

# Computer and Information Security

(ECE560, Fall 2024, Duke Univ., Prof. Tyler Bletsch)

## Homework 2

Name: xxx

Duke NetID: xxx

### Instructions - **read all carefully:**

- **COMPUTERS YOU WILL NEED:** We'll use the computers described below.
  - From the Duke VCM service:
    - VCM "ECE 560 F24 Ubuntu 22.04": this is your **Linux VM**.
    - VCM "ECE 560 F24 Win10": this is your **Windows VM**.
    - New VCM "ECE 560 F24 Kali": this is your **Kali Linux VM**.
  - We'll refer to your own machine on Duke wifi as your **personal computer**; this may be Windows, Linux, or Mac.
- **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**.
  - 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.  
Staff reserves the right to penalize submissions that flagrantly fail to do this.
  - We're looking for **synthesis of understanding**: That you can demonstrate understanding via novel thought. So if I ask "Explain what DNS is", an optimal answer will be a description of the *what*, *why*, and *how* of DNS in your own words. Answers which simply quote or closely paraphrase will not receive credit.
  - **Many questions will require research on your part**. The answers will often not be in the slides.
  - Follow directions carefully! For long/complex questions, the actions or items you need to submit are marked in **cyan**.
- **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 **Canvas 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.**

1. `cp dukevm dukevm.privpem`
2. `ssh-keygen -p -N "" -m pem -f dukevm.privpem`
3. `openssl rsautl -decrypt -inkey dukevm.privpem -in secret-fg96.key.enc -out decrypted.txt`
4. `openssl enc -d -aes-256-cbc -in data-fg96.dat -out decrypted_output.dat -K 90302d13f3122849e086fa77bd6fd85e4149b30350f95d77a2ad9bcd6ad54071 -iv dd1c46682156e180e85786dd821e9e59`
5. `sha256sum decrypted_output.dat`
6. `nslookup target.colab.duke.edu`

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

3348a19a0c57d7f2b6dcadd42d88c6d11ccaabee

## Question 2: Chapters 2, 20, and 21 - Cryptography (18 points)

Answer the questions below. (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?  
Inputs for encryption function: plaintext, secret key  
Outputs for encryption function: ciphertext  
Inputs for decryption function: ciphertext, secret key  
Outputs for decryption function: plaintext
- b. How many keys are required for two people to communicate via a symmetric cipher?  
One
- c. What is a message authentication code?  
One authentication technique involves the use of a secret key to generate a small block of data, known as a message authentication code, that is appended to 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 44 or so.)
  - Figure 2.5(a) is a message digest encrypted using symmetric encryption, source A takes the original message and passes it through a hash function, generating a hash value, and the hash value is then encrypted using a symmetric key. The encryption process creates a MAC, which is sent along with the original message. Destination B receives both the original message and the MAC. It performs the same hash operation on the received message. The MAC is decrypted using the same symmetric key, and the resulting hash value is compared with the newly generated hash from destination B's version of the message. If they are the same, then the integrity of the data is guaranteed.
  - Figure 2.5(b) is a message digest encrypted using public key. Source A creates a hash of the message using a hash function. The resulting hash value is encrypted using source A's private key, creating the MAC. The MAC and the original message are sent to destination B. Destination B receives both and generates a hash of the received message using the same hash function. The MAC is decrypted using source A's public key, and the resulting decrypted hash value is compared with the hash that destination B computed. If they are the same, then the integrity of the data is guaranteed.
  - Figure 2.5(c) is a message digest encrypted using a secret value. Source A appends a secret key to the original message. The combination of the message + key is hashed using a hash function (H), creating a MAC. Source A sends the original message, without the key, to destination B. Destination B receives the message, appends the same shared secret key, and runs the same hash function to compute a new MAC. The computed MAC is then compared to the received MAC. If they match, the integrity and authenticity of the message are guaranteed.

- e. What properties must a hash function have to be useful for message authentication?
1. Hash function can be applied to a block of data of any size
  2. Hash function produces a fixed-length output
  3.  $H(x)$  is relatively easy to compute for any given  $x$ , making both hardware and software implementations practical.
  4. Preimage resistant: For any given code  $h$ , it is computationally infeasible to find  $x$  such that  $H(x) = h$ .
  5. Second preimage resistant: For any given block  $x$ , it is computationally infeasible to find  $y \neq x$  with  $H(y) = H(x)$ .
  6. It is computationally infeasible to find any pair  $(x, y)$  such that  $H(x) = H(y)$ .
- f. What problem with symmetric ciphers is solved by asymmetric ciphers?  
Asymmetric ciphers make it not necessary to do secure key exchange beforehand.
- g. What are the inputs and output of an asymmetric cipher's encryption function? Its decryption function?  
Input of encryption function: plaintext, public key  
Output of the encryption function: ciphertext  
Input of decryption function: ciphertext, private key  
Output of the decryption function: plaintext
- h. How many keys are required for two people to communicate via an asymmetric cipher?  
Two keys – a key pair of public key and 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.
1. Key generation: each party generates a key pair consisting of a public key and a private key.
  2. Key share: User A shares his public key with User B. User B wants to send a message to User A.
  3. Message encryption: User B uses the public key to encrypt the message and sends over the network.
  4. Message decryption: User A receives the ciphertext and use his own private key to decrypt the message.
- This use of cryptography defeats the threat model of:  
Asset: private user conversation  
Vulnerability: Packets may be intercepted and message will be read by a third party  
Attacker's capabilities/knowledge: know how to intercept message, eavesdrop and read the content user A sends to user B.
- 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.
1. Key generation: each party generates a key pair consisting of a public key and a private key.
  2. Hashing: Alice computes a cryptographic hash of the message.
  3. Generate Digital Signature: User A uses her private key to encrypt the hash value,

and attach the digital signature to the original message.

4. Decrypt the Digital Signature: When user B receives the message and the digital signature, it uses public key to decrypt the digital signature and retrieve the original hash.

5. Compute the hash: User B uses the same hash function to compute the hash of the received message.

6. Compare the hash: If two hashes are the same, the message was indeed signed by user A because only user A possesses the corresponding private key needed to create the signature. The message has also not altered during the transmission, which guarantees the integrity of the data.

k. What is a digital signature?

A digital signature is a cryptographic mechanism used to validate the authenticity and integrity of digital data. The signer applies a hash function to the original message, the hash value is then encrypted using the signer's private key, creating the digital signature. The hash value is then encrypted using the signer's private key, creating the digital signature. The recipient receives the message along with the digital signature and uses the signer's public key, the recipient decrypts the digital signature to obtain the original hash value. The recipient independently computes a hash of the received message and compare the hash computed with the hash value received. If match, then the message is not altered.

l. What is a public-key certificate?

Public-key certificate is an electronic document consists of a public key plus a user id of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority that is trusted by the user community.

m. What are the two general approaches to attacking cryptography?

Cryptanalysis attacks: exploit weaknesses in algorithm or manner of its use.

Brute-force attack: try all possible keys, stop when decrypted result seems readable.

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

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

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

It accomplish exchanging a secret symmetric key over an open communication channel without letting others steal the key in the middle.

Asset: key need to be exchanged secretly

Vulnerability: Packets may be intercepted and the symmetric key will be read by a third party

Attacker's capabilities/knowledge: know how to intercept message, eavesdrop and read the content user A sends to user B.

- q. Describe how symmetric and asymmetric cryptography can 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. What is the name for this approach?
1. The sender generates a random symmetric key called session key. The sender encrypts the large message using the symmetric session key
  2. The sender obtains the recipient's public key and encrypt the session key. The sender sends both the encrypted message and the encrypted symmetric key to the recipient.
  3. The recipient uses their private key to decrypt the encrypted symmetric key.
  4. Using the decrypted symmetric key, the recipient decrypts the large message
- The name of this approach is called hybrid encryption
- r. 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.
- ECB (Electronic Codebook) mode is the simplest and most straightforward encryption mode for block ciphers. In ECB, each block of plaintext is encrypted with the same key. This is bad because if the same plaintext block appears multiple times in the message, it will produce the same ciphertext block each time, this will leak information, and it is easy to find pattern within the structure and content of the plaintext. A better mode is called Cipher Block Chaining (CBC) Mode. It uses an initialization vector to ensure that identical plaintexts encrypt to different ciphertexts, this prevents attackers from identifying patterns based on ciphertext repetition. Besides, each ciphertext block depends on the plaintext block and the previous ciphertext block, this means a change in one plaintext block affects all subsequent ciphertext blocks, and thus improve the security.

### Question 3: Diffie-Hellman computation (3 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.

1. Compute Alice's public key:

$$A = g^a \bmod p = 23489^{23} \bmod 128903289043 = 34743366840$$

2. Compute Bob's public key:

$$B = g^b \bmod p = 23489^3 \bmod 128903289043 = 69330374869$$

3. Compute Alice's shared key:

$$S = A^b \bmod p = 34743366840^3 \bmod 128903289043 = 68870229433$$

4. Compute Bob's shared key:

$$S = B^a \bmod p = 69330374869^{23} \bmod 128903289043 = 68870229433$$

## 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. Alternatively, 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](#) which has been developed for this course. You can retrieve it by running:

```
$ wget https://people.duke.edu/~tkbl3/courses/ece560/homework/hw2/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 22.04 VM. Other platforms may give the error "RuntimeError: Bad magic number in .pyc file". If you want to develop on your Kali VM, this variant will work there: [cryptotest-kali.pyc](#). If you use this variant, note it clearly in an attached readme.

The 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:

```
tkb13@vcm-21420: ~  
tkbletsch@OBAMA: ~/560/Homework/Homework2/crypto-program $ python3 cryptotest.pyc example-duke-crypter-good  
cryptotest v2.0.1 by Tyler Bletsch (Tyler.Bletsch@duke.edu)  
  
Generating input file 'test_input'...  
  
Encrypting to 'test_ciphred'...  
$ ./example-duke-crypter-good -e test_input test_ciphred  
Passphrase:  
  
Encryption exit status == 0? [ ok] 1/ 1 pts  
  
Calculating compression ratio...  
Cipher compressability > 95%? [ ok] 3/ 3 pts (Ratio: 100.06%)  
  
Decrypting to 'test_ciphred_deciphred'...  
$ ./example-duke-crypter-good -d test_ciphred test_ciphred_deciphred  
Passphrase:  
  
Decryption exit status == 0? [ ok] 1/ 1 pts  
  
Comparing 'test_input' and 'test_ciphred_deciphred'...  
Decrypted content matches? [ ok] 6/ 6 pts  
  
Attempting decryption with wrong secret key to 'test_ciphred_deciphred_bad'...  
$ ./example-duke-crypter-good -d test_ciphred test_ciphred_deciphred_bad  
Passphrase: gpg: decryption failed: Bad session key  
  
Decryption exit status != 0? [ ok] 1/ 1 pts (Exit status 2).  
  
Comparing 'test_input' and 'test_ciphred_deciphred_bad'...  
Mis-decrypted content differs? [ ok] 4/ 4 pts  
  
Tampering with ciphertext to produce 'test_ciphred_tampered'...  
  
Attempting decryption of tampered file to 'test_ciphred_tampered_deciphred'...  
$ ./example-duke-crypter-good -d test_ciphred_tampered test_ciphred_tampered_deciphred  
Passphrase: gpg: WARNING: encrypted message has been manipulated!  
  
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.  
  
tkbletsch@OBAMA: ~/560/Homework/Homework2/crypto-program $
```

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

```
tkb13@vcm-21420: ~  
tkbletsch@OBAMA:~/560/Homework/Homework2/crypto-program $ python3 cryptotest.pyc example-duke-crypter-bad  
cryptotest v2.0.1 by Tyler Bletsch (Tyler.Bletsch@duke.edu)  
  
Generating input file 'test_input'...  
  
Encrypting to 'test_ciphred'...  
$ ./example-duke-crypter-bad -e test_input test_ciphred  
  
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_ciphred_deciphred'...  
$ ./example-duke-crypter-bad -d test_ciphred test_ciphred_deciphred  
  
Decryption exit status == 0? [ ok] 1/ 1 pts  
  
Comparing 'test_input' and 'test_ciphred_deciphred'...  
Decrypted content matches? [FAIL] 0/ 6 pts Deciphered file doesn't match input.  
  
Attempting decryption with wrong secret key to 'test_ciphred_deciphred_bad'...  
$ ./example-duke-crypter-bad -d test_ciphred test_ciphred_deciphred_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_ciphred_deciphred_bad'...  
Mis-decrypted content differs? [ ok] 4/ 4 pts  
  
Tampering with ciphertext to produce 'test_ciphred_tampered'...  
  
Attempting decryption of tampered file to 'test_ciphred_tampered_deciphred'...  
$ ./example-duke-crypter-bad -d test_ciphred_tampered test_ciphred_tampered_deciphred  
  
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.  
  
tkbletsch@OBAMA:~/560/Homework/Homework2/crypto-program $
```

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

```
cryptotest v2.0.1 by Tyler Bletsch (Tyler.Bletsch@duke.edu)
= Certified results report =
```

```
Binary under test: ./example-duke-crypter-bad
Test points: (1, 0, 1, 0, 0, 4, 0)
Total points: 6 / 20
Current username: tkbletsc
Current hostname: OBAMA
Timestamp: 2021-09-12 21:13:26.133890
```

```
Signatures:
f315c799324a9e42be4810f35b638631
f56eed3398785b4a26b16f4bda2e10b1
9c075f315765df739a421eb192370746
```

To prevent tampering with this report, the signatures at the bottom are Message Authentication Codes (MACs) of the cryptotest tool, your binary, 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 Canvas.**

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.

Specifically:

- [+5pts] Making a valid certificate that awards 20/20 without a valid encryption program.
- [+5pts] Making a valid certificate that appears to award 999/20.

To check if your forged certificate passes validation, you can use this tiny web tool:

<http://target.colab.duke.edu:9000/>

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

## Question 5: Calming down about Post-Quantum Crypto (3 points)

In lecture, I talked about development and standardization efforts around “Post-Quantum Cryptography” -- algorithms that hope to be resistant to cracking attempts by quantum computers. It’s worth noting that state-of-the-art quantum computers are nowhere near posing even the mildest threat to traditional cryptography algorithms; the danger is currently just theoretical.

As an interesting counterpoint, review this talk by noted computer scientist Peter Gutmann<sup>1</sup> entitled “[Why Quantum Cryptanalysis is Bollocks](#)”.

**Briefly summarize his key points. Do you agree with the conclusions in the talk? Why or why not?** (There’s not one correct answer here.)

The key points are that the quantum cryptanalysis should not be treated as a major threat to the security. No actual attacks (by using quantum cryptanalysis) have been successfully carried out, and the resources required are extreme. Therefore, instead of focusing on impractical cryptographic attacks, it is needed to focus more on more pressing and practical vulnerabilities. I agree with the conclusion in the talk, I believe that the quantum computing threat is still largely theoretical, and while quantum computers may one day be a threat, it's not one that demands immediate action.

## Question 6: Rolling Your Own Crypto Badly (3 points)

In 2023, a security researcher named Crnković discovered a chat program, Converso, that claimed to have extraordinary and novel end-to-end encryption features. The app was reverse engineered and analyzed, and the results were, frankly, hilarious. [The write-up of these failures is here](#).

Review the article. As you do so, you will likely encounter concepts we didn’t cover -- research these as well. Then, **answer the following**.

---

<sup>1</sup> A bit of trivia: Peter Gutmann is known for creating the [Gutmann Method of hard drive erasure](#), known for its excessive number of writing passes to counter what were considered hypothetical attacks at the time. Now, he’s saying we need to actually worry *less* about a possible attack. I thought that was interesting.

- (a) Identify a term or concept that was unfamiliar to you, and define it, citing sources as needed.

Forward Secrecy: Forward secrecy is a feature that ensures that even if long-term private keys are compromised, past session keys remain secure. This means that an attacker who gains access to the long-term keys cannot decrypt past communications encrypted with session keys derived during key exchanges that utilized perfect forward secrecy. (Source: Wikipedia)

- (b) The analysis shows several absolutely absurd weaknesses. Pick your favorite, describe it, and explain what the authors of Converso should have done differently.

Converso relies on a third-party authority—Seald's servers—as the sole certificate authority (CA) for mapping user identities to public keys. This centralization means that Seald's servers are the only source of truth for associating a user's identity (e.g., phone number) with their public encryption key. Because Seald controls the mapping of identities to public keys, they have the ability to impersonate any user by substituting a legitimate public key with a fraudulent one. Authors should reduce reliance on a single CA and explore decentralized approaches and use multiple, independent CAs to distribute trust.

- (c) Review the timeline at the end of the article. Timelines such as this are a common feature in vulnerability disclosures, as it shows how much notice was given to the vendor, how quickly they reacted, and what they did. In this case, the vendor initially tried to address the issues, but quickly switched to baseless legal threats before eventually pulling their app from the Apple and Google app stores entirely. How many days elapsed between the legal threats and them closing down their app entirely? If you find this number funny, feel free to append “lol” to your answer (optional).

May 11–12, 2023: The founder of Converso issued legal threats

May 16, 2023: The Converso apps appear to be no longer available for download from the Google Play Store or iOS App Store.

Five days, lol.

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

For the storage of passwords, Microsoft introduced [NTLM](#) hashing starting in Windows NT (the precursor of Windows 2000, XP, 7, 8, 10, and now 11). 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 F23: 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 may need to turn off "Safe browsing" in advanced settings or specifically visit the Downloads interface and choose "keep anyway", as it will default to blocking some of the tools downloaded below.

Lastly, when you run mimikatz, you may get a "Microsoft Defender SmartScreen" dialog with a "Don't run" button -- you can reveal a "Run anyway" button by clicking the "More info" link.

## 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: qwert12345

Test3: bny0rtd9\*waw-

## 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. To do this, we need to convert the hex to raw binary, then feed it to the MD4 algorithm. Research how to use “xxd” to convert hex into raw data, then pipe that into “openssl dgst -md4” to do the MD4 hash.

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

```
echo "73 00 69 00 6D 00 70 00 6C 00 65 00" > hex.txt
xxd -r -p hex.txt | openssl dgst -md4
MD4(stdin)= c51602d46e08e6fe02b5dc5c6439e538
```

This operation is common enough that it’s just called doing an “NTLM hash”, and [online tools exist for this](#).

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

*NOTE: If the mimikatz EXE disappears when you try to run it, the security software on your VM is still set to enforcing mode; post on Ed with your NetID and we'll get it set to permissive.*

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

```
RID : 000003ec (1004)
User : test1
Hash NTLM: c51602d46e08e6fe02b5dc5c6439e538

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
Random Value : c5da792af249e277a90b8aa3b11f710b

* Primary:Kerberos-Newer-Keys *
Default Salt : VCM-42420.WIN.DUKE.EDUtest1
Default Iterations : 4096
Credentials
aes256_hmac (4096) : f093dc21a7046040310051a61b02e020f0946bde3f741e71a58c365abad9d4fb
aes128_hmac (4096) : e860b29a4fefc6b9dde37a5e5563cab5
des_cbc_md5 (4096) : 377c7fd69ef48623

* Packages *
NTLM-Strong-NTOWF

* Primary:Kerberos *
Default Salt : VCM-42420.WIN.DUKE.EDUtest1
Credentials
des_cbc_md5 : 377c7fd69ef48623

RID : 000003ed (1005)
User : test2
Hash NTLM: ab8179ce3bc2c0c1176fdc40cd5f3456

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
Random Value : d9a662bc085f36603963906a3b118886

* Primary:Kerberos-Newer-Keys *
Default Salt : VCM-42420.WIN.DUKE.EDUtest2
Default Iterations : 4096
Credentials
aes256_hmac (4096) : b68668457a70aed2843000bd7173d8fcf73bca8f237108b9ecd43e91e9b23938
aes128_hmac (4096) : 89227105b4d559c9924e23acccf4c83a
des_cbc_md5 (4096) : fe9be9b998f74abc
```



```

* Primary:Kerberos *
  Default Salt : VCM-42420.WIN.DUKE.EDUtest2
  Credentials
    des_cbc_md5      : fe9be9b998f74abc

RID : 000003ee (1006)
User : test3
  Hash NTLM: cf560195a02331817e74af676ad49230

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
  Random Value : bdfa7382f4b0dfcd61469447915e91e5

* Primary:Kerberos-Newer-Keys *
  Default Salt : VCM-42420.WIN.DUKE.EDUtest3
  Default Iterations : 4096
  Credentials
    aes256_hmac      (4096) : 32ba29d0eed4fa40f3047a719e05c8ceea33f40499a83af53ceb96b927ff9136
    aes128_hmac      (4096) : 8ebb6132a2373f5eb4a1cbf4ec477abb
    des_cbc_md5      (4096) : 8fa731a83ed51564

* Packages *
  NTLM-Strong-NTOWF

* Primary:Kerberos *
  Default Salt : VCM-42420.WIN.DUKE.EDUtest3
  Credentials
    des_cbc_md5      : 8fa731a83ed51564

```

## 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
  ... (other stuff) ...

RID : 000001f5 (501)
User : Guest

RID : 000001f8 (504)
User : WDAGUtilityAccount
  Hash NTLM: 9a5c28fd8410c737d9b2c0f7f4ba4926
  ... (other stuff) ...

```

Reduce this to:

```

7ee17fdad80eef8865e6710064b9e686
9a5c28fd8410c737d9b2c0f7f4ba4926

```

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

```
grep -i "Hash NTLM:" hash.txt | sed 's/.*Hash NTLM:[ \t]*//'  
43ee9fc5c26d564b648ba8596ff5a383  
877511555b3d9521c1643b18c3c97302  
4c6623b63f6ad5f8090b838d89a7f796  
c51602d46e08e6fe02b5dc5c6439e538  
ab8179ce3bc2c0c1176fdc40cd5f3456  
cf560195a02331817e74af676ad49230
```

## Crack the hashes (1 point)

Take all the hashes and submit them all together to an online hash database such as [crackstation.net](https://crackstation.net). You should certainly be able to crack the test1 password ("simple").

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

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

```
43ee9fc5c26d564b648ba8596ff5a383  
877511555b3d9521c1643b18c3c97302  
4c6623b63f6ad5f8090b838d89a7f796  
c51602d46e08e6fe02b5dc5c6439e538  
ab8179ce3bc2c0c1176fdc40cd5f3456  
cf560195a02331817e74af676ad49230
```



**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
43ee9fc5c26d564b648ba8596ff5a383	Unknown	Not found.
877511555b3d9521c1643b18c3c97302	Unknown	Not found.
4c6623b63f6ad5f8090b838d89a7f796	Unknown	Not found.
c51602d46e08e6fe02b5dc5c6439e538	NTLM	simple
ab8179ce3bc2c0c1176fdc40cd5f3456	NTLM	qwerty12345
cf560195a02331817e74af676ad49230	Unknown	Not found.

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

2 passwords are cracked.

## Question 8: CVEs and Kaspersky Password Generator (6 points)

The Common Vulnerability and Exposure (CVE) system is a standardized database of vulnerabilities in commonly deployed software. Issues are assigned CVE numbers, allowing potentially dangerous software flaws to be identified in a unified way.

Who maintains CVEs? Where can they be found? (1pt)

CVEs are maintained by the MITRE Corporation, an American not-for-profit organization with dual headquarters in Bedford, Massachusetts, and McLean, Virginia. CVEs can be found on their website: <https://cve.mitre.org>.

Look up CVE-2020-27020 in the CVE database. The CVE database summarizes the issue briefly; it's designed to inform users when updates are needed. To get a full picture, also look up additional write-ups or articles about it so that you fully understand the issue.

**What software was affected? What is this software used for? (1pt)**

Kaspersky Password Manager was affected, the software is used for generating strong random passwords and managing all digital passwords.

**Briefly summarize the nature of the flaw. What was the specific cryptographic mistake? (2pt)**

The password generator feature was not completely cryptographically strong and potentially allowed an attacker to predict generated passwords, because the generator utilized the current system timestamp as the seed for its pseudo-random number generator when creating passwords. This is problematic because system timestamps are predictable, especially if an attacker knows or can estimate the approximate time when the password was generated.

**How does this vulnerability affect all three aspects of the CIA triad? Give an example of each. (2pt)**

- **Confidentiality:** The vulnerability compromises the confidentiality of passwords generated by KPM. Because the passwords are generated using a predictable seed (the system timestamp), attackers can potentially reproduce or predict these passwords. By exploiting the predictable seed, the attacker generates possible passwords and successfully predicts the user's banking password. As a result, the attacker gains unauthorized access to the user's bank account
- **Integrity:** An attacker gains access to a company's internal database by predicting the administrator's password generated by KPM. The attacker modifies the database records, alters transaction details, which compromises the integrity of the company's data.
- **Availability:** An attacker predicts a user's password for their cloud storage account. After logging in, the attacker changes the password and security settings, and thus locking the user out. An attacker can also hack into the website backend and made the website unavailable (cannot be accessed by the general public).

## **Question 9: 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.

**Show the SSH command used. (2pt)**

```
ssh -D 1080 -N -f fg96@ vcm-42506.vm.duke.edu
```

Paste a screenshot of your local browser visiting <https://ipchicken.com/> showing the IP address and hostname of your Kali VM. (1pt)



Show a terminal output from your local computer running a traceroute to duke.edu. (1pt)

The command for this differs on Mac, Windows, and Linux -- look it up.

traceroute to duke.edu (152.3.72.104), 64 hops max, 40 byte packets

```
1  * cdf-tel-200-j15-n3600-ipe-1_10_239_1_2_vlan3301.netcom.duke.edu
(10.239.1.2)  7.462 ms *
2  cdf-tel-200-j15-n3600-f-1_10_239_1_1_vlan3301.netcom.duke.edu
(10.239.1.1)  6.543 ms * *
3  * * *
4  cdf-tel-200-j15-n9500-sc-1_10_238_4_21_p20.netcom.duke.edu
(10.238.4.21)  16.518 ms  7.334 ms  7.249 ms
5  cdf-tel-200-n15-n3600-pc-1_10_238_4_78_p1.netcom.duke.edu
(10.238.4.78)  7.890 ms  7.498 ms  7.577 ms
6  dc-fitzeast-b220-d4-n7700-d-1_10_237_254_1_p1.netcom.duke.edu
(10.237.254.1)  7.692 ms  7.904 ms  7.545 ms
7  duke-web-fitz.oit.duke.edu (152.3.72.104)  6.793 ms  6.437 ms
6.807 ms
8  fitz-ltm-02-ex.oit.duke.edu (152.3.72.252)  2907.391 ms !H *
1005.030 ms !H
```

You will observe that the traceroute output is NOT affected by the SSH tunnel being connected, since it only proxies traffic configured to use the tunnel (like your browser).

Turn off the proxy setting in your browser and close the SSH session before proceeding.

## Question 10: 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 can include *all* network traffic generated by a client.

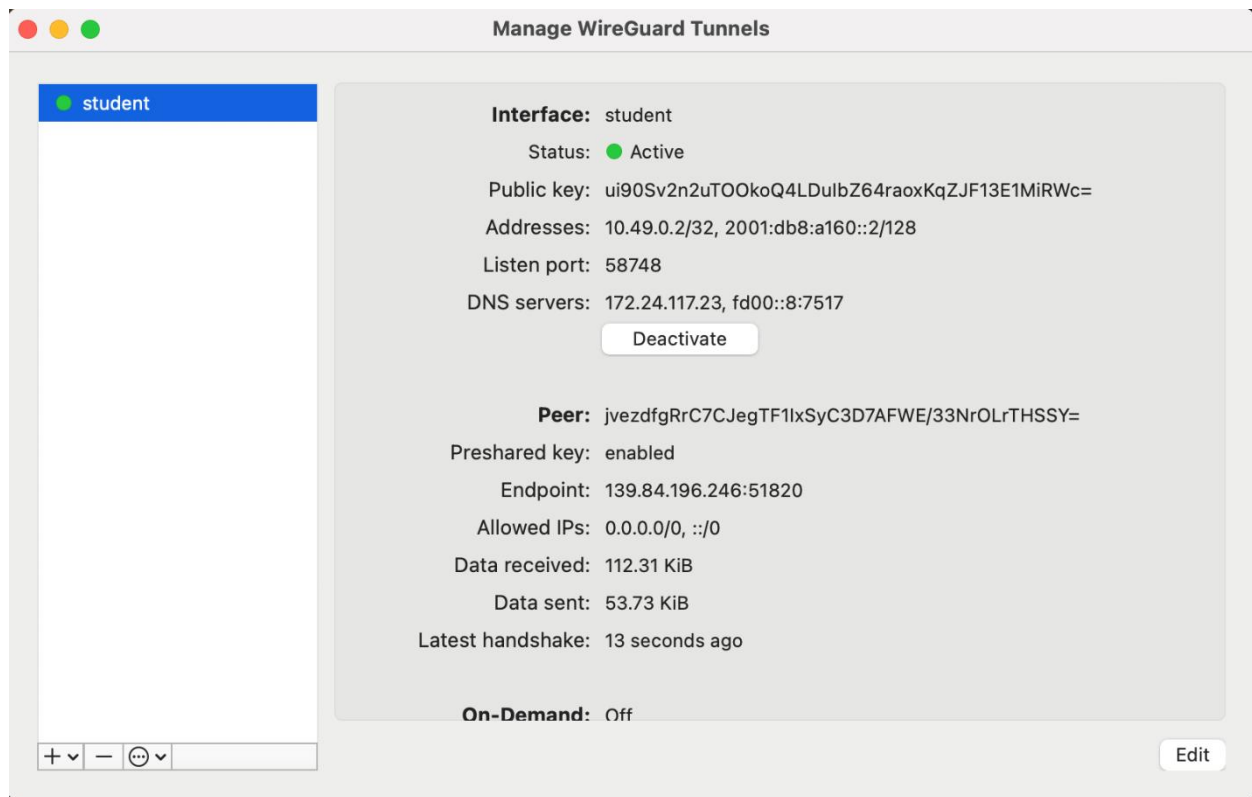
VPNs are very commonly deployed. However, there's a lot of cryptography setup and other configuration choices involved in setting up a VPN, and [it's easy to do it in a way that subtly compromises security](#).

[Algo](#) is a set of scripts to automate deployment of a WireGuard VPN server, either on your server or as a new VM on a cloud service. A WireGuard deployment has already been set up with Algo for your use -- [you can download the configuration file](#). You'll feed that config to the WireGuard client, available on a variety of platforms.

NOTE: The WireGuard config file requires Duke NetID login access to download. Do not share it publicly (e.g. by putting it into a public git repo, posting online, etc.)! Many unscrupulous people would love access to a VPN unconnected to their identity or finances to anonymize their illegal activities for free.

**Configure your personal computer to connect to the provided VPN.**

**Show a screenshot of your personal computer's VPN client software showing the successful connection. (1pt)**



Show a screenshot of a browser on your personal computer visiting an IP address identifier site like IPChicken.com. (1pt)



Show a terminal output from your local computer running a traceroute to duke.edu while the VPN is connected. (1pt)

```
traceroute to duke.edu (152.3.72.104), 64 hops max, 40 byte packets
 1  * 10.239.1.2 (10.239.1.2) 157.705 ms *
 2  10.239.1.1 (10.239.1.1) 45.267 ms * *
 3  * * *
```

```

4  10.238.4.21 (10.238.4.21)  18.505 ms  27.186 ms  11.056 ms
5  10.238.4.78 (10.238.4.78)   7.171 ms  7.089 ms  7.360 ms
6  10.237.254.1 (10.237.254.1)  7.332 ms  15.524 ms  7.602 ms
7  152.3.72.104 (152.3.72.104)  6.715 ms  33.443 ms  8.551 ms
8  * * *
9  * * 152.3.72.252 (152.3.72.252) 1060.645 ms !H
10 * * 152.3.72.252 (152.3.72.252) 1038.694 ms !H

```

**How does it differ from the output before the VPN was connected? Why is this? (2pt)**

It requires more time to reach to the duke internet network as it requires to go through VPN servers in between. The IP address shown also indicates that no hostname resolution is available for the internal hops anymore, which might indicate that internal DNS is not accessible now that traffic is encrypted and tunneled through the VPN.

**Databases of which IP addresses are in what physical locations have been developed; this technology is called IP Geolocation. Locate and use an IP Geolocation service to identify where this VPN server is deployed. (1pt)**

Victoria state, Australia

## Question 11: Tor Project: Anonymity Online (5 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. How is it similar to a regular VPN? How is it different? (3 points)**

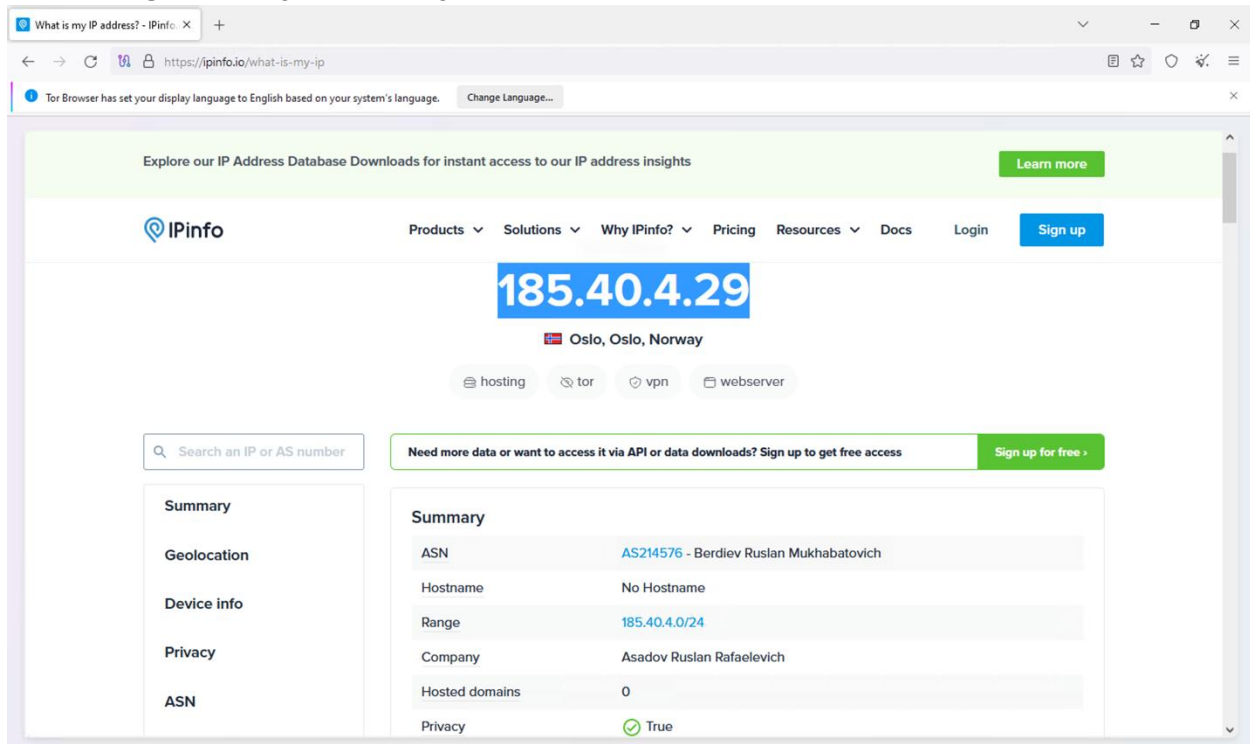
Tor is a free software that enable user to have private access to an uncensored internet. It was originally developed by the U.S. Naval Research Laboratory to protect online communications. The good purpose for this software is that it can protect as much privacy as possible and protect user's privacy from surveillance or data collection. The nefarious use of this software might be to obscure users' identities and locations so that criminals can use it to conduct illegal activities online without being monitored or followed easily. The tor and regular VPN are similar in terms of the encryption (both software encrypt your information transmitted online) and both tools are used to hide your IP address and anonymize your online identity. The difference might be concluded in two aspects. One is that tor uses multi-layered routing that sends your traffic through several nodes around the world before reaching its destination, while the VPN directs your traffic through a single encrypted tunnel to a VPN server, which then forwards it to the internet. In this case, the VPN server will know the information while for tor, no single node knows anything about the origin and the destination. Another aspect would be that tor is decentralized, anyone can set up a Tor node, and there is no central authority controlling the network, while usually VPN is managed by a central node (VPN server), making it easier to ensure speed but also means that you must trust the VPN provider.

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 address: 185.40.4.29

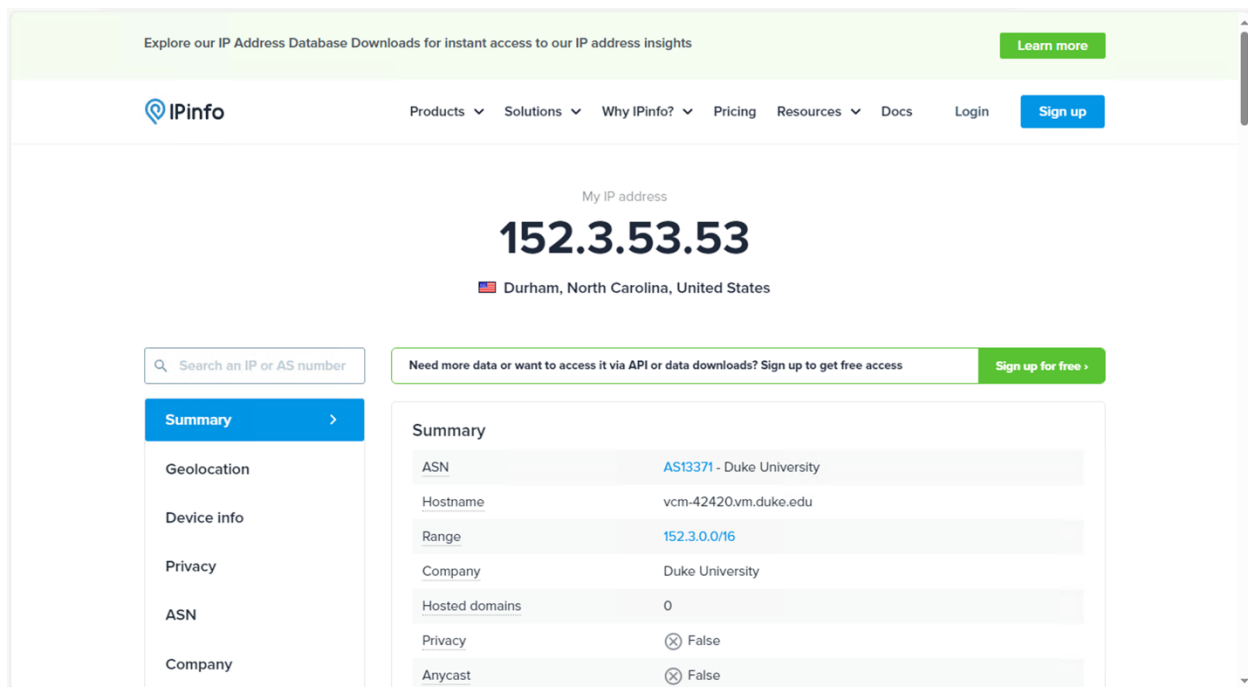
No hostname

**Use an IP Geolocation service in a Tor and non-Tor Browser. Show screenshots of both here. Geographically, where is your Tor exit node? (1 points)**



(image of using the tor browser)





Explore our IP Address Database Downloads for instant access to our IP address insights [Learn more](#)

IPinfo Products Solutions Why IPinfo? Pricing Resources Docs Login [Sign up](#)

My IP address

# 152.3.53.53

Durham, North Carolina, United States

[Need more data or want to access it via API or data downloads? Sign up to get free access](#) [Sign up for free](#)

- Summary
- Geolocation
- Device info
- Privacy
- ASN
- Company

**Summary**

ASN	AS13371 - Duke University
Hostname	vcm-42420.vm.duke.edu
Range	152.3.0.0/16
Company	Duke University
Hosted domains	0
Privacy	<input checked="" type="checkbox"/> False
Anycast	<input checked="" type="checkbox"/> False

(image of not using the tor browser)

Geographically, the exit node is located in Oslo, Norway.

## Question 12: Zero Trust (3 points)

We've learned about basic user access control, but the state of the art is more complex than that. Recently, support for so-called "Zero Trust" environments has grown, and much noise and confusion exists around this topic. However, in 2022, the U.S. federal government put forth new guidelines on moving toward Zero Trust that have clarified and simplified this goal.

[This article does a good job of explaining the modern take on Zero Trust networks.](#) Review it, **then answer the following**. You may wish to do additional research and reading as needed; as always, cite sources if you do so.

- (a) The article boils Zero Trust down to "don't give long-lived credentials to your users". What are examples of long-lived credentials? Why do we want to avoid them?

Password, especially those that are not changed frequently or do not expire, is a common example of long-lived credentials. Another one would be the SSH keys, which usually does not change frequently and is also used for a long time as a quicker way of authentication. Long-lived credentials are not frequently changed, provide attackers with more time to exploit them if they are compromised. And once compromised, an attacker can use the password or their SSH keys repeatedly until it's changed, or the account is locked. Also, for a practical reason, if the password is too long, user will forget it.

(b) What kinds of MFA does the article consider “bad”? Why? Which are “good”? Why?

Bad MFA includes 1) SMS and 2) phone-based MFA, these MFA are vulnerable to phishing and sim-swapping attacks. The author also consider one-time code like TOTP protocol (Google Authenticator), because it still rely on user to manually input the data, which still can be phished by attackers if the user is tricked into providing the code on a fraudulent site providing that the code is used without expiring.

Good MFA refers to “phishing resistant tokens”, like PIV cards and yubikeys. Attackers cannot simply intercept a token or key like they can with SMS or one-time codes, making these solutions much more secure. Author also discussed passwordless access is the future, which reduces the risk of password-related vulnerabilities.

(c) What is perimeter defense, and how are VPNs linked with this concept? Why is perimeter defense considered insufficient?

Historically, organizations have relied on the perimeter defense model, creating a digital "fortress" around their network resources. Within this fortress, everything was considered safe, while everything outside was deemed a threat. (Source:

<https://www.twingate.com/blog/zero-trust-alternative-to-perimeter-defense-and-vpns>)

Typical technique includes firewalls, Intrusion Detection/Prevention Systems (IDS/IPS), and VPN plays a key role in perimeter defense by extending the secure, trusted network to external users, so that the external user can use VPN to access the internal network as if they were physically present inside the organization's network.

The reason that perimeter defense is considered insufficient is that perimeter defense assumes that all users and devices inside the network are trusted, which fails to account for insider threats, and like VPN, if there is a way for attackers to bypass the defense, they can access the internal resources and (Source:

<https://www.pomerium.com/blog/the-perimeter-problem>).

### Question 13: Bitlocker, FileVault, and LUKS (3 points)

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

Microsoft BitLocker is a full-volume encryption feature included with Windows operating systems. It is designed to protect data by providing encryption for entire volumes or drives. (Source: Wikipedia). BitLocker uses the Advanced Encryption Standard (AES) algorithm for encryption, and supports both 128-bit and 256-bit key lengths. The mode of operation includes (1) Cipher Block Chaining (CBC) mode and (2) ciphertext stealing (Source: <https://learn.microsoft.com/en-us/mem/configmgr/protect/tech-ref/bitlocker/settings>). By default, BitLocker uses XTS-AES 128-bit encryption for automatic encryption, on Windows 10, BitLocker

supports (1) AES-CBC 128-bit (2) AES-CBC 256-bit (3) XTS-AES 128-bit and (4) XTS-AES 256-bit.

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

Mac computers offer FileVault, a built-in encryption capability, to secure all data at rest. FileVault uses the AES-XTS (XEX-based Tweaked codebook mode with ciphertext Stealing) data encryption algorithm, so mode of operation is XTS, to protect full volumes on internal and removable storage devices. (Source: <https://support.apple.com/guide/deployment/intro-to-filevault-dep82064ec40/web>).

**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 Unified Key Setup (LUKS) is a disk encryption specification widely used in Linux systems for full-disk encryption. LUKS supports the following encryption modes: (1) ECB (2) CBC-PLAIN64 (3) CBC-ESSIV:hash and (4) XTS-PLAIN64. Various Linux distributions may use different default settings when setting up an encrypted volume. The common use one is XTS-PLAIN64 (Source: <https://blog.elcomsoft.com/2020/08/breaking-luks-encryption/>).

## Question 14: SSL certificate management (10 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.

In this scenario, three machines will be involved:

- The **Certificate Authority (CA)** will be played by the **Kali VM**. It will generate a root certificate and sign the web server's certificate using openssl.
- The **web server** will be played by the **Linux VM**. It will generate the certificate signing request using openssl, receive the signed certificate, and host an apache web server with that certificate using HTTPS.
- The **web client** will be played by the **Windows VM**. It will have the Kali VM's root certificate installed into its trust store.

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

1. **Kali VM (CA):** Generate a new RSA key pair. In doing so, use an option that encrypts it with AES and use a strong passphrase. This way, even if the private key leaks, it cannot be used without the passphrase.

```
> openssl genrsa -aes256 -out ca_key.pem 4096
```

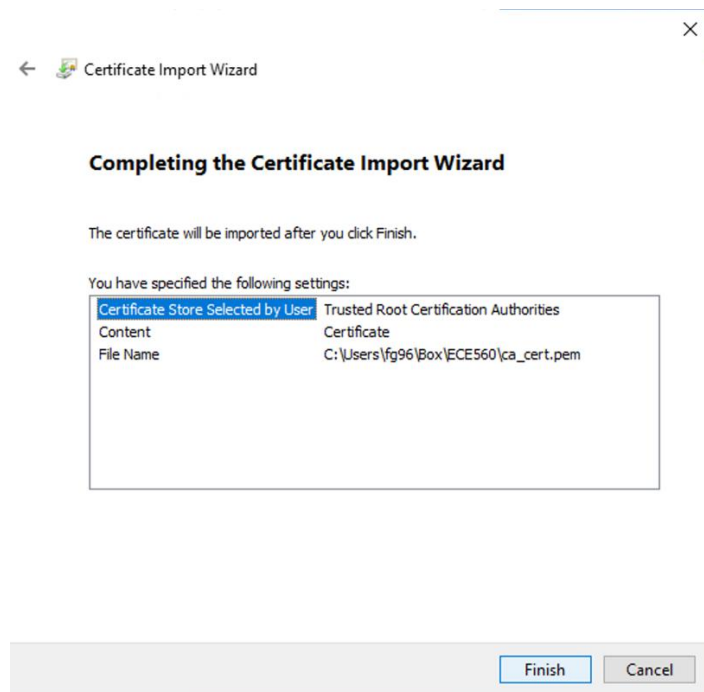
```
Enter PEM pass phrase: GFc1029384756#
Verifying - Enter PEM pass phrase: *****
```

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

```
> openssl req -x509 -new -key ca_key.pem -sha256 -days 3650 -out
ca_cert.pem -subj "/CN=fg96 Root CA"
Enter pass phrase for ca_key.pem:*****
```

3. **Windows VM (Client):** Install the certificate into the trust store of your Windows VM. Now the Windows VM will trust any certificates signed by the Kali VM.

**Trusted Root Certification Authorities -> Certificates.**



4. **Linux VM (Server):** Create a key pair, and from it, a certificate signing request (CSR) on your Linux VM (not the Kali VM!). Set the common name to the domain name of your Linux VM (e.g. vcm-5341.vm.duke.edu). Be sure to enable the Subject Alternative Name (SAN) extension, required for modern browsers (this trips up a lot of students -- don't skip it!). Once done, transfer the CSR to the Kali VM.

```
> openssl genrsa -out server_key.pem 2048
```

```
> openssl req -new -key server_key.pem -out server.csr -subj
"/CN= vcm-42411.vm.duke.edu" -reqexts SAN -config <(cat
/etc/ssl/openssl.cnf <(printf "[SAN]\nsubjectAltName=DNS:vcm-
42411.vm.duke.edu"))
```

```
> scp server.csr fg96@vcm-42506.vm.duke.edu:~/
```

5. **Kali VM (CA):** Sign the CSR on the Kali VM using the key pair created in step 1, thus creating a signed certificate for your Linux VM. Transfer the certificate to the Linux VM.

```
> openssl x509 -req -in server.csr -CA ca_cert.pem -CAkey  
ca_key.pem -CAcreateserial -out server_cert.pem -days 365 -sha256  
-extensions SAN -extfile <(printf "[SAN]\n  
subjectAltName=DNS:vcm-42411.vm.duke.edu")  
Certificate request self-signature ok  
subject=CN= vcm-42411.vm.duke.edu  
Enter pass phrase for ca_key.pem:*****  
  
> scp server_cert.pem fg96@vcm-42411.vm.duke.edu:~/
```

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

```
> sudo apt update  
> sudo apt install apache2 -y  
> sudo a2enmod ssl  
> sudo systemctl restart apache2
```

7. **Linux VM (Server):** Configure HTTPS to use the certificate you created.

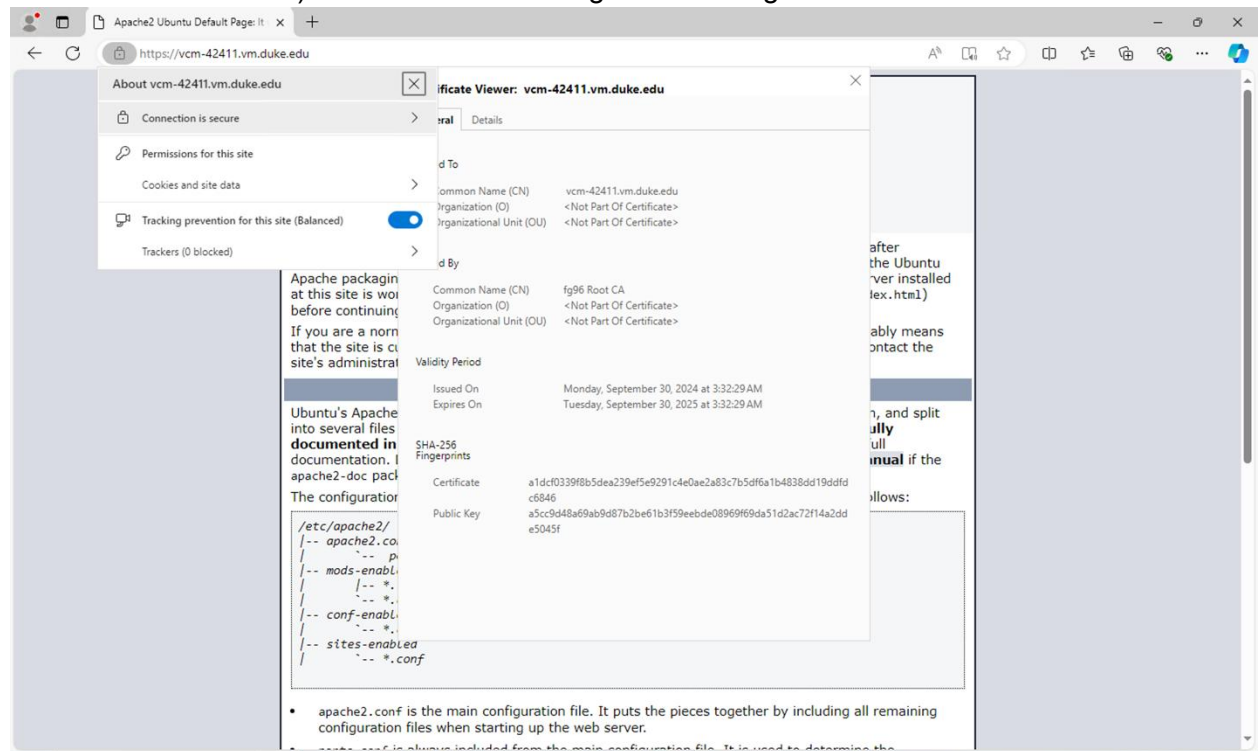
```
> sudo cp server_cert.pem /etc/ssl/certs/server_cert.pem  
> sudo cp server_key.pem /etc/ssl/private/server_key.pem  
> sudo nano /etc/apache2/sites-available/default-ssl.conf
```

```
SSLCertificateFile /etc/ssl/certs/server-cert.pem  
SSLCertificateKeyFile /etc/ssl/private/server-key.pem
```

```
> sudo apache2ctl configtest  
Syntax OK  
> sudo a2ensite default-ssl  
> sudo systemctl reload apache2
```

8. **Windows VM (Client):** 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). No certificate warnings should be generated.



Congratulations, you have mastered certificates!

## Question 15: TLS in Depth (7 points)

In class, we covered a high level overview of how encryption happens on the web via HTTPS, which is HTTP over Transport Layer Security (TLS). This website gives an excellent deep dive on the topic by showing and annotating every single byte of a real TLS handshake:

<https://tls12.xargs.org/>

Read through this site, expanding the annotations as needed. Using this info and additional research, **answer the following questions**.

- a. What cipher suite does the *client* \*most\* prefer? How do you know?

TLS\_ECDHE\_RSA\_WITH\_CHACHA20\_POLY1305\_SHA256. The client provides an ordered list of which cryptographic methods it will support for key exchange, and the list is in the order preferred by the client, with highest preference first.

- b. The Server Name Indication (SNI) field is mentioned; research this. Why does the client transmit the hostname of the server it is talking to?  
SNI enables multiple HTTPS websites to be served from the same IP address. Without

SNI, each secure website would require its own dedicated IP address, which is inefficient given the scarcity of IPv4 addresses (Source: <https://comodosslstore.com/resources/what-is-sni-or-server-name-indication/>).

- c. What cipher suite did the *server* choose? How do you know?  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA, server hello contains cipher suite (0c 13) which means the server has selected cipher suite 0xC013 from the list client provided.
- d. What is the *length* of the server certificate provided? How do you know?  
The length of all certificate data is 808 bytes, the length of the first (and only) certificate is 805 bytes. The info is included in server certificate sent from the server.
- e. What are the four pieces of information used during Client Encryption Key Calculation? What is the *client write key* (the key used to encrypt content sent by the client)? How do you know?
  - 1. server random
  - 2. client random
  - 3. server public key
  - 4. client private key

write key: f656d037b173ef3e11169f27231a84b6

It is calculated by first calculate the PreMasterSecret from the server public key and client private key. Then they calculate the MasterSecret by using client random, server random and PreMasterSecret. Then they will calculate the final encryption keys using a key expansion, where the client key will be the string starting from the calculated key 40 byte to 56 byte.

- f. What is the purpose of the Client Handshake?  
The purpose of the client handshake is to (1) send available cipher suite and discuss TLS versions (2) provide random data for PreMasterSecret calculation in order to calculate the final session key (3) facilitate key exchange. The main purpose is to provide the information above so that they establish secure communication afterwards.
- g. What is the actual payload sent and received in the Client Application Data and Server Application Data stages?

The actual payload sent from client is 17 03 03 00 30 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 6c 42 1c 71 c4 2b 18 3b fa 06 19 5d 13 3d 0a 09 d0 0f c7 cb 4e 0f 5d 1c da 59 d1 47 ec 79 0c 99, which includes a header, an initialization vector and an encrypted data that can be decrypted using the initialization vector and the client write key, the encrypted data contains a message authentication code (MAC) and padding it is

larger than the decrypted data. The decrypted data is “ping”.

The actual payload sent from server is 17 03 03 00 30 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 97 83 48 8a f5 fa 20 bf 7a 2e f6 9d eb b5 34 db 9f b0 7a 8c 27 21 de e5 40 9f 77 af 0c 3d de 56, which includes a header, an initialization vector and an encrypted data that can be decrypted using the initialization vector and the server write key. the encrypted data contains a message authentication code (MAC) and padding it is larger than the decrypted data. The decrypted data is “pong”.

**Extra credit:** In 2013, Google introduced the QUIC protocol as an alternative to traditional TCP-based TLS. Chrome made good use of it with Google’s own servers, but broader adoption was slower, with Firefox only getting support in 2021. Eventually, it formed the underpinning of HTTP/3. This UDP protocol makes several tweaks to increase network performance.

For up to 7 points of extra credit, answer the same questions for QUIC as you did for TLS, using this site: <https://quic.xargs.org/>

Note: some concepts don’t map perfectly one-to-one. Use your best judgment to find the aspects of QUIC that are most analogous to those in traditional TLS.

a. What cipher suite does the *client* \*most\* prefer? How do you know?

The client provides an ordered list of cipher suites in ClientHello, which is included in the Client initial packet. Just like the TLS handshake, the list is in the order preferred by the client, with highest preference first.

b. The Server Name Indication (SNI) field is mentioned; research this. Why does the client transmit the hostname of the server it is talking to?

Similar to TLS, in ClientHello, there is an extension of server name which includes SNI, Without this extension a HTTPS server would not be able to provide service for multiple hostnames (virtual hosts) on a single IP address.

c. What cipher suite did the *server* choose? How do you know?

When server finish the calculation of the key, it will send back a server initial packet containing a ServerHello, which includes the server’s selected cipher suite.

d. What is the *length* of the server certificate provided? How do you know?

After the ServerHello packet, the server follows up with a "Handshake" packet containing one or more certificate, and the Certificates Length (total length of certificates) and Certificate Length (the length of the first certificate).



e. What are the four pieces of information used during Client Encryption Key Calculation? What is the *client write key* (the key used to encrypt content sent by the client)? How do you know?

Client uses actually *three* information to calculate Client Encryption Key.

1. Server public key from ServerHello
2. Client public key
3. SHA256 hash of ClientHello and ServerHello

The write key is called client application key. It looks like this:

e010a295f0c2864f186b2a7e8fdc9ed7

f. What is the purpose of the Client Handshake?

QUIC's advancement is that it combines the TCP and TLS handshakes and it establishes a secure and authenticated communication channel between the client and the server.

g. What is the actual payload sent and received in the Client Application Data and Server Application Data stages?

The client sends its first post-handshake packet, containing stream data with the contents "ping". The server sends its first post-handshake packet, containing stream data with the reply "pong". Both are encrypted using the application key calculated above.