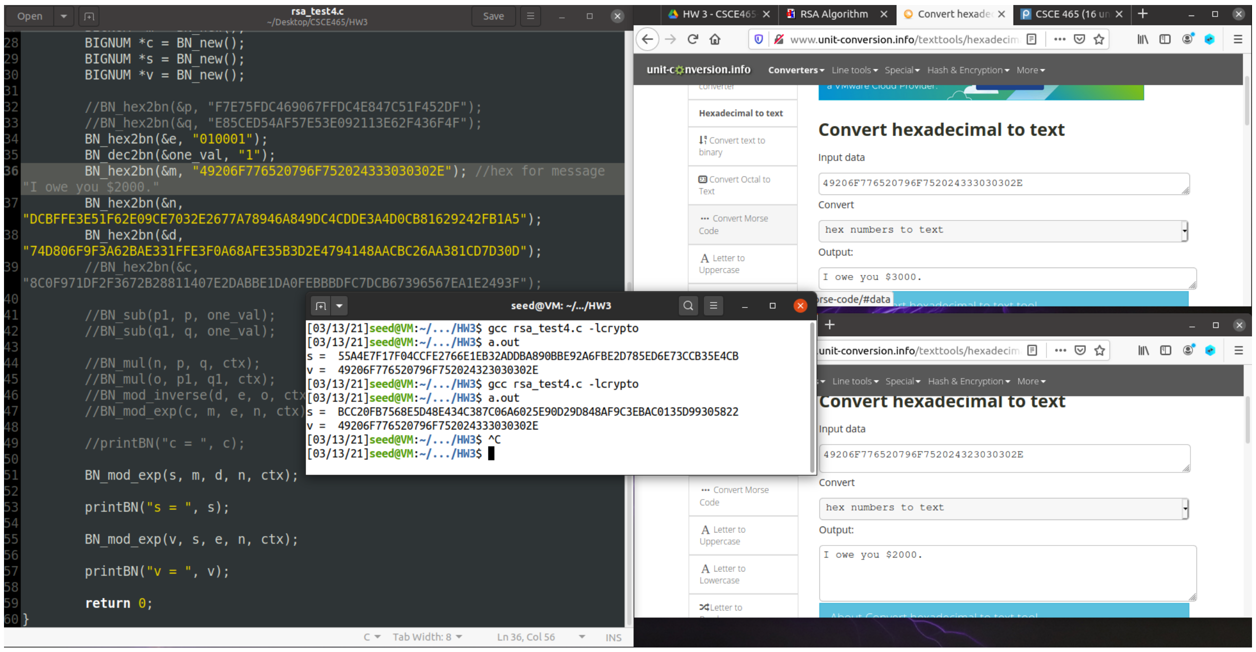
About Convert hexadecimal to text tool

Convert text into hexadecimal format. Computers store text as numbers, and with hex you display the numbers not as a decimal

```
seed@VM: ~/.../HW3
[03/11/21]seed@VM:~/.../HW3$ gcc rsa_test3.c -lcrypto
[03/11/21]seed@VM:~/.../HW3$ a.out
m =  50617373776F72642069732064656573
```

Similar to what I performed as a check in the previous task, I decrypted the provided encrypted message using the modular exponent function and the private key to yield the original message. To confirm this, I ran the output hex string through a converter (in place of the python commands, which I couldn't get to work) and confirmed that the output message was the same as what was originally sent.

Task 4:



```c
        BIGNUM *c = BN_new();
        BIGNUM *s = BN_new();
        BIGNUM *v = BN_new();

        //BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        //BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&e, "010001");
        BN_dec2bn(&one_val, "1");
        BN_hex2bn(&m, "49206F776520796F752024333030302E"); //hex for message
"I owe you $2000."
        BN_hex2bn(&n,
"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        //BN_hex2bn(&c,
"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

        //BN_sub(p1, p, one_val);
        //BN_sub(q1, q, one_val);

        //BN_mul(n, p, q, ctx);
        //BN_mul(o, p1, q1, ctx);
        //BN_mod_inverse(d, e, o, ctx
        //BN_mod_exp(c, m, e, n, ctx)

        //printBN("c = ", c);

        BN_mod_exp(s, m, d, n, ctx);

        printBN("s = ", s);

        BN_mod_exp(v, s, e, n, ctx);

        printBN("v = ", v);

        return 0;
}
```

```
[03/13/21]seed@VM:~/.../HW3$ gcc rsa_test4.c -lcrypto
[03/13/21]seed@VM:~/.../HW3$ a.out
s =  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
v =  49206F776520796F752024323030302E
[03/13/21]seed@VM:~/.../HW3$ gcc rsa_test4.c -lcrypto
[03/13/21]seed@VM:~/.../HW3$ a.out
s =  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
v =  49206F776520796F752024333030302E
[03/13/21]seed@VM:~/.../HW3$ ^C
[03/13/21]seed@VM:~/.../HW3$ 
```

Convert hexadecimal to text

Input data: 49206F776520796F752024333030302E

Convert: hex numbers to text

Output: I owe you $3000.

Convert hexadecimal to text

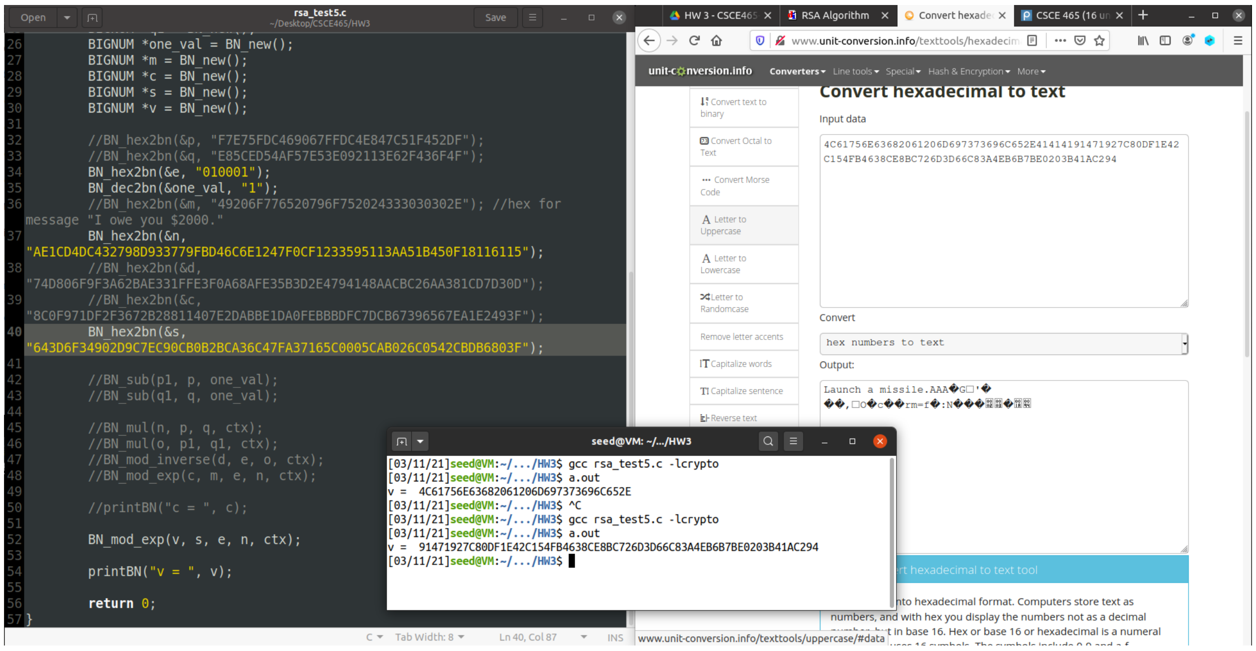
Input data: 49206F776520796F752024323030302E

Convert: hex numbers to text

Output: I owe you $2000.

Similar to the encryption step in Task 2, this task requested that I sign the message being sent. To do this, I signed the message using a private key and sending the message to the recipient. To verify that the signature matched what was expected, I had the message decrypted into the original message against the signature using regular decryption via the public key e. This is demonstrated as functioning properly in the associated hex to text converter on the right for each of the messages sent.

Task 5:



```
        BIGNUM *one_val = BN_new();
        BIGNUM *m = BN_new();
        BIGNUM *c = BN_new();
        BIGNUM *s = BN_new();
        BIGNUM *v = BN_new();

        //BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        //BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&e, "010001");
        BN_dec2bn(&one_val, "1");
        //BN_hex2bn(&m, "49206F776520796F752024333030302E"); //hex for
message "I owe you $2000."
        BN_hex2bn(&n,
"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
        //BN_hex2bn(&d,
"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        //BN_hex2bn(&c,
"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
        BN_hex2bn(&s,
"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

        //BN_sub(p1, p, one_val);
        //BN_sub(q1, q, one_val);

        //BN_mul(n, p, q, ctx);
        //BN_mul(o, p1, q1, ctx);
        //BN_mod_inverse(d, e, o, ctx);
        //BN_mod_exp(c, m, e, n, ctx);

        //printBN("c = ", c);

        BN_mod_exp(v, s, e, n, ctx);

        printBN("v = ", v);

        return 0;
}
```

```
[03/11/21]seed@VM:~/.../HW3$ gcc rsa_test5.c -lcrypto
[03/11/21]seed@VM:~/.../HW3$ a.out
v =  4C61756E63682061206D697373696C652E
[03/11/21]seed@VM:~/.../HW3$ ^C
[03/11/21]seed@VM:~/.../HW3$ gcc rsa_test5.c -lcrypto
[03/11/21]seed@VM:~/.../HW3$ a.out
v =  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
[03/11/21]seed@VM:~/.../HW3$
```

Convert hexadecimal to text

Input data

```
4C61756E63682061206D697373696C652E41414191471927C80DF1E42
C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
```

Convert: hex numbers to text

Output:

```
Launch a missile.AAA�G□'�
��,□o�c��rm=f�:N���□□�□□
```

Finally, in this task I did the same as I did in my checking step in Task 4 where I decrypted the signed message received using the public key to verify that the message was correct as well as the signature. To make sure this worked, I sent this through the same hex to text converter and verified that the decryption algorithm worked as expected. Similarly, I also altered the signed message and ran it through the converter and got corrupted information as a result, illustrated in the attached image as well.