Jonathan Smith

Prof Nhut Nguyen

CS 4393.001

24 October 2023

RSA Public-Key Encryption and Signature Lab

Task 1

```
[10/23/23]seed@VM:~/.../Lab03$ gcc task1.c -o task1 -lcrypto
[10/23/23]seed@VM:~/.../Lab03$ ./task1
Private Key (d) =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496
AEB
```

The above screenshot shows the private key generated from the c code located in the task1.txt file within the compressed “code” file included speratley. The RSA encryption algorithm generates its public key through a series of complex math to prevent attackers from gaining any more information givent that they have gotten either the plaintext, ciphertext, or keys. The code takes in given p, q, and e values and calculates n = p * q, totient = (p-1)(q-1), and d = e^-1 mod(totient(n))), where d is the private key, then prints it out.

Task 2

```
[10/23/23]seed@VM:~/.../Lab03$ gcc task2.c -o task2 -lcrypto
[10/23/23]seed@VM:~/.../Lab03$ ./task2
Encrption =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

Proceeding with the demonstration of the RSA encryption algorithm, the above screen shot is product of the code located in task2.txt in the compressed “code” folder which encrypts a message given. The code in task2.txt calculates the value of the encrypted message with the known equation, encrypted message= M^e * mod(n), where M, n, and e are given in the lab document.

Task 3

```
[10/23/23]seed@VM:~/.../Lab03$ gcc task3.c -o task3 -lcrypto
[10/23/23]seed@VM:~/.../Lab03$ ./task3
Decryption =  50617373776F72642069732064656573
```

The above screenshot is the demonstration of the process opposite of what was given in task 2, where we are given a ciphertext and are told to decypher it. The code used to this is located in task3.txt in the compressed "code" file. The code decrypts the message by calculating the known equation decrypted message = C^d mod(n), where C, d, and n are given in the lab document.

```
[10/23/23]seed@VM:~/.../Lab03$ python3 -c 'print(bytes.fromhex("50617373776F7264
2069732064656573").decode("utf-8"))'
Password is dees
```

With the code in task3.txt giving the decrypted message in the form of hexidecimal, the above screenshot is the transformation of the hexidecimal encrypted message into letters, printing out the phrase "Password is dees."

Task 4

```
[10/24/23]seed@VM:~/.../Lab03$ python3 -c 'print("I owe you $2000.".encode("utf-
8").hex())'
49206f776520796f752024323030302e
```

```
[10/24/23]seed@VM:~/.../Lab03$ gcc task4.c -o task4 -lcrypto
[10/24/23]seed@VM:~/.../Lab03$ ./task4
Signature =  55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB
```

The first screen shot is the conversion of a given message in to a hexidecimal which will be hardcoded into the code located in task4.txt to generate a signature to provide authentication. The second screen shot is the signature produced by task4.txt where signature = M^d mod(n) and M, d, and n are given in the lab.

```
[10/24/23]seed@VM:~/.../Lab03$ python3 -c 'print("I owe you $3000.".encode("utf-
8").hex())'
49206f776520796f752024333030302e
```

```
[10/24/23]seed@VM:~/.../Lab03$ gcc task4.c -o task4 -lcrypto
[10/24/23]seed@VM:~/.../Lab03$ ./task4
Signature =  BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822
```

The above two screen shots are a repeat of task 4 but with a slightly different message, but producing a completley different signature, almost baring no resemblance to the first signature that was gerneated. This is a good thing for security purposes as this makes it harder for hackers to decrypt and steal peoples messages if given the chance.

Task 5

```
[10/24/23]seed@VM:~/.../Lab03$ gcc task5.c -o task5 -lcrypto
[10/24/23]seed@VM:~/.../Lab03$ ./task5
Message (hex) =  4C61756E63682061206D697373696C652E
[10/24/23]seed@VM:~/.../Lab03$ python3 -c 'print(bytes.fromhex("4C61756E63682061
206D697373696C652E").decode("utf-8"))'
Launch a missile.
```

```
[10/24/23]seed@VM:~/.../Lab03$ gcc task5.c -o task5 -lcrypto
[10/24/23]seed@VM:~/.../Lab03$ ./task5
Message (hex) =  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC29
4
[10/24/23]seed@VM:~/.../Lab03$ python3 -c 'print(bytes.fromhex("91471927C80DF1E4
2C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294").decode("iso-8859-1"))'
ñä,O´cè¼rm=fÈ:N¶·¾´Â
```

The first screen shot is the product of task5.txt located in the compressed code folder. Task5.txt calculates the message given the signature and other values with the known equation verification = S^e mod(n), where S, e, and n are all given in the lab. The second screen shot tries to turn the signature back into the message, but one bit is corrupted, causing the code to spit out something that doesn't even closely resemble the original message "Launch a missle."

Task 6

```
[10/24/23]seed@VM:~/.../Lab03$ openssl s_client -connect www.domain.com:443 -sho
wcerts
```

```
[10/24/23]seed@VM:~/.../Lab03$ openssl x509 -in certificate1.pem -noout -modulus
Modulus=EB48BCECB0A61915C61D1320D47C3EB3A2B0789A823E50A4035CC684EFF7534C7D839DFA
96C8F59F3C11DAE66423592F7D612847C934B16A4B461046EC701C11B0BDDEF612E921FE8AD695C4
2FA523C218869530BE05B8B2ECA6DBD006838FFD28A8168B99371CAE33A1C33537FFC4CF7E2C8853
44818354CB249EDF4BA7EAA5B90EB99B0CEB8F10C3CE095370ADC5D471EFC4767DD7976E87264A18
CCE9BA0D5D302E9271F4DF38EA393BE48327FA580CB8074BA71E5B6DB29A92210122BC4E1F95780E
3CDE01CBAD07AB5C6437C42224B6325EBE0B3C09D229F391A64E14F8749F73A0966CA43E8032D0BB
7AFD6480D311AC094922992F8FD0C527109739C3
```

```
[10/24/23]seed@VM:~/.../Lab03$ openssl x509 -in c1.pem -text -noout| grep "Expon
ent"
                Exponent: 65537 (0x10001)
```

```
[10/24/23]seed@VM:~/.../Lab03$ cat signature | tr -d '[:space:]:'
46a8c14c5c1939748d2402f126f15dea60050e367233a3f81325d6e38e99c0d5032fc4514acf4883
7133e6df96d225173c402f9e86caeeecdb005f110237eab2579836330af88e8e93b3f8a34d0b6d92
2182c0e26ab0380195ce0ece31506efffb026e5745f9d508d6b534a0c3111b9bafac7257fcc19b46
2ccc040510ce5ec9d472694221937d4651663a4af2e81da19a21b6f50c925f9bf09f8145b40f4f4f
722f0e699b9c11488330f2ef1add8318658bd498e653e6bdca98da8dcfd5772b87b8e9262ba6328c
ba6dbf152662c2e7fb172224ab3f96071903723c5d9cebdc69282c9b0e5f0e3e133f95225c3debee
```

```
[10/24/23]seed@VM:~/.../Lab03$ gcc task6.c -o task6 -lcrypto
[10/24/23]seed@VM:~/.../Lab03$ ./task6
Message (hex) =  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFF003031300D0609608648016503040201050004208558EFF2CDE32CA7A
2BDF849EAFBC97D5739BFA08B9ECE4EB0C34CE2B18FCD23
[10/24/23]seed@VM:~/.../Lab03$ openssl asn1parse -i -in c0.pem -strparse 4 -out
c0_body.bin -noout
[10/24/23]seed@VM:~/.../Lab03$ sha256sum c0_body.bin
8558eff2cde32ca7a2bdf849eafbc97d5739bfa08b9ece4eb0c34ce2b18fcd23  c0_body.bin
```

The above screen shots are the subsequent steps given in the lab. Cascading down the list of screen shots, the first one is the line of code used to generate both certificates given in the code folder labled as c0.txt and c1.txt. Screenshots 2, 3, and 4 show the generation of variables, n, e, and s respecitvely, which are needed to verify the signature of the website with the equation verification = S^e mod(n). These values are hardcoded into task6.txt and turned into our predicted hash of the website, which is compared to the actual hash shown in the last screen shot. The hash generated matches the actual hash of the website aside from the 'F's which are meant to serve as padding, therefore the certificate is correctly varified.