Jared Ren
Final lab
5/8/20

Problem 4

Task 1.

```
[05/13/20]seed@VM:~$ dig @a.root-servers.net www.example.net
```

Task 2.

```
[05/13/20]seed@VM:~$ ping google.com
PING google.com (172.217.4.206) 56(84) bytes of data.
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
42.5 ms
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
30.7 ms
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
31.4 ms
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
31.6 ms
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
30.8 ms
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
31.9 ms
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
31.1 ms
64 bytes from ord37s19-in-f14.1e100.net (172.217.4.206): icmp
30.7 ms
^C
--- google.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7016m
rtt min/avg/max/mdev = 30.747/32.626/42.531/3.765 ms
[05/13/20]seed@VM:~$ ping facebook.com
PING facebook.com (157.240.26.35) 56(84) bytes of data.
```

Task 3.

```
[05/13/20]seed@VM:~$ dig www.example.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10506
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        73595   IN      A       93.184.216.34

;; Query time: 24 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Wed May 13 01:15:06 EDT 2020
;; MSG SIZE  rcvd: 60
```

Questions

- **What is DNS?**

  The domain naming system, the author refers to it as the phone book for the internet. Where domain names are given IP addresses.

- **Describe the process of executing a DNS query.**

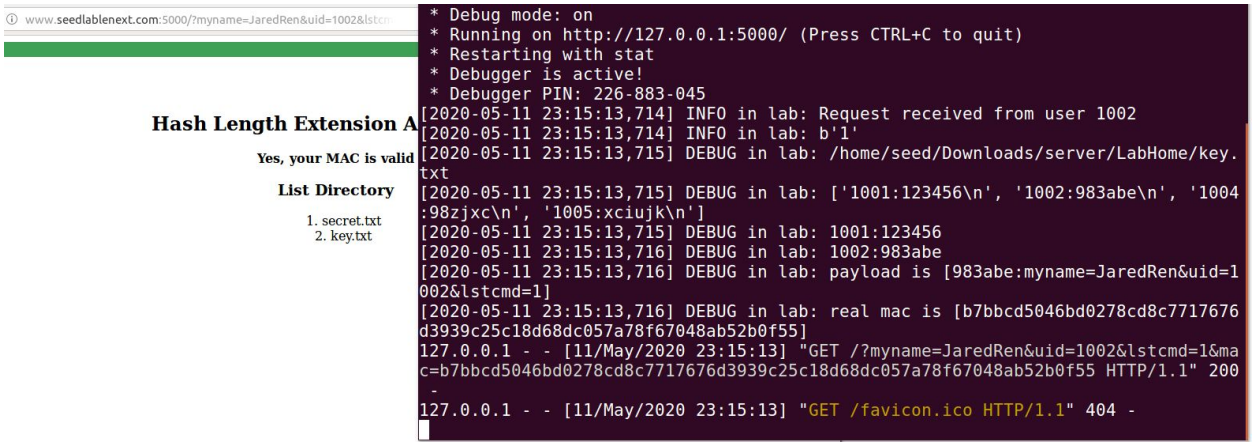
  When an application needs to communicate with another machine it will go to the DNS resolver to find the IP address of the other machine. DNS resolver searches its own data for the IP address, if it cannot find anything it goes to the DNS server which checks its own data. If the DNS server does not find the IP address the application requires it will go to other DNS servers to find the IP address needed. When that address is found the server returns it to the application.

- **What is a local DNS cache poisoning attack?**

  A local DNS cache poisoning attack is where an attacker on the same network as the local DNS server or a user machine they can eavesdrop on network traffic then sends a forged DNS reply to the DNS server. The server does not distinguish the forged reply and puts it into the cache for the server. The poisoning part is where the forged reply is stored in the DNS server's cache.

PROBLEM 6

Task 1.




Questions:

- **Describe how a hash algorithm works.**

The two big hash algorithm series MD and SHA-0 upt to SHA-3 all rely on a construction structure called Merkle-Damgård construction. The construction structure goes as such: the initialization vector and the original input are fed into the compression function for a number of times. With the output of each compression function being fed into the compression function again.

- **Describe the SHA series of hash functions (with specifics about each individual algorithm in the series).**

The SHA series of hash functions are published by the National Institute of Standards and Technology. Every SHA iteration follows the Merkle-Damgård construction except for SHA-3.

SHA-0: published then retracted because of an undisclosed significant flaw.

SHA-1: a 160 bit hash function, meaning it generates a 160 bit hash, made by the NSA, eventually broken by researchers in 2017.

SHA-2: a family of functions made by the NSA. There are many versions in this family depending on the size of the hash needed. The two main ones are SHA-256 and SHA-512.

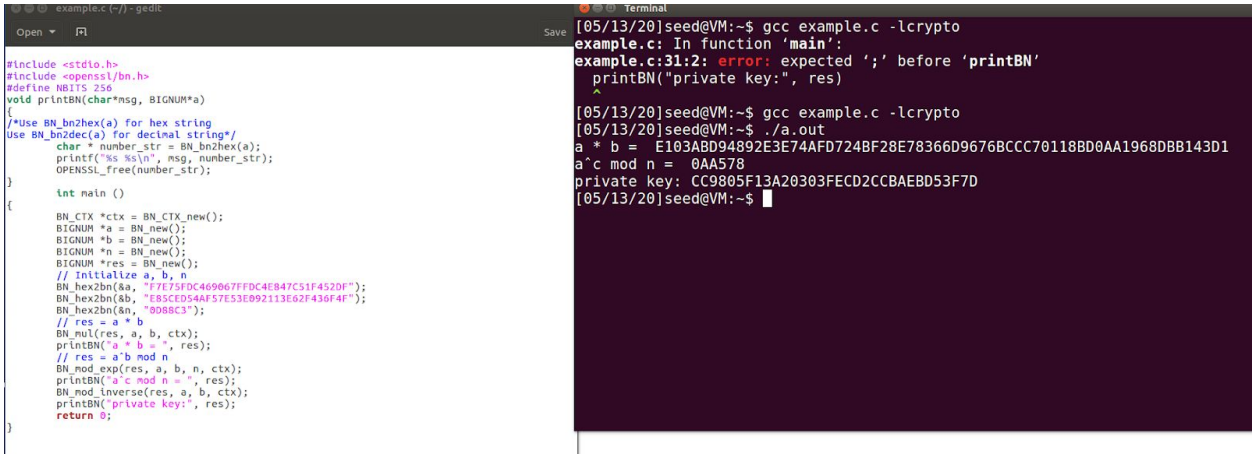
SHA-3: the newest iteration in the series, not made by the NSA instead by 4 people. It is also not based on Merkle-Damgård construction but on a new system called Keccak.

- **What is Message Authentication Code?**

A message authentication code is a hash value that is attached to data sent from the sender to the receiver. It is created from a key and message along with the initialization vector when they are fed into a compression function.

Problem 7

## Task 1. Deriving the private key

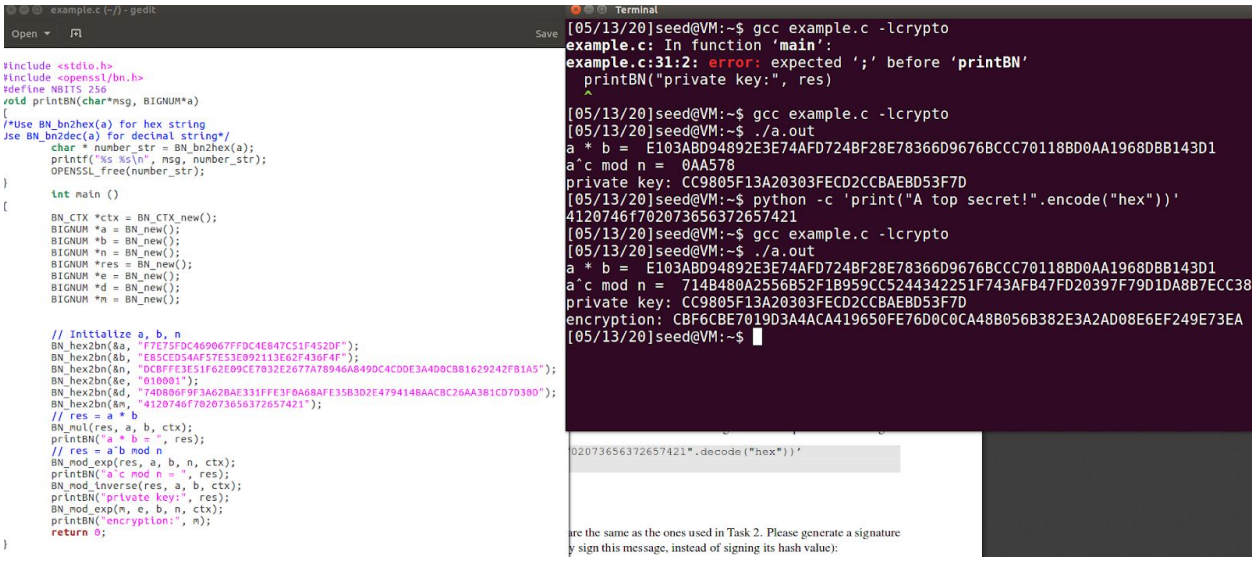


```
example.c (~/) - gedit
Open  Save

#include <stdio.h>
#include <openssl/bn.h>
#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 ()
{
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *a = BN_new();
        BIGNUM *b = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *res = BN_new();
        // Initialize a, b, n
        BN_hex2bn(&a, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&b, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&n, "0D88C3");
        // res = a * b
        BN_mul(res, a, b, ctx);
        printBN("a * b = ", res);
        // res = a^b mod n
        BN_mod_exp(res, a, b, n, ctx);
        printBN("a^c mod n = ", res);
        BN_mod_inverse(res, a, b, ctx);
        printBN("private key:", res);
        return 0;
}
```

```
Terminal
[05/13/20]seed@VM:~$ gcc example.c -lcrypto
example.c: In function 'main':
example.c:31:2: error: expected ';' before 'printBN'
  printBN("private key:", res)
  ^
[05/13/20]seed@VM:~$ gcc example.c -lcrypto
[05/13/20]seed@VM:~$ ./a.out
a * b =  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
a^c mod n =  0AA578
private key: CC9805F13A20303FECD2CCBAEBD53F7D
[05/13/20]seed@VM:~$
```

## Task 2.



```
example.c (~/) - gedit
Open  Save

#include <stdio.h>
#include <openssl/bn.h>
#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 ()
{
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *a = BN_new();
        BIGNUM *b = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *res = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *m = BN_new();

        // Initialize a, b, n
        BN_hex2bn(&a, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&b, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&e, "010001");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&m, "4120746f70207365637265742");
        // res = a * b
        BN_mul(res, a, b, ctx);
        printBN("a * b = ", res);
        // res = a^b mod n
        BN_mod_exp(res, a, b, n, ctx);
        printBN("a^c mod n = ", res);
        BN_mod_inverse(res, a, b, ctx);
        printBN("private key:", res);
        BN_mod_exp(m, e, b, n, ctx);
        printBN("encryption:", m);
        return 0;
}
```

```
Terminal
[05/13/20]seed@VM:~$ gcc example.c -lcrypto
example.c: In function 'main':
example.c:31:2: error: expected ';' before 'printBN'
  printBN("private key:", res)
  ^
[05/13/20]seed@VM:~$ gcc example.c -lcrypto
[05/13/20]seed@VM:~$ ./a.out
a * b =  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
a^c mod n =  0AA578
private key: CC9805F13A20303FECD2CCBAEBD53F7D
[05/13/20]seed@VM:~$ python -c 'print("A top secret!".encode("hex"))'
4120746f70207365637265742
[05/13/20]seed@VM:~$ gcc example.c -lcrypto
[05/13/20]seed@VM:~$ ./a.out
a * b =  E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
a^c mod n =  714B480A2556B52F1B959CC5244342251F743AFB47FD20397F79D1DA8B7ECC38
private key: CC9805F13A20303FECD2CCBAEBD53F7D
encryption: CBF6CBE7019D3A4ACA419650FE76D0C0CA48B056B382E3A2AD08E6EF249E73EA
[05/13/20]seed@VM:~$
```

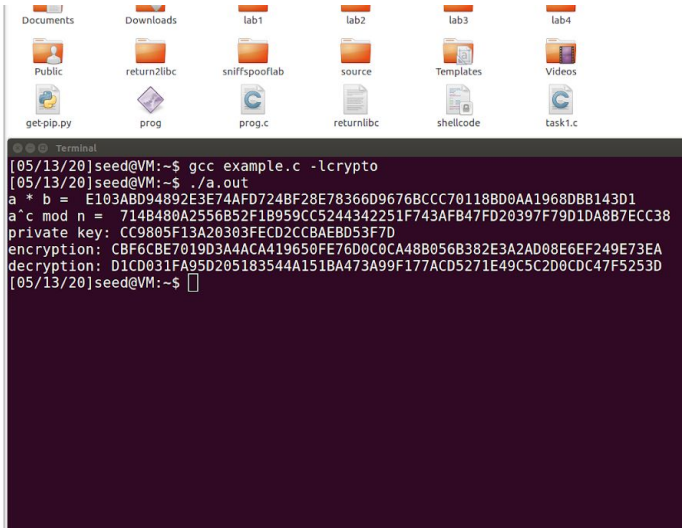
```
02073656372657421".decode("hex"))'
```

are the same as the ones used in Task 2. Please generate a signature
y sign this message, instead of signing its hash value):

Task 3.

```c
#include <stdio.h>
#include <openssl/bn.h>
#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 ()
{
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *a = BN_new();
        BIGNUM *b = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *res = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *m = BN_new();
        BIGNUM *new_m = BN_new();

        // Initialize a, b, n
        BN_hex2bn(&a, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&b, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&e, "010001");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&m, "4120746f702073656372657421");
        // res = a * b
        BN_mul(res, a, b, ctx);
        printBN("a * b = ", res);
        // res = a^b mod n
        BN_mod_exp(res, a, b, n, ctx);
        printBN("a^c mod n = ", res);
        BN_mod_inverse(res, a, b, ctx);
        printBN("private key:", res);
        BN_mod_exp(n, e, b, n, ctx);
        printBN("encryption:", n);
        BN_mod_exp(new_m, e, d, n, ctx);
        printBN("decryption:", new_m);
        return 0;
}
```




```
[05/13/20]seed@VM:~$ python -c 'print("4120746f702073656372657421".decode("hex")
)'
A top secret!
[05/13/20]seed@VM:~$
```

**Questions:**

- **Explain how a digital signature is created by describing what is happening in figure 23.4 (page 547) in your book.**

Alice and Bob are sending a digital document to each other, Alice signs her document using her private key and message to generate a digital signature. Bob then uses Alice's public key to verify Alice's signature on the document. Once verified Bob can see the document Alice sent him.

- **Explain how the chip reader in a credit card works by describing what is happening in figures 23.8 and 23.9 (pages 558 and 559) of your book**

Figure 23.8

When the card is inserted into the reader, the reader reads the public key certificate from that card. When it reads it uses the issuers public key to verify the certificate. Then the terminal sends a challenge to the card. The card passes the challenge by using its private key and creates a signature that is then read by the reader. The reader must verify this signature using the card's public key.

Figure 23.9

The vendor sends transaction data to the card's chip then the card uses the private key to sign the transaction data creating signatures on each transaction. The vendor must then verify the signature of the transaction and send the transaction data and signature to the card issuer.