Lab Environment

Seed Ubuntu V 16.04

Libraries required: Openssl- BIGNUM library

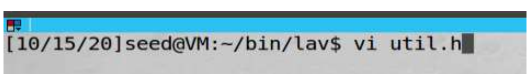
RSA Algorithm

The RSA cryptosystem is the most widely-used public key cryptography algorithm in the world. It can be used to encrypt a message without the need to exchange a secret key separately. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers. Thus we use Openssl library with BIGNUM for performing the operations on these huge numbers.

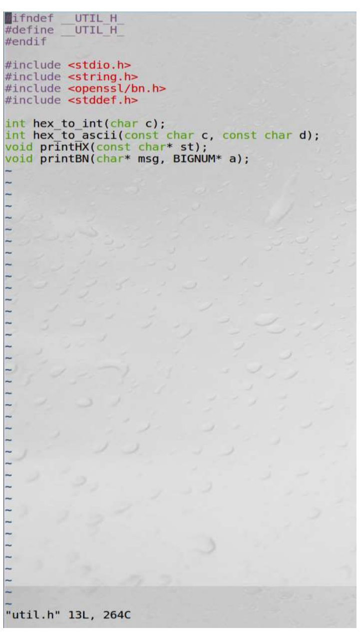
Party A can send an encrypted message to party B without any prior exchange of secret keys. A just uses B's public key to encrypt the message and B decrypts it using the private key, which only he knows. RSA can also be used to sign a message, so A can sign a message using their private key and B can verify it using A's public key.

Code files:

<u>Util.h</u>



```
[10/15/20]seed@VM:~/bin/lav$ vi util.h
```



```
#ifndef __UTIL_H_
#define __UTIL_H_
#endif

#include <stdio.h>
#include <string.h>
#include <openssl/bn.h>
#include <stddef.h>

int hex_to_int(char c);
int hex_to_ascii(const char c, const char d);
void printHX(const char* st);
void printBN(char* msg, BIGNUM* a);
"util.h" 13L, 264C
```

It defines the functions to convert hexadecimal numbers to ASCII string, hexadecimal numbers to integers, to print hexadecimal digits and to print BIGNUM .

Util.c

```
[10/15/20]seed@VM:~/bin/lav$ vi util.h
[10/15/20]seed@VM:~/bin/lav$ vi util.c
```

```
#include "util.h"

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

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

void printHX(const char* str)
{
        int length = strlen(str);
        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, str[i]));
                else
                    buf = str[i];
        }
        printf("\n");
}

void printBN(char* msg, BIGNUM * a)
{
    char * num_str = BN_bn2hex(a);
    printf("%s 0x%s\n", msg, num_str);
    OPENSSL_free(num_str);
}
~
~
~
~
~
~
~
"util.c" 44L, 808C
```

This defines the functions hex_to_int: converts hexadecimal string to decimal integer, hex_to_ascii: converts the hexadecimal string to ascii.

PrintHX: First , we check the length of the hexadecimal string, if it has even number of characters in the string then it is valid else it is not valid.

Next, we run a loop and pass each character of the hexadecimal string to the function hex_to_ascii which calls the functions hex_to-int.

hex_to_int: Here the characters are checked if they are in small letters they are converted to Capital letters by doing `c-32` . It checks if the final result is < 9 (Single Integer value lies from 0-9) and thus returns the integer.

Hex_to_ascii: Calls the hex-to_int function and then performs operations to find the ascii value of a given hexadecimal string.

RSA.h

```
[10/15/20]seed@VM:~/bin/lav$ vi rsa.h
```

It defines the three functions rsa_priv_key, rsa_encrypt, rsa_decrypt

```
#ifndef __RSA_H_
#define __RSA_H_
#endif

#include <openssl/bn.h>
#include "util.h"

BIGNUM* rsa_priv_key(BIGNUM* p, BIGNUM* q, BIGNUM* e);
BIGNUM* rsa_encrypt(BIGNUM* message, BIGNUM* mod, BIGNUM* pub_key);
BIGNUM* rsa_decrypt(BIGNUM* encrypted_message, BIGNUM* priv_key, BIGNUM* pub_key);

~
"rsa.h" 11L, 292C
```

RSA.c

```
[10/15/20]seed@VM:~/bin/lav$ vi rsa.h
[10/15/20]seed@VM:~/bin/lav$ vi rsa.c
```

```
#include "rsa.h"

BIGNUM* rsa_priv_key(BIGNUM* p, BIGNUM* q, BIGNUM* e)
{
	/*
		given two large prime numbers, compute a private key
		using the modulo inverse
	*/
	BN_CTX *ctx = BN_CTX_new();
	BIGNUM* p_minus_1 = BN_new();
	BIGNUM* q_minus_1 = BN_new();
	BIGNUM* one = BN_new();
	BIGNUM* t = BN_new();

	BN_dec2bn(&one, "1");
	BN_sub(p_minus_1, p, one);
	BN_sub(q_minus_1, q, one);
	BN_mul(t, p_minus_1, q_minus_1, ctx);

	BIGNUM* result = BN_new();
	BN_mod_inverse(result, e, t, ctx);
	BN_CTX_free(ctx);
	return result;
}

BIGNUM* rsa_encrypt(BIGNUM* message, BIGNUM* mod, BIGNUM* pub_key)
{
	/*
		compute the RSA cipher for the message
		the ciphertext is: message^mod (modulo 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)
{
	/*
		compute the original message: (message ^ mod) ^ 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;
}
"rsa.c" 51L, 1108C
```

`BN_CTX *ctx = BN_CTX_new()` is a structure that hold the BIGNUM temporary variables.

`BIGNUM *a = BN_new()` is initializing a BIGNUM variable.

`BN_dec2bn` assigns a value from decimal number string

`BN_hex2bn` assigns a value from a hex number string

`BN_mod_inverse(b, a, n, ctx)` : given `a * b mod n = 1`, we need to find `b`. The value b is called the inverse of a, with respect to modular n.

Rsa_priv_key returns the private key when two large prime numbers are provided by using modulo inverse.

Rsa-encrypt computes the RSA ciphertext for the message where ciphertext is message ^ modulus

Rsa-decrypt computes the original message given the ciphertext by (message ^ mod) ^ pubkey

1.Choose two prime numbers p and q.
2. Compute n = p*q.
3. Calculate n = (p-1) * (q-1).
4. Choose an integer e such that 1 < e < n and gcd(e, n) = 1;
5. Calculate d as d ≡ e−1 (mod n), d is the modular multiplicative inverse of e modulo n.
6. For encryption, c= m*e mod n, where m = original message and c is ciphertext
7. For decryption, m = c* d mod n, where m= original message and c is ciphertext

Lab2.c

```
[10/15/20]seed@VM:~/bin/lav$ gcc lab2.c -lssl -lcrypto -L/usr/local/ssl -o lab2
[10/15/20]seed@VM:~/bin/lav$ ./lab2
```

3.1 Task 1: Deriving the Private Key

Given: p, q, and e are three prime numbers where n = p*q. We will use (e, n) as the public key.

Calculate: d is the private key that has to be calculated.

Values:

```
p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3
```

Code:

```
#include <stdlib.h>
#include "util.c"
#include "rsa.c"

int main ()
{
        printf("\n");
        printf("Task 1 - Deriving a private key \n");
        /* p and q are large prime numbers*/

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

        // Assigning a value to p
        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");

        // Assigning a value to q
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

        // (e,n) is the public key
        BN_hex2bn(&e, "0D88C3");

        // d is the private key
        BIGNUM* d = rsa_priv_key(p, q, e);
        printBN("The  Private key derived from task 1 is :",d);

        BIGNUM* enc = BN_new();
        BIGNUM* dec = BN_new();

        printf("------------------------------------------------------------\n");
```

p, q, e are three large prime numbers.
We assign their pre-decided values.
Next we pass them into the function rsa_priv_key to calculate the private key, d.

Output:

```
Task 1 - Deriving a private key
The  Private key derived from task 1 is : 0x3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
------------------------------------------------------------
```

3.2 Task 2: Encrypting a Message

Given Hex Values:

```
n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
e = 010001
M = A top secret!
d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D
```

Code:

```
printf("-----------------------------------------------------------\n");
printf("Task 2 - Encrypting a message\n");

// Assigning the private key
BIGNUM* priv_key = BN_new();
BN_hex2bn(&priv_key, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

// Assigning  the public key
BIGNUM* pub_key = BN_new();
BN_hex2bn(&pub_key, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
printBN("The public key is: ", pub_key);

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

// The message to be encrypted is 'A top secret!'
// In RSA, first convert this message into hex
// Then convert the hex into a BIGNUM for further calculations
BIGNUM* message = BN_new();
BN_hex2bn(&message, "4120746f702073656372657421");
enc = rsa_encrypt(message, mod, pub_key);
printBN("The encrypted message for task 2 is: ", enc);
dec = rsa_decrypt(enc, priv_key, pub_key);
printf("The decrypted message for task 2 is: ");
printHX(BN_bn2hex(dec));
printf("-----------------------------------------------------------\n");
```

We are given the private key , public key, e and the message to be encrypted.

`python -c 'print("A top secret!".encode("hex"))'`: We first convert the message to hexadecimal string, then convert the hexadecimal string to BIGNUM.

Now the message is encrypted using rsa_encrypt function with public key.

Later on , it is decrypted using the private key to show that the decrypted message is correct.

Output:

```
-----------------------------------------------------------
Task 2 - Encrypting a message
The public key is:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
The encrypted message for task 2 is:  0x6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
The decrypted message for task 2 is: A top secret!
-----------------------------------------------------------
```

3.3 Task 3: Decrypting a Message

Given Cipher text,

```
C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F
```

Code:

```
printf("Task 3 - Decrypting a message\n");

// Now decrypt the given ciphertext which is in hexadecimal format
// So convert to a BIGNUM for further calculations.
BIGNUM* task3_enc = BN_new();
printf(" The given cipher text is:8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F\n");
BN_hex2bn(&task3_enc, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

// We already have the public and private keys.
// We can decrypt using our rsa_decrypt function.
dec = rsa_decrypt(task3_enc, priv_key, pub_key);
printf("The decrypted message for task 3 is: ");
printHX(BN_bn2hex(dec));
printf("\n");

printf("-----------------------------------------------------------\n");
```

A ciphertext is given which has to be decrypted.

First the hexadecimal string is converted to Bignum and then rsa_decrypt function uses private and public key to decrypt the given ciphertext.

Output:

```
-----------------------------------------------------------------
Task 3 - Decrypting a message
 The given cipher text is:8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F
The decrypted message for task 3 is: Password is dees

-----------------------------------------------------------------
```

3.4 Task 4: Signing a Message

Code:

```
printf("-----------------------------------------------------------------\n");

printf("Task 4 - Signing a message\n");

// This task deals with signing a message by generating digital signature.
// The message is "I owe you $2000".
// This needs to be converted to hex.
// This command is used to find the hex value:python -c 'print("I owe you $2000".encode("hex"))'
// Once the  hex is obtained, convert hex to a BIGNUM for the computations.
printf("The first message being signed is:\n I owe you $2000.\n");
BIGNUM* BN_task4 = BN_new();
printf("The hex value of above message is:49206f776520796f752024323030302e\n");
BN_hex2bn(&BN_task4,"49206f776520796f752024323030302e");

// Use the already known  private key to encrypt
enc = rsa_encrypt(BN_task4, priv_key, pub_key);
printBN("The signature for first message is:\n ", enc);

// To verify if the process was correct, decryption is done using public key and cipher text
dec = rsa_decrypt(enc, mod, pub_key);
printf("The message after decryption  is:\n ");
printHX(BN_bn2hex(dec));

printf("\n\n");
// Change the message to check how the signature gets affected.
printf("The new message being signed is:\n I owe you $3000.\n");
BIGNUM* BN_task4b = BN_new();
printf("The hex value for above message is:49206f776520796f752024333030302e\n");
BN_hex2bn(&BN_task4b,"49206f776520796f752024333030302e");

enc = rsa_encrypt(BN_task4b, priv_key, pub_key);
printBN("The new signature for second message is:\n ",enc);
// To verify if the process was correct, decryption is done using public key and cipher text
dec = rsa_decrypt(enc,mod, pub_key);
printf("The message after decryption is:\n");
printHX(BN_bn2hex(dec));
printf("\n");


printf("-----------------------------------------------------------------\n");
```

A message is given whose signature must be generated. We also have to check if the message is changed, does the signature change or remains same. First generate the hex value of the given message using the command: `python -c 'print("I owe you $200.encode("hex"))'`. Private key is used to encrypt the message and get the signature. To verify if the process was correct, we decrypt using public key and check if the original message is received.

Now change the message to below text.

Generating a signature for the message:
`M = I owe you $2000.`

Output:

```
Task 4 - Signing a message
The first message being signed is:
 I owe you $2000.
The hex value of above message is:49206f776520796f752024323030302e
The signature for first message is:
  0x55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
The message after decryption  is:
 I owe you $2000.
```

Made a slight difference to the message M, by changing $2000 to $3000, and signed the modified message this time.

`M = I owe you $3000.`

Now we can see that the signature is Different. Thus we can conclude that even a slight change in the message will generate a different signature. We also decrypt the ciphertext to show the original message.

Output:

```
The new message being signed is:
 I owe you $3000.
The hex value for above message is:49206f776520796f752024333030302e
The new signature for second message is:
  0xBCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
The message after decryption is:
I owe you $3000.
```

3.5 Task 5: Verifying a Signature

Bob receives a message M = `"Launch a missile."` from Alice, with her signature `S`. We know that Alice's public key is `(e, n)`. Verifying whether the signature is Alice's or not?

```
M = Launch a missle.
S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
e = 010001
n = AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
```

Code:

```
   printf("------------------------------------------------------------------------\n");

   printf("Task 5 - Verifying a signature\n");

   // In this task, signature verification is performed,where the signature is represented by S.
   // The given message is "Launch a missile".It's hex value is taken.
   // Public key is used to decrypt a message which has been encrypted  with the private key
   // And then compare the message with  decrypted result.
   BIGNUM* BN_task5 = BN_new();
   BIGNUM* S = BN_new();
   BN_hex2bn(&BN_task5, "4c61756e63682061206d697373696c652e");
   BN_hex2bn(&pub_key, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
   BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");

   // Decrypt the message with the public key.
   dec = rsa_decrypt(S, mod, pub_key);
   printf("The message for task5 with orginal signature is:\n ");

   printHX(BN_bn2hex(dec));
   printf("\n");

   // Now, change the last two digits of the signature, and verify again.
   BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

   // Decrypting a corrupted message with the public key.
   dec = rsa_decrypt(S, mod, pub_key);
   printf("\nThe message for task5 with modified signature is: \n");
   // Thus,the output will be different from the orginal version.
   printHX(BN_bn2hex(dec));
   printf("\n");
   printf("------------------------------------------------------------------------");
   printf("\n");
```

Output :

```
Task 5 - Verifying a signature
The message for task5 with orginal signature is:
 Launch a missile.
```

Now the signature is corrupted, such that the last byte of the signature changes from `2F` to `3F`. Now signature verification is done using public key which shows that the message is completely different.

Output:

```
The message for task5 with modified signature is:
▯▯,▯▯▯c▯▯▯rm=f▯:N▯▯▯▯▯▯
```

3.6 Task 6: Manually Verifying an X.509 Certificate

In this task, we will manually verify an X.509 certificate using our program. An X.509 contains data about a public key and an issuer's signature on the data. We will download a real X.509 certificate from a web server, get its issuer's public key, and then use this public key to verify the signature on the certificate.

Code:

```
        printf("Task 6 - Manually Verifying an X.509 Certificate\n");

        // We extracted the public key and modulus from the certificate at example.com
        // The command was: openssl x509 -in c1.pem -text -noout

        // Assign the public key
        BIGNUM* task6_pub_key = BN_new();
        BN_hex2bn(&task6_pub_key,"00D3487CBEF305865D5BD52F854E4BE086AD15AC61CF5BAF3E6A0A47FB9A7691600B8A6BCDCFDC577E60980BE454D956ED21CC02B65A815F976AEE022F2327B86DD4B0
E70602780B1F5CA99936FE8BAC1B05FA57CD81104067D6308B5835D49661BED08C7A979F1AF922E6142FA9C6E8011FABF8260FAC8E4D2C32391D819B8D1C65B21CDB61A8892F60E7EBC24A18C46F2AE9109209ED
17D1002BE67DEF0489144E33A1B20F97879FB3A0CD2FBC2CECB88368313D1FD54A90101908B195D62976 51F93676D0B7097A384AD76F8CBF137C39EDBAAE90FC95F77B7809365E74931E25F0FFD4ADAE686BC6FF
0FD535F1556E4849F8F8B8EF88F8F15E1177AADF02B3");
        printBN("the public key is: ", task6_pub_key);

        // Assign the modulus
        BIGNUM* task6_mod = BN_new();
        BN_hex2bn(&task6_mod, "10001");

        // We extracted the signature (RSA-signed sha256) from the certificate:
        // openssl x509 -in c0.pem -text -noout

        // Assign the signature to a pointer to a BIGNUM
        BIGNUM* BN_task6 = BN_new();
        BN_hex2bn(&BN_task6, "4668db7e7a0ee058744df56f807ca947611c86f882b8de3618cc661b39b39f98e962afad1dbb9b0fd3b36b04d98d36b5b82422d26ced79fe0e0391948e243ecaf1e8a3b98d
3385c1768208032aaeb0be0e90720cb81b58a73c033bd1f32d6f21ef427e1432c812c6cba817be0f8cb80e1addebfaba739caa02e8998a8219a6c22551ee50b790e43478801dd6fe9adce1bee6b34ce406c1b744
9ef9ad83f2408532a4b9b3819efd89129dce2003543624b7e81d78b6ae38ebeb4cf9d0d5b61b2982b9e9229a3274fc45edbfd200d6e46d9e6e7a93d2ec010a99669a501c930af85ee2eca7272b4dfe1698dfcd21
378c7013f09931265dd1a9560403829e74061e");

        // We generated the hash of the certificate body like so:
        // First, extract the body of the certificate
        // openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
        // Then, compute the hash:
        // sha256sum c0_body.bin
        // This hash will be used for comparison for when we decrypt the signature.

        // Now we decrypt the signature using the public key and modulus given from the certificate.
        // If the signature is valid, it should match our hash of the certificate body we computed earlier.
        BIGNUM* task6_dec = rsa_decrypt(BN_task6, task6_mod, task6_pub_key);

        // Print the decrypted hash. This is a non-masked value.
        // Follow up - we want to bitmask & this value with 256
        // In other words, the first 32 bytes of this value should
        // match the hash generated from the body of the certificate.
        printBN("Hash from the server: ", task6_dec);

        printf("\n");

        printf("Pre-computed hash: ");
        printf("662052b1de8743a7e8801af9e476c0eae5c2eacb4493a3fde4efa76386e544eb");
        printf("\n");
        printf("------------------------------------------------------------------\n"); }
                                                                                                                    190,1         99%
```

Step 1: Download a certificate from a real web server.

I have used the website - Amazon.com and obtained it's certificates.

```
[10/15/20]seed@VM:~/bin/lav$ openssl s_client -connect www.amazon.com:443 -showcerts
CONNECTED(00000003)
depth=3 C = US, O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "(c) 2006 VeriSign, Inc. - For authorized use only", CN = VeriSign Class 3 Public Primary Cert
ication Authority - G5
verify return:1
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root G2
verify return:1
depth=1 C = US, O = DigiCert Inc, CN = DigiCert Global CA G2
verify return:1
depth=0 C = US, ST = Washington, L = Seattle, O = "Amazon.com, Inc.", CN = www.amazon.com
verify return:1
```

These are the two certificates:

```
Certificate chain
 0 s:/C=US/ST=Washington/L=Seattle/O=Amazon.com, Inc./CN=www.amazon.com
   i:/C=US/O=DigiCert Inc/CN=DigiCert Global CA G2
-----BEGIN CERTIFICATE-----
MIIHxjCCBq6gAwIBAgIQBG3NDm3AOeSmxTaWm6dsmjANBgkqhkiG9w0BAQsFADBE
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMR4wHAYDVQQDExVE
aWdpQ2VydCBHbG9iYWwgQ0EgRzIwHhcNMjAwMTIzMDAwMDAwWhcNMjAxMjMxMTIw
MDAwWjBoMQswCQYDVQQGEwJVUzETMBEGA1UECBMKV2FzaGluZ3RvbjEQMA4GA1UE
BxMHU2VhdHRsZTEZMBcGA1UEChMQQW1hem9uLmNvbSwgSW5jLjEXMBUGA1UEAxMO
d3d3LmFtYXpvbi5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCl
iCwYkF1H8Zvg1+sqst3AApBs+JOfS38YwnYia0t+nhaHr8mk+ImZ9SX+5okYF4dW
jhz1mhMzVYvWbJLjzkkNrZ6EoSK40ibYT/hQHaSvhG+pWaRrOWI0RZeMiBr22j+0
ZIwmYuxXkUjDsxzY3UUcuv7sbQO4QsQjuTgdsHEHBnLyNdFGSqyZI2zMR6Ezqzxz
I+3ERe Xdsqm6HPVvz8cgRpzZIXhrpYRrv02bTlWu+CXxaHdloiVgUgp2pqrbXsV8
PSpmAi0rQAyckHCntSN4WWVkBHcK675MVut/sK3SM5D039vQKgNIicoglxD3paw1
hHKxTbYgwgKahKtzjrB3AgMBAAGjggSOMIIEijAfBgNVHSMEGDAWgBQkbist0GqS
UVElaQGqmkemiedAIDAdBgNVHQ4EFgQUmI69RYRC6FU5GGpiM3sBBlyIKXIwggHG
BgNVHREEggG9MIIBuYIKYW1hem9uLmNvbYIIYW16bi5jb22CEWJ1eWJveC5hbWF6
b24uY29tghRjb3Jwb3JhdGUuYW1hem9uLmNvbYIPaG9tZS5hbWF6b24uY29tghFp
cGhvbmUuYW1hem9uLmNvbYIWa29ucmFkLXRlc3QuYW1hem9uLmNvbYISbXAzcmVj
cy5hbWF6b24uY29tgiVwLW50LXd3dy1hbWF6b24tY29tLWthbGlhcy5hbWF6b24u
Y29tgiVwLXkzLXd3dy1hbWF6b24tY29tLWthbGlhcy5hbWF6b24uY29tgiVwLXlv
LXd3dy1hbWF6b24tY29tLWthbGlhcy5hbWF6b24uY29tghFzdGF0aWMuYW1hem9u
LmNvbYITdGVzdC13d3cuYW1hem9uLmNvbYIRdWVkYXRhLmFtYXpvbi5jb22CDXVz
LmFtYXpvbi5jb22CDnd3dy5hbWF6b24uY29tggx3d3cuYW16bi5jb22CEnd3dy5j
ZG4uYW1hem9uLmNvbYIQd3d3Lm0uYW1hem9uLmNvbYIWeWVsbG93cGFnZXMuYW1h
em9uLmNvbYINeXAuYW1hem9uLmNvbTAOBgNVHQ8BAf8EBAMCBaAwHQYDVR0lBBYw
FAYIKwYBBQUHAwEGCCsGAQUFBwMCMHcGA1UdHwRwMG4wNaAzoDGGL2h0dHA6Ly9j
cmwzLmRpZ2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbENBRzIuY3JsMDWgM6Axhi9o
dHRwOi8vY3JsNC5kaWdpY2VydC5jb20vRGlnaUNlcnRHbG9iYWxDQUcyLmNybDBM
BgNVHSAERTBDMDcGCWCGSAGG/WwBATAqMCgGCCsGAQUFBwIBFhxodHRwczovL3d3
dy5kaWdpY2VydC5jb20vQ1BTMAgGBmeBDAECAjB0BggrBgEFBQcBAQRoMGYwJAYI
KwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTA+BggrBgEFBQcwAoYy
aHR0cDovL2NhY2VydHMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0R2xvYmFsQ0FHMi5j
cnQwCQYDVR0TBAIwADCCAQUGCisGAQQB1nkCBAIEgfYEgfMA8QB3ALIeBcyLos2K
IE6HZvkruYolIGdr2vpw57JJUy3vi5BeAAABb9SYN2AAAAQDAEgwRgIhANv+9u4w
rDU63HYVCsbbcsZb4Jq5R9oRdy1IBSv28JQaAiEAu40rmDhXOA3pNxjFTdIQEyD4
Ytx4X4IdkPrilcEE8L0AdgDwlaRZ8gDRgkAQLS+TiI6tS/4dR+OZ4dA0prCoqo6y
cwAAAW/UmDeZAAAEAwBHMEUCIFJD9ek/2K7+4J6PAIir9TX40g+sODQkiR1HfR4j
JzZxAiEA342uQeitSnONAEdzRrOc54kQcERM5evX2TPqi0AeUZgwDQYJKoZIhvcN
AQELBQADggEBAEZo2356DuBYdE31b4B8qUdhHIb4grjeNhjMZhs5s5+Y6WKvrR27
mw/Ts2sE2Y02tbgkItJs7Xn+DgORlI4kPsrx6KO5jTOFwXaCCAMqrrC+DpByDLgb
WKc8AzvR8y1vIe9CfhQyyBLGy6gXvg+MuA4a3ev6unOcqgLomYqCGabCJVHuULeQ
5DR4gB3W/prc4b7ms0zkBsG3RJ75rYPyQIUypLmzgZ79iRKdziADVDYkt+gdeLau
QOvrTPnQ1bYbKYK56SKaMnT8Re2/0gDW5G2ebnqT0uwBCplmmlAckwr4XuLspycr
Tf4WmN/NITeMcBPwmTEmXdGpVgQDgp50Bh4=
-----END CERTIFICATE-----
```

The second certificate belongs to an intermediate CA. In this task, we will use CA's certificate to verify a server certificate.

```
 1 s:/C=US/O=DigiCert Inc/CN=DigiCert Global CA G2
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Global Root G2
-----BEGIN CERTIFICATE-----
MIIEizCCA3OgAwIBAgIQDI7gyQ1qiRWIBAYe4kH5rzANBgkqhkiG9w0BAQsFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBH
MjAeFw0xMzA4MDExMjAwMDBaFw0yODA4MDExMjAwMDBaMEQxCzAJBgNVBAYTAlVT
MRUwEwYDVQQKEwxEaWdpQ2VydCBJbmMxHjAcBgNVBAMTFURpZ2lDZXJ0IEdsb2Jh
bCBDQSBHMjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANNIfL7zBYZd
W9UvhU5L4IatFaxhz1uvPmoKR/uadpFgC4przc/cV35gmAvkVNlW7SHMArZagV+X
au4CLyMnuG3UsOcGAngLH1ypmTb+u6wbBfpXzYEQQGfWMItYNdSWYb7QjHqXnxr5
IuYUL6nG6AEfq/gmD6yOTSwyOR2Bm40cZbIc22GoiS9g5+vCShjEbyrpEJIJ7RfR
ACvmfe8EiRROM6GyD5eHn7OgzS+8LOy4g2gxPR/VSpAQGQuBldYpdlH5NnbQtwl6
OErXb4y/E3w57bqukPyV93t4CTZedJMeJfD/1K2uaGvG/w/VNfFVbkhJ+Pi474j4
8V4Rd6rfArMCAwEAAaOCAVowggFWMBIGA1UdEwEB/wQIMAYBAf8CAQAwDgYDVR0P
AQH/BAQDAgGGMDQGCCsGAQUFBwEBBCgwJjAkBggrBgEFBQcwAYYaHR0cDovL29j
c3AuZGlnaWNlcnQuY29tMHsGA1UdHwR0MHIwN6A1oDOGMWh0dHA6Ly9jcmw0LmRp
Z2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwN6A1oDOGMWh0dHA6
Ly9jcmwzLmRpZ2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwPQYD
VR0gBDYwNDAyBgRVHSAAMCowKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3LmRpZ2lj
ZXJ0LmNvbS9DUFMwHQYDVR0OBBYEFCRuKy3QapJRUSVpAaqaR6aJ50AgMB8GA1Ud
IwQYMBaAFE4iVCAYlebjbuYP+vq5Eu0GF485MA0GCSqGSIb3DQEBCwUAA4IBAQAL
OYSR+ZfrqoGvhOlaOJL84mxZvzbIRacxAxHhBsCsMsdaVSnaT0AC9aHesO3ewPj2
dZ12uYf+QYB6z13jAMZbAuabeGLJ3LhimnftiQjXS8X9Q9ViIyfEBFltcT8jW+rZ
8uckJ2/0lYDblizkVIvP6hnZf1WZUXoOLRg9eFhSvGNoVwvdRLNXSmDmyHBwW4co
atc7TlJFGa8kBpJIERqLrqwYElesA8u49L3KJg6nwd3jM+/AVTANlVlOnAM2BvjA
jxSZnE0qnsHhfTuvcqdFuhOWKU4Z0BqYBvQ3lBetoxi6PrABDJXWKTUgNX31EGDk
92hiHuwZ4STyhxGs6QiA
-----END CERTIFICATE-----
```

There is also an additional certificate for this site.

```
 2 s:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Global Root G2
   i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority -
G5
-----BEGIN CERTIFICATE-----
MIIE3zCCA8egAwIBAgIQYxgNOPuAl3ip0DWjFhj4QDANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMTcxMTA2MDAwMDAwWhcNMjIxMTA1MjM1OTU5WjBhMQsw
CQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3d3cu
ZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBHMjCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALs3zTTce2vJsmiQrUp1/0a6
IQoIjfUZVMn7iNvzrvI6iZE8euarBhprz6wt6F4JJES6Ypp+1qOofuBUdSAFrFC3
nGMabDDc2h8Zsdce3v3X4MuUgzeu7B9DTt17LNK9LqUv5Km4rTrUmaS2JembawBg
kmD/TyFJGPdnkKthBpyP8rrptOmSMmu181foXRvNjB2rlQSVSfM1LZbjSW3dd+P7
SUu0rFUHqY+Vs7Qju0xtRfD2qbKVMLT9TFWMJ0pXFHyCnc1zktMWSgYMjFDRjx4J
vheh5iHK/YPlELyDpQrEZyj2cxQUPUZ2w4cUiSE0Ta8PRQymSaG6u5zFsTODKYUC
AwEAAaOCAScwggEjMB0GA1UdDgQWBBROIlQgGJXm427mD/r6uRLtBhePOTAPBgNV
HRMBAf8EBTADAQH/MF8GA1UdIARYMFYwVAYEVR0gADBMMCMGCCsGAQUFBwIBFhdo
dHRwczovL2Quc3ltY2IuY29tL2NwczAlBggrBgEFBQcCAjAZDBdodHRwczovL2Qu
c3ltY2IuY29tL3JwYIAvBgNVHR8EKDAmMCSgIqAghh5odHRwOi8vcy5zeW1jYi5j
b20vcGNhMy1nN55jcmwwDgYDVR0PAQH/BAQDAgGGMC4GCCsGAQUFBwEBBCIwIDAe
BggrBgEFBQcwAYYSaHR0cDovL3Muc3ltY2QuY29tMB8GA1UdIwQYMBaAFH/TZafC
3ey78DAJ80M5+gKvMzEzMA0GCSqGSIb3DQEBCwUAA4IBAQBQ3dNWKSUBip6n5X1N
ua8bjKLSJzXlnescavPECMpFBlIKH2mc6mL2Xr/wkSIBDrsqAO3sBcmoJN+n8V3
0O5JelrtEAFYSyRDXfu78ZlHn6kvV5/jPUFECEM/hdN0x8WdLpGjJMqfs0EG5qHj
+UaxpucWD445wea4zlK7hUR+MA8fq0Yd1HEKj4c8TcgaQIHMa4KHr448cQ69e3CP
ECRhRNg+RAKT2I7SlaVzLvaB/8yym2oMCEsoqiRT8dbXg35aKEYmmzn3O/mnB7bG
Ud/EUrkIf7FVamgYZd1fSzQeg1cHqf0ja6eHpvq2bTl+cWFHaq/84KlHe5Rh0Csm
pZzn
-----END CERTIFICATE-----
---
Server certificate
subject=/C=US/ST=Washington/L=Seattle/O=Amazon.com, Inc./CN=www.amazon.com
issuer=/C=US/O=DigiCert Inc/CN=DigiCert Global CA G2
---
No client certificate CA names sent
Peer signing digest: SHA256
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 5072 bytes and written 431 bytes
```

```
Server certificate
subject=/C=US/ST=Washington/L=Seattle/O=Amazon.com, Inc./CN=www.amazon.com
issuer=/C=US/O=DigiCert Inc/CN=DigiCert Global CA G2
---
No client certificate CA names sent
Peer signing digest: SHA256
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 5072 bytes and written 431 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: 6EFC7960F4F75361F85226FD028D16FFF2BA27D02FDBA752846AC917AECF915D
    Session-ID-ctx:
    Master-Key: 195CC6A73A0481FCDD6D3592472EAB6BEA3826789248EBE74FA79D36131962681AE3404329204E2899E2911AC0E06DE1
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 83100 (seconds)
    TLS session ticket:
    0000 - 00 00 3b d4 42 d5 a1 e2-ae 1e fb b6 04 c7 e5 1c   ..;.B...........
    0010 - 0b c6 1c 2f df 1a ff 03-fe 93 13 fb 25 b5 fa 33   .../........%..3
    0020 - 35 9b c7 e5 e4 91 97 58-2b 25 ff 07 52 89 84 fa   5......X+%..R...
    0030 - 42 ec 73 b0 82 18 35 3c-4e c3 4a ed 06 b1 3c 8b   B.s...5<N.J...<.
    0040 - 69 3a 5d b4 d5 93 2f 3d-98 76 80 d7 31 b8 29 79   i:].../=.v..1.)y
    0050 - 85 d6 27 09 43 1f eb f3-dd 55 11 a1 76 a1 60 67   ..'.C....U..v.`g
    0060 - 0a bd 7c 7a ab e1 a2 28-c0 b4 fd 92 b0 22 c6 d9   ..|z...(.....".. 
    0070 - 86 3d 54 97 57 1f 1b 74-49 e5 1f 88 4b f7 d7 af   .=T.W..tI...K...
    0080 - e4 fe 0f 6a 0e 93 d6 6c-55 66 f4 4b e6 c2 15 f9   ...j...lUf.K....
    0090 - 6b dd 17 3a f1 7a fd 2b-ac e2 64 cd f8 09 fe 68   k..:.z.+..d....h

    Start Time: 1602796361
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
closed
```

Now, Save the two certificates into c0.pem and c1.pem respectively

```
[10/15/20]seed@VM:~/bin/lav$ vi c0.pem
[10/15/20]seed@VM:~/bin/lav$ vi c1.pem
[10/15/20]seed@VM:~/bin/lav$
```

c0.pem:


```
-----BEGIN CERTIFICATE-----
MIIHxjCCBq6gAwIBAgIQBG3NDm3AOeSmxTaWm6dsmjANBgkqhkiG9w0BAQsFADBE
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMR4wHAYDVQQDExVE
aWdpQ2VydCBHbG9iYWwgQ0EgRzIwHhcNMjAwMTIzMDAwMDAwWhcNMjAxMjMxMTIw
MDAwWjBoMQswCQYDVQQGEwJVUzETMBEGA1UECBMKV2FzaGluZ3RvbjEQMA4GA1UE
BxMHU2VhdHRsZTEZMBcGA1UEChMQQW1hem9uLmNvbSwgSW5jLjEXMBUGA1UEAxMO
d3d3LmFtYXpvbi5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCl
iCwYkF1H8Zvg1+sqst3AApBs+JOfS38YwnYia0t+nhaHr8mk+ImZ9SX+5okYF4dW
jhz1mhMzVYvWbJLjzkkNrZ6EoSK40ibYT/hQHaSvhG+pWaRrOWI0RZeMiBr22j+0
ZIwmYuxXkUjDsxzY3UUcuv7sbQO4QsQjuTgdsHEHBnLyNdFGSqyZI2zMR6Ezqzxz
I+3EReXdsqm6HPVvz8cgRpzZIXhrpYRrv02bTlWu+CXxaHdloiVgUgp2pqrbXsV8
PSpmAi0rQAyckHCntSN4WWVkBHcK675MVut/sK3SM5D039vQKgNIicoglxD3paw1
hHKxTbYgwgKahKtzjrB3AgMBAAGjggSOMIIEijAfBgNVHSMEGDAWgBQkbist0GqS
UVElaQGqmkemiedAIDAdBgNVHQ4EFgQUmI69RYRC6FU5GGpiM3sBBlyIKXIwggHG
BgNVHREEggG9MIIBuYIKYW1hem9uLmNvbYIIYW16bi5jb22CEWJ1eWJveC5hbWF6
b24uY29tghRjb3Jwb3JhdGUuYW1hem9uLmNvbYIPaG9tZS5hbWF6b24uY29tghFp
cGhvbmUuYW1hem9uLmNvbYIWa29ucmFkLXRlc3QuYW1hem9uLmNvbYISbXAzcmVj
cy5hbWF6b24uY29tgiVwLW50LXd3dy1hbWF6b24tY29tLWthbGlhcy5hbWF6b24u
Y29tgiVwLXkzLXd3dy1hbWF6b24tY29tLWthbGlhcy5hbWF6b24uY29tgiVwLXlv
LXd3dy1hbWF6b24tY29tLWthbGlhcy5hbWF6b24uY29tghFzdGF0aWMuYW1hem9u
LmNvbYITdGVzdC13d3cuYW1hem9uLmNvbYIRdWVkYXRhLmFtYXpvbi5jb22CDXVz
LmFtYXpvbi5jb22CDnd3dy5hbWF6b24uY29tggx3d3cuYW16bi5jb22CEnd3dy5j
ZG4uYW1hem9uLmNvbYIQd3d3Lm0uYW1hem9uLmNvbYIWeWVsbG93cGFnZXMuYW1h
em9uLmNvbYINeXAuYW1hem9uLmNvbTAOBgNVHQ8BAf8EBAMCBaAwHQYDVR0lBBYw
FAYIKwYBBQUHAwEGCCsGAQUFBwMCMHcGA1UdHwRwMG4wNaAzoDGGL2h0dHA6Ly9j
cmwzLmRpZ2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbENBRzIuY3JsMDWgM6Axhi9o
dHRwOi8vY3JsNC5kaWdpY2VydC5jb20vRGlnaUNlcnRHbG9iYWxDQUcyLmNybDBM
BgNVHSAERTBDMDcGCWCGSAGG/WwBATAqMCgGCCsGAQUFBwIBFhxodHRwczovL3d3
dy5kaWdpY2VydC5jb20vQ1BTMAgGBmeBDAECAjB0BggrBgEFBQcBAQRoMGYwJAYI
KwYBBQUHMAGGGGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTA+BggrBgEFBQcwAoYy
aHR0cDovL2NhY2VydHMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0R2xvYmFsQ0FHMi5j
cnQwCQYDVR0TBAIwADCCAQUGCisGAQQB1nkCBAIEgfYEgfMA8QB3ALIeBcyLos2K
IE6HZvkruYolIGdr2vpw57JJUy3vi5BeAAABb9SYN2AAAAQDAEgwRgIhANv+9u4w
rDU63HYVCsbbcsZb4Jq5R9oRdy1IBSv28JQaAiEAu40rmDhXOA3pNxjFTdIQEyD4
Ytx4X4IdkPrilcEE8L0AdgDwlaRZ8gDRgkAQLS+TiI6tS/4dR+OZ4dA0prCoqo6y
cwAAAW/UmDeZAAAEAwBHMEUCIFJD9ek/2K7+4J6PAIir9TX40g+sODQkiR1HfR4j
JzZxAiEA342uQeitSnONAEdzRrOc54kQcERM5evX2TPqi0AeUZgwDQYJKoZIhvcN
AQELBQADggEBAEZo2356DuBYdE31b4B8qUdhHIb4grjeNhjMZhs5s5+Y6WKvrR27
mw/Ts2sE2Y02tbgkItJs7Xn+DgORlI4kPsrx6KO5jTOFwXaCCAMqrrC+DpByDLgb
WKc8AzvR8y1vIe9CfhQyyBLGy6gXvg+MuA4a3ev6unOcqgLomYqCGabCJVHuULeQ
5DR4gB3W/prc4b7ms0zkBsG3RJ75rYPyQIUypLmzgZ79iRKdziADVDYkt+gdeLau
OOvrTPnQ1bYbKYK56SKaMnT8Re2/0gDW5G2ebnqT0uwBCplmmlAckwr4XuLspycr
Tf4WmN/NITeMcBPwmTEmXdGpVgQDgp50Bh4=
-----END CERTIFICATE-----
```


```
~
~
~
~
~
~
"c0.pem" 45L, 2757C
```

c1.pem:


```
-----BEGIN CERTIFICATE-----
MIIEizCCA3OgAwIBAgIQDI7gyQ1qiRWIBAYe4kH5rzANBgkqhkiG9w0BAQsFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBH
MjAeFw0xMzA4MDExMjAwMDBaFw0yODA4MDExMjAwMDBaMEQxCzAJBgNVBAYTAlVT
MRUwEwYDVQQKEwxEaWdpQ2VydCBJbmMxHjAcBgNVBAMTFURpZ2lDZXJ0IEdsb2Jh
bCBDQSBHMjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANNIfL7zBYZd
W9UvhU5L4IatFaxhz1uvPmoKR/uadpFgC4przc/cV35gmAvkVNlW7SHMArZagV+X
au4CLyMnuG3UsOcGAngLH1ypmTb+u6wbBfpXzYEQQGfWMItYNdSWYb7QjHqXnxr5
IuYUL6nG6AEfq/gmD6yOTSwyOR2Bm40cZbIc22GoiS9g5+vCShjEbyrpEJIJ7RfR
ACvmfe8EiRROM6GyD5eHn7OgzS+8LOy4g2gxPR/VSpAQGQuBldYpdlH5NnbQtwl6
OErXb4y/E3w57bqukPyV93t4CTZedJMeJfD/1K2uaGvG/w/VNfFVbkhJ+Pi474j4
8V4Rd6rfArMCAwEAAaOCAVowggFWMBIGA1UdEwEB/wQIMAYBAf8CAQAwDgYDVR0P
AQH/BAQDAgGGMDQGCCsGAQUFBwEBBCgwJjAkBggrBgEFBQcwAYYYaHR0cDovL29j
c3AuZGlnaWNlcnQuY29tMHsGA1UdHwR0MHIwN6A1oDOGMWh0dHA6Ly9jcmw0LmRp
Z2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwN6A1oDOGMWh0dHA6
Ly9jcmwzLmRpZ2ljZXJ0LmNvbS9EaWdpQ2VydEdsb2JhbFJvb3RHMi5jcmwwPQYD
VR0gBDYwNDAyBgRVHSAAMCowKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3LmRpZ2lj
ZXJ0LmNvbS9DUFMwHQYDVR0OBBYEFCRuKy3QapJRUSVpAaqaR6aJ50AgMB8GA1Ud
IwQYMBaAFE4iVCAYlebjbuYP+vq5Eu0GF485MA0GCSqGSIb3DQEBCwUAA4IBAQAL
OYSR+ZfrqoGvhOlaOJL84mxZvzbIRacxAxHhBsCsMsdaVSnaT0AC9aHesO3ewPj2
dZ12uYf+QYB6z13jAMZbAuabeGLJ3LhimnftiQjXS8X9Q9ViIyfEBFltcT8jW+rZ
8uckJ2/0lYDblizkVIvP6hnZf1WZUXoOLRg9eFhSvGNoVwvdRLNXSmDmyHBwW4co
atc7TlJFGa8kBpJIERqLrqwYElesA8u49L3KJg6nwd3jM+/AVTANlVlOnAM2BvjA
jxSZnE0qnsHhfTuvcqdFuhOWKU4Z0BqYBvQ3lBetoxi6PrABDJXWKTUgNX31EGDk
92hiHuwZ4STyhxGs6QiA
-----END CERTIFICATE-----
```


```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"c1.pem" 28L, 1636C
```

Step 2: Extract the public key (e, n) from the issuer's certificate.

`Openssl` provides commands to extract certain attributes from the x509 certificates. We can extract the value of n using -modulus.To get the modulus value(n) : `openssl x509 -in c1.pem -noout -modulus`

```
[10/15/20]seed@VM:~/bin/lav$ openssl x509 -in c1.pem -noout -modulus
Modulus=D3487CBEF305865D5BD52F854E4BE086AD15AC61CF5BAF3E6A0A47FB9A7691600B8A6BCDCFDC577E60980BE454D956ED21CC02B65A815F976AEE022F2327B86DD4B0E70602780B1F5CA99936FEBBAC1B
05FA57CD81104067D6308B5835D49661BED08C7A979F1AF922E6142FA9C6E8011FABF8260FAC8E4D2C32391D819B8D1C65B21CDB61A8892F60E7EBC24A18C46F2AE9109209ED17D1002BE67DEF0489144E33A1B2
0F97879FB3A0CD2FBC2CECB88368313D1FD54A9010190B8195D6297651F93676D0B7097A384AD76F8CBF137C39EDBAAE90FC95F77B7809365E74931E25F0FFD4ADAE686BC6FF0FD535F1556E4849F8F8B8EF88F8
F15E1177AADF02B3
[10/15/20]seed@VM:~/bin/lav$
```

To find the exponent value(e): `openssl x509 -in c1.pem -text -noout`

```
[10/15/20]seed@VM:~/bin/lav$ openssl x509 -in c1.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0c:8e:e0:c9:0d:6a:89:15:88:04:06:1e:e2:41:f9:af
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert Global Root G2
        Validity
            Not Before: Aug  1 12:00:00 2013 GMT
            Not After : Aug  1 12:00:00 2028 GMT
        Subject: C=US, O=DigiCert Inc, CN=DigiCert Global CA G2
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d3:48:7c:be:f3:05:86:5d:5b:d5:2f:85:4e:4b:
                    e0:86:ad:15:ac:61:cf:5b:af:3e:6a:0a:47:fb:9a:
                    76:91:60:0b:8a:6b:cd:cf:dc:57:7e:60:98:0b:e4:
                    54:d9:56:ed:21:cc:02:b6:5a:81:5f:97:6a:ee:02:
                    2f:23:27:b8:6d:d4:b0:e7:06:02:78:0b:1f:5c:a9:
                    99:36:fe:bb:ac:1b:05:fa:57:cd:81:10:40:67:d6:
                    30:8b:58:35:d4:96:61:be:d0:8c:7a:97:9f:1a:f9:
                    22:e6:14:2f:a9:c6:e8:01:1f:ab:f8:26:0f:ac:8e:
                    4d:2c:32:39:1d:81:9b:8d:1c:65:b2:1c:db:61:a8:
                    89:2f:60:e7:eb:c2:4a:18:c4:6f:2a:e9:10:92:09:
                    ed:17:d1:00:2b:e6:7d:ef:04:89:14:4e:33:a1:b2:
                    0f:97:87:9f:b3:a0:cd:2f:bc:2c:ec:b8:83:68:31:
                    3d:1f:d5:4a:90:10:19:0b:81:95:d6:29:76:51:f9:
                    36:76:d0:b7:09:7a:38:4a:d7:6f:8c:bf:13:7c:39:
                    ed:ba:ae:90:fc:95:f7:7b:78:09:36:5e:74:93:1e:
                    25:f0:ff:d4:ad:ae:68:6b:c6:ff:0f:d5:35:f1:55:
                    6e:48:49:f8:f8:b8:ef:88:f8:f1:5e:11:77:aa:df:
                    02:b3
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
            Authority Information Access:
                OCSP - URI:http://ocsp.digicert.com

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl4.digicert.com/DigiCertGlobalRootG2.crl
```

Step 3: Extract the signature from the server's certificate.

```
[10/15/20]seed@VM:~/bin/lav$ openssl x509 -in c0.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:6d:cd:0e:6d:c0:39:e4:a6:c5:36:96:9b:a7:6c:9a
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, CN=DigiCert Global CA G2
        Validity
            Not Before: Jan 23 00:00:00 2020 GMT
            Not After : Dec 31 12:00:00 2020 GMT
        Subject: C=US, ST=Washington, L=Seattle, O=Amazon.com, Inc., CN=www.amazon.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a5:88:2c:18:90:5d:47:f1:9b:e0:d7:eb:2a:b2:
                    dd:c0:02:90:6c:f8:93:9f:4b:7f:18:c2:76:22:6b:
                    4b:7e:9e:16:87:af:c9:a4:f8:89:99:f5:25:fe:e6:
                    89:18:17:87:56:8e:1c:f5:9a:13:33:55:8b:d6:6c:
                    92:e3:ce:49:0d:ad:9e:84:a1:22:b8:d2:26:d8:4f:
                    f8:50:1d:a4:af:84:6f:a9:59:a4:6b:39:62:34:45:
                    97:8c:88:1a:f6:da:3f:b4:64:8c:26:62:ec:57:91:
                    48:c3:b3:1c:d8:dd:45:1c:ba:fe:ec:6d:03:b8:42:
                    c4:23:b9:38:1d:b0:71:07:06:72:f2:35:d1:46:4a:
                    ac:99:23:6c:cc:47:a1:33:ab:3c:73:23:ed:c4:45:
                    e5:dd:b2:a9:ba:1c:f5:6f:cf:c7:20:46:9c:d9:21:
                    78:6b:a5:84:6b:bf:4d:9b:4e:55:ae:f8:25:f1:68:
                    77:65:a2:25:60:52:0a:76:a6:aa:db:5e:c5:7c:3d:
                    2a:66:02:2d:2b:40:0c:9c:90:70:a7:b5:23:78:59:
                    65:64:04:77:0a:eb:be:4c:56:eb:7f:b0:ad:d2:33:
                    90:f4:df:db:d0:2a:03:48:89:ca:20:97:10:f7:a5:
                    ac:35:84:72:b1:4d:b6:20:c2:02:9a:84:ab:73:8e:
                    b0:77
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:24:6E:2B:2D:D0:6A:92:51:51:25:69:01:AA:9A:47:A6:89:E7:40:20

            X509v3 Subject Key Identifier:
                98:8E:BD:45:84:42:E8:55:39:18:6A:62:33:7B:01:06:5C:88:29:72
            X509v3 Subject Alternative Name:
                DNS:amazon.com, DNS:amzn.com, DNS:buybox.amazon.com, DNS:corporate.amazon.com, DNS:home.amazon.com, DNS:iphone.amazon.com, DNS:konrad-test.amazon.com, D
NS:mp3recs.amazon.com, DNS:p-nt-www-amazon-com-kalias.amazon.com, DNS:p-y3-www-amazon-com-kalias.amazon.com, DNS:p-yo-www-amazon-com-kalias.amazon.com, DNS:static.amazo
n.com, DNS:test-www.amazon.com, DNS:uedata.amazon.com, DNS:us.amazon.com, DNS:www.amazon.com, DNS:www.amzn.com, DNS:www.cdn.amazon.com, DNS:www.m.amazon.com, DNS:yellow
pages.amazon.com, DNS:yp.amazon.com
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:
```

```
        X509v3 CRL Distribution Points:

            Full Name:
              URI:http://crl3.digicert.com/DigiCertGlobalCAG2.crl

            Full Name:
              URI:http://crl4.digicert.com/DigiCertGlobalCAG2.crl

        X509v3 Certificate Policies:
            Policy: 2.16.840.1.114412.1.1
              CPS: https://www.digicert.com/CPS
            Policy: 2.23.140.1.2.2

        Authority Information Access:
            OCSP - URI:http://ocsp.digicert.com
            CA Issuers - URI:http://cacerts.digicert.com/DigiCertGlobalCAG2.crt

        X509v3 Basic Constraints:
            CA:FALSE
        CT Precertificate SCTs:
            Signed Certificate Timestamp:
                Version   : v1(0)
                Log ID    : B2:1E:05:CC:8B:A2:CD:8A:20:4E:87:66:F9:2B:B9:8A:
                            25:20:67:6B:DA:FA:70:E7:B2:49:53:2D:EF:8B:90:5E
                Timestamp : Jan 23 22:49:03.072 2020 GMT
                Extensions: none
                Signature : ecdsa-with-SHA256
                            30:46:02:21:00:DB:FE:F6:EE:30:AC:35:3A:DC:76:15:
                            0A:C6:DB:72:C6:5B:E0:9A:B9:47:DA:11:77:2D:48:05:
                            2B:F6:F0:94:1A:02:21:00:BB:8D:2B:98:38:57:38:0D:
                            E9:37:18:C5:4D:D2:10:13:20:F8:62:DC:78:5F:82:1D:
                            90:FA:E2:95:C1:04:F0:BD
            Signed Certificate Timestamp:
                Version   : v1(0)
                Log ID    : F0:95:A4:59:F2:00:D1:82:40:10:2D:2F:93:88:8E:AD:
                            4B:FE:1D:47:E3:99:E1:D0:34:A6:B0:A8:AA:8E:B2:73
                Timestamp : Jan 23 22:49:03.129 2020 GMT
                Extensions: none
                Signature : ecdsa-with-SHA256
                            30:45:02:20:52:43:F5:E9:3F:D8:AE:FE:E0:9E:8F:00:
                            88:AB:F5:35:F8:D2:0F:AC:38:34:24:89:1D:47:7D:1E:
                            23:27:36:71:02:21:00:DF:8D:AE:41:E8:AD:4A:73:8D:
                            00:47:73:46:B3:9C:E7:89:10:70:44:4C:E5:EB:D7:D9:
                            33:EA:8B:40:1E:51:98
```

```
  Signature Algorithm: sha256WithRSAEncryption
       46:68:db:7e:7a:0e:e0:58:74:4d:f5:6f:80:7c:a9:47:61:1c:
       86:f8:82:b8:de:36:18:cc:66:1b:39:b3:9f:98:e9:62:af:ad:
       1d:bb:9b:0f:d3:b3:6b:04:d9:8d:36:b5:b8:24:22:d2:6c:ed:
       79:fe:0e:03:91:94:8e:24:3e:ca:f1:e8:a3:b9:8d:33:85:c1:
       76:82:08:03:2a:ae:b0:be:0e:90:72:0c:b8:1b:58:a7:3c:03:
       3b:d1:f3:2d:6f:21:ef:42:7e:14:32:c8:12:c6:cb:a8:17:be:
       0f:8c:b8:0e:1a:dd:eb:fa:ba:73:9c:aa:02:e8:99:8a:82:19:
       a6:c2:25:51:ee:50:b7:90:e4:34:78:80:1d:d6:fe:9a:dc:e1:
       be:e6:b3:4c:e4:06:c1:b7:44:9e:f9:ad:83:f2:40:85:32:a4:
       b9:b3:81:9e:fd:89:12:9d:ce:20:03:54:36:24:b7:e8:1d:78:
       b6:ae:38:eb:eb:4c:f9:d0:d5:b6:1b:29:82:b9:e9:22:9a:32:
       74:fc:45:ed:bf:d2:00:d6:e4:6d:9e:6e:7a:93:d2:ec:01:0a:
       99:66:9a:50:1c:93:0a:f8:5e:e2:ec:a7:27:2b:4d:fe:16:98:
       df:cd:21:37:8c:70:13:f0:99:31:26:5d:d1:a9:56:04:03:82:
       9e:74:06:1e
```

We need to remove the spaces and colons from the data, so we can get a hex-string that we can feed into our program. The “tr” command is a Linux utility tool for string operations. In this case, the “-d” option is used to delete ":" and "space" from the data. Save the signature in a file named “signature”.

```
46:68:db:7e:7a:0e:e0:58:74:4d:f5:6f:80:7c:a9:47:61:1c:
       86:f8:82:b8:de:36:18:cc:66:1b:39:b3:9f:98:e9:62:af:ad:
       1d:bb:9b:0f:d3:b3:6b:04:d9:8d:36:b5:b8:24:22:d2:6c:ed:
       79:fe:0e:03:91:94:8e:24:3e:ca:f1:e8:a3:b9:8d:33:85:c1:
       76:82:08:03:2a:ae:b0:be:0e:90:72:0c:b8:1b:58:a7:3c:03:
       3b:d1:f3:2d:6f:21:ef:42:7e:14:32:c8:12:c6:cb:a8:17:be:
       0f:8c:b8:0e:1a:dd:eb:fa:ba:73:9c:aa:02:e8:99:8a:82:19:
       a6:c2:25:51:ee:50:b7:90:e4:34:78:80:1d:d6:fe:9a:dc:e1:
       be:e6:b3:4c:e4:06:c1:b7:44:9e:f9:ad:83:f2:40:85:32:a4:
       b9:b3:81:9e:fd:89:12:9d:ce:20:03:54:36:24:b7:e8:1d:78:
       b6:ae:38:eb:eb:4c:f9:d0:d5:b6:1b:29:82:b9:e9:22:9a:32:
       74:fc:45:ed:bf:d2:00:d6:e4:6d:9e:6e:7a:93:d2:ec:01:0a:
       99:66:9a:50:1c:93:0a:f8:5e:e2:ec:a7:27:2b:4d:fe:16:98:
       df:cd:21:37:8c:70:13:f0:99:31:26:5d:d1:a9:56:04:03:82:
       9e:74:06:1e
"signature" 16L, 909C
```

```
cat signature | tr -d ’[:space:]:’
```

```
[10/15/20]seed@VM:~/bin/lav$ cat signature | tr -d '[:space:]:'
4668db7e7a0ee058744df56f807ca947611c86f882b8de3618cc661b39b39f98e962afad1dbb9b0fd3b36b04d98d36b5b82422d26ced79fe0e0391948e243ecaf1e8a3b98d3385c1768208032aaeb0be0e90720c
b81b58a73c033bd1f32d6f21ef427e1432c812c6cba817be0f8cb80e1addebfaba739caa02e8998a8219a6c22551ee50b790e43478801dd6fe9adce1bee6b34ce406c1b7449ef9ad83f2408532a4b9b3819efd89
129dce2003543624b7e81d78b6ae38ebeb4cf9d0d5b61b2982b9e9229a3274fc45edbfd200d6e46d9e6e7a93d2ec010a99669a501c930af85ee2eca7272b4dfe1698dfcd21378c7013f09931265dd1a956040382
9e74061e[10/15/20]seed@VM:~/bin/lav$
```

Step 4: Extract the body of the server's certificate.

A Certificate Authority (CA) generates the signature for a server certificate by first computing the hash of the certificate, and then sign the hash. To verify the signature, we also need to generate the hash from a certificate. Since the hash is generated before the signature is computed, we need to exclude the signature block of a certificate when computing the hash.

Openssl has a command called asn1parse, which can be used to parse a X.509 certificate.

```
e74061e[10/15/20]seed@VM:~/bin/lav$ openssl asn1parse -i -in c0.pem
   0:d=0  hl=4 l=1990 cons: SEQUENCE
   4:d=1  hl=4 l=1710 cons:  SEQUENCE
   8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
  10:d=3  hl=2 l=   1 prim:    INTEGER           :02
  13:d=2  hl=2 l=  16 prim:   INTEGER           :046DCD0E6DC039E4A6C536969BA76C9A
  31:d=2  hl=2 l=  13 cons:   SEQUENCE
  33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
  44:d=3  hl=2 l=   0 prim:    NULL
  46:d=2  hl=2 l=  68 cons:   SEQUENCE
  48:d=3  hl=2 l=  11 cons:    SET
  50:d=4  hl=2 l=   9 cons:     SEQUENCE
  52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
  57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
  61:d=3  hl=2 l=  21 cons:    SET
  63:d=4  hl=2 l=  19 cons:     SEQUENCE
  65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
  70:d=5  hl=2 l=  12 prim:      PRINTABLESTRING   :DigiCert Inc
  84:d=3  hl=2 l=  30 cons:    SET
  86:d=4  hl=2 l=  28 cons:     SEQUENCE
  88:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  93:d=5  hl=2 l=  21 prim:      PRINTABLESTRING   :DigiCert Global CA G2
 116:d=2  hl=2 l=  30 cons:   SEQUENCE
 118:d=3  hl=2 l=  13 prim:    UTCTIME           :200123000000Z
 133:d=3  hl=2 l=  13 prim:    UTCTIME           :201231120000Z
 148:d=2  hl=2 l= 104 cons:   SEQUENCE
 150:d=3  hl=2 l=  11 cons:    SET
 152:d=4  hl=2 l=   9 cons:     SEQUENCE
 154:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
 159:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
 163:d=3  hl=2 l=  19 cons:    SET
 165:d=4  hl=2 l=  17 cons:     SEQUENCE
 167:d=5  hl=2 l=   3 prim:      OBJECT            :stateOrProvinceName
 172:d=5  hl=2 l=  10 prim:      PRINTABLESTRING   :Washington
 184:d=3  hl=2 l=  16 cons:    SET
 186:d=4  hl=2 l=  14 cons:     SEQUENCE
 188:d=5  hl=2 l=   3 prim:      OBJECT            :localityName
 193:d=5  hl=2 l=   7 prim:      PRINTABLESTRING   :Seattle
 202:d=3  hl=2 l=  25 cons:    SET
 204:d=4  hl=2 l=  23 cons:     SEQUENCE
 206:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
 211:d=5  hl=2 l=  16 prim:      PRINTABLESTRING   :Amazon.com, Inc.
 229:d=3  hl=2 l=  23 cons:    SET
 231:d=4  hl=2 l=  21 cons:     SEQUENCE
 233:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
 238:d=5  hl=2 l=  14 prim:      PRINTABLESTRING   :www.amazon.com
 254:d=2  hl=4 l= 290 cons:   SEQUENCE
 258:d=3  hl=2 l=  13 cons:    SEQUENCE
 260:d=4  hl=2 l=   9 prim:     OBJECT            :rsaEncryption
 271:d=4  hl=2 l=   0 prim:     NULL
 273:d=3  hl=4 l= 271 prim:    BIT STRING
 548:d=2  hl=4 l=1166 cons:   cont [ 3 ]
```

```
 548:d=2  hl=4 l=1166 cons:   cont [ 3 ]
 552:d=3  hl=4 l=1162 cons:    SEQUENCE
 556:d=4  hl=2 l=  31 cons:     SEQUENCE
 558:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Authority Key Identifier
 563:d=5  hl=2 l=  24 prim:      OCTET STRING      [HEX DUMP]:30168014246E2B2DD06A925151256901AA9A47A689E74020
 589:d=4  hl=2 l=  29 cons:     SEQUENCE
 591:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject Key Identifier
 596:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:0414988EBD458442E85539186A62337B01065C882972
 620:d=4  hl=4 l= 454 cons:     SEQUENCE
 624:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject Alternative Name
 629:d=5  hl=4 l= 445 prim:      OCTET STRING      [HEX DUMP]:308201B9820A616D617A6F6E2E636F6D8208616D7A6E2E636F6D8211627579626F782E616D617A6F6E2E636F6D8214636F72706F7
26174652E616D617A6F6E2E636F6D820F686F6D652E616D617A6F6E2E636F6D82116970686F6E652E616D617A6F6E2E636F6D82166B6F6E7261642D746573742E616D617A6F6E2E636F6D82126D7033726563732
E616D617A6F6E2E636F6D8225702D6E742D7777772D616D617A6F6E2D636F6D2D6B616C6961732E616D617A6F6E2E636F6D8225702D79332D7777772D616D617A6F6E2D636F6D2D6B616C6961732E616D617A6F6
E2E636F6D8225702D796F2D7777772D616D617A6F6E2D636F6D2D6B616C6961732E616D617A6F6E2E636F6D821737461746963732E616D617A6F6E2E636F6D8213746573742D7777772E616D617A6F6E2E636F6D8
21175656461746 12E616D617A6F6E2E636F6D820D75732E616D617A6F6E2E636F6D820E7777772E616D617A6F6E2E636F6D820C7777772E616D7A6E2E636F6D82127777772E63646E2E616D617A6F6E2E636F6D8
2107777772E6D2E616D617A6F6E2E636F6D821679656C6C6F7770616765732E616D617A6F6E2E636F6D820D79702E616D617A6F6E2E636F6D
 1078:d=4  hl=2 l=  14 cons:     SEQUENCE
 1080:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Key Usage
 1085:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
 1088:d=5  hl=2 l=   4 prim:      OCTET STRING      [HEX DUMP]:030205A0
 1094:d=4  hl=2 l=  29 cons:     SEQUENCE
 1096:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Extended Key Usage
 1101:d=5  hl=2 l=  22 prim:      OCTET STRING      [HEX DUMP]:301406082B0601050507030106082B06010505070302
 1125:d=4  hl=2 l= 119 cons:     SEQUENCE
 1127:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 CRL Distribution Points
 1132:d=5  hl=2 l= 112 prim:      OCTET STRING      [HEX DUMP]:306E3035A033A031862F687474703A2F2F63726C332E6469676963572742E636F6D2F4469676943657274476C6F62616C4341473
22E63726C3035A033A031862F687474703A2F2F63726C342E6469676963572742E636F6D2F4469676943657274476C6F62616C434147322E63726C
 1246:d=4  hl=2 l=  76 cons:     SEQUENCE
 1248:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Certificate Policies
 1253:d=5  hl=2 l=  69 prim:      OCTET STRING      [HEX DUMP]:3043303706096086480186FD6C0101302A302806082B0601050507020116 1C68747470733A2F2F7777772E6469676963572742E6
36F6D2F4350533008060667810C010202
 1324:d=4  hl=2 l= 116 cons:     SEQUENCE
 1326:d=5  hl=2 l=   8 prim:      OBJECT            :Authority Information Access
 1336:d=5  hl=2 l= 104 prim:      OCTET STRING      [HEX DUMP]:3066302406082B0601050507300186 1868747470 3A2F2F6F6373702E6469676963572742E636F6D303E06082B0601050507300 28
632687474703A2F2F6361636572747 32E6469676963572742E636F6D2F4469676943657274476C6F62616C434147322E637274
 1442:d=4  hl=2 l=   9 cons:     SEQUENCE
 1444:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Basic Constraints
 1449:d=5  hl=2 l=   2 prim:      OCTET STRING      [HEX DUMP]:3000
 1453:d=4  hl=4 l= 261 cons:     SEQUENCE
 1457:d=5  hl=2 l=  10 prim:      OBJECT            :CT Precertificate SCTs
 1469:d=5  hl=3 l= 246 prim:      OCTET STRING      [HEX DUMP]:0481F300F1007700B21E05CC8BA2CD8A204E8766F92BB98A2520676BDAFA70E7B249532DEF8B905E0000016FD4983760000004030
0483046022100DBFEF6EE30AC353ADC76150AC6DB72C65BE09AB947DA11772D48052BF6F0941A022100BB8D2B983857380DE93718C54DD2101320F862DC785F821D90FAE295C104F0BD007600F095A459F200D18
240102D2F93888EAD4BFE1D47E399E1D034A6B0A8AA8EB2730000016FD4983799000004030047304502205243F5E93FD8AEFEE09E8F0088ABF535F8D20FAC383424891D477D1E23273671022100DF8DAE41E8AD4
A738D004773 46B39CE7891070444CE5EBD7D933EA8B401E5198
 1718:d=1  hl=2 l=  13 cons:  SEQUENCE
 1720:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1731:d=2  hl=2 l=   0 prim:   NULL
 1733:d=1  hl=4 l= 257 prim:  BIT STRING
[10/15/20]seed@VM:~/bin/lav$
```

In this case, the Certificate body is from offset 4 to 1717, while the Signature block is from 1718 to the end of the file.

We can use the `-strparse` option to get the field from the offset 4, which will give us the body of the certificate, excluding the signature block for hashing.

We can calculate its hash using the following command:

```
[10/15/20]seed@VM:~/bin/lav$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
[10/15/20]seed@VM:~/bin/lav$ sha256sum c0_body.bin
662052b1de8743a7e8801af9e476c0eae5c2eacb4493a3fde4efa76386e544eb  c0_body.bin
[10/15/20]seed@VM:~/bin/lav$
```

Step 5: Verify the signature.

Now we verify if the hash from the server matches our pre-computed hash .

```
------------------------------------------------------------------
Task 6 - Manually Verifying an X.509 Certificate
the public key is:  0xD3487CBEF305865D5BD52F854E4BE086AD15AC61CF5BAF3E6A0A47FB9A7691600B8A6BCDCFDC577E60980BE454D956ED21CC02B65A815F976AEE022F2327B86DD4B0E70602780B1F5C
A99936FEBBAC1B05FA57CD81104067D6308B5835D49661BED08C7A979F1AF922E6142FA9C6E8011FABF8260FAC8E4D2C32391D819B8D1C65B21CDB61A8892F60E7EBC24A18C46F2AE9109209ED17D1002BE67DEF
0489144E33A1B20F97879FB3A0CD2FBC2CECB88368313D1FD54A9010190B8195D6297651F93676D0B7097A384AD76F8CBF137C39EDBAAE90FC95F77B7809365E74931E25F0FFD4ADAE686BC6FF0FD535F1556E48
49F8F8B8EF88F8F15E1177AADF02B3
Hash from the server:  0x01FFFF
```