# NETSEC VT2022

## Practical Laboratory

## Lab #2

*23 February 2022*

**Group 40:**

*Jawdat Kour (jako0237)*

*Anas Kwefati (ankw6718)*

*Munish Sharma (mush5681)*

**Department of Computer and Systems Sciences**

**Spring 2022**

# Traffic Interception in Local Networks

### Question 1

**Check the common protocols that we use on the internet (or parts thereof), i.e. HTTP, HTTPS, ftp, telnet, ssh, pop3, imap, smtp, Which of these are in clear text and which support some form of encryption.**
**If you check when these protocols were defined/put into use can you see a trend developing over time with respect to cryptography?**

HTTP is a plain text protocol while HTTPS uses SSL/TLS (Transport Layer Security) to encrypt HTTP traffic. FTP has no encryption support, however some variations such as SFTP uses SSH encrypted tunnel. Telnet traffic is sent and received in a plain text, whereas the secure alternative SSH employs encryption algorithms for authentication and encryption. Both POP3 and IMAP have no security function to encrypt data but they can be used over SSL/TLS. SMTP in its turn lacks encryption when transferring emails.

Most of the previous protocols rely on SSL/TLS to secure their traffic. The first version of SSL was not published because of security flaws, the next version SSL 2.0 was developed in 1995 and specified by RFC 6176 but it had a security issue as well. However, some of the internet protocols started at that time to employ SSL such as SSH which was developed in 1995 but defined by RFC 4250 in 2006. HTTPS was defined by RFC 2818 after some time in 2000. We summarise that the employment of cryptography in the Internet Protocol officially began around the year 2000 [1],[2],[3].

# Symmetric Encryption – Block Ciphers

### Question 2 (optional)

**If you have an email client that connects to a smtp-server directly (i.e. not a web mail service) try and capture the traffic with the smtp-server as you're sending an email (using Wireshark). Were you successful, why/why not?**

Since we don't have a client application on a machine that is connected directly to a mail server, it was not possible to capture the traffic.

## Question 3

**“What can be considered as the secret key of the ROT13 cipher? Does this cipher use substitution or transposition as encryption technique? Based only on the intercepted known ciphertext, is the attacker capable of retrieving the plaintext and if yes, how?”**

Rot13 is a substitution cipher. It “replaces a letter with the 13th letter after it in the alphabet”, the secret key is (13) since the shift is 13 places, due to the reciprocal substituting alphabet used by Rot13, the decryption and decryption algorithms are the same. Rot13 (Rot13(x)) = x. Decryption can be done by replacing every character again with the 13th letter after in the 26 loop of the English alphabet [4].

We can achieve Rot13 encryption using a command `tr` on linux. The next command line tells to take a file message.txt as an input, replace the lowercase letters `a-m` with `n-z`, and `n-z` with `a-m`, the same things for the uppercase letters (with the 13th letter after), and output the cipher text into the file message_rot13.txt

```
$ cat message.txt | tr a-mn-zA-MN-Z n-za-mN-ZA-M >
message_rot13.txt
```

## Question 4

**So use symmetric encryption and encrypt the file containing only the phrase "This is a secret message" (w/o the quote marks) with the crypto AES256 and using "ASCII armour" to make the output printable. Use the key "secret" (again w/o quote marks). If your input file is named secret.txt you should end up with a file secret.txt.asc**
**Include the full encrypted PGP message block in your report.**

The next command will generate a ciphertext from the text file `secret.txt` using `AES256` symmetric algorithm, and it will prompt you to enter the secret key to encrypt the file which is `secret`.

```
# gpg --cipher-algo AES256 --symmetric -a secret.txt
```

The output file `secret.txt.asc` contains the following encrypted PGP message block

```
-----BEGIN PGP MESSAGE-----
jA0ECQMCUQUXz62TOAXk0lIBeeAqxphi7c7qXmVaohKzzAWyvX4dp8wGFKisCAvv
tIewQTvCcMNiQEr9KJ3Jfao3Qz27HmBC2XSw4C2NsuUm+jsbx4sV0AjpelFSkcqv
uzl7
=KaaI
-----END PGP MESSAGE-----
```

Encrypting the same file `secret.txt` with the same command gave a different output message block.

## Question 5

**With the encryption above, what is the size of the generated key? (Both to the algorithm and the effective key length?) Can you identify any security problems with the above approach?**

AES256 is a symmetric block cipher, the block size is 128 bit length. The key length is 256 bits and that makes it very secure and brute forcing is infeasible in a short time. However, many attacks against AES variations have been reported but that didn't affect its reputation. Usually the hacker uses a side-channel attack against AES to get leaked information about keys for example. Sharing the secret key between two parties is another problem, an asymmetric approach should be used for secret key transmission [5].

## Message Integrity

### Question 6

**What are the hash values of the file? What are the hash algorithms' output sizes? Briefly explain why a larger output size of a hash algorithm is preferable." Note that most/all editors/operating systems add line-feed characters to a file. And these are different for different operating systems The file I encrypted contained a single carriage return character (ASCII CR) at the end.**

To solve this problem, Ubuntu 20.04 is used, where sha256sum command works to get the sum of the file. I have created a new file secret.txt with text "This is a secret message" as directed in Question 4. After executing the sha256sum command, the hash sum found was – 47985b79593f1df4031e333bc1d1f83906d9ff5493ece9549e5020b26642e46b. There are a number of sizes when it comes to SHA for SHA 0-3 variants. From the security perspective of data, a bigger hash output gives a bigger challenge to an adversary to crack it and plain text will be surely defended against dictionary attacks[6].

In order to solve the second part of the question, I sent secret.txt to my third partner in this assignment. My partner checked the secret messages hash sum on his system on receiving, which was identical to mine. Hence, the message was delivered with full integrity. To check if the hash sum is different from the original hash, I changed the message to "this is a secret message" and this time a completely different sum was generated.

### Question 7

**Do the checksums match after the change? How different are the checksums? Also, is the protocol you're using (i.e. sending the checksum in clear text with the message) secure? Why/why not?**

No, there is a significant difference between the original message and altered messages hash sums. No, I believe sending the hash of text in plain form is not secure as if a man in the middle happens, both the text and hash associated with it could be easily modified. Moreover, this modification can go undetected, as the receiver might just compare the checksum and feel secure about it.

## Public Key Cryptography

### Question 8

**What key-size and key-type did you choose? I.e. what type of public key cryptography algorithms. Why?**

I have generated my GPG key pair using the RSA algorithm, using a key size of 2048 bits. The reason for using RSA as it is asymmetric, as it uses a public and private key. So someone would be able to use the public in order to send me encrypted messages. Having a key of 2048 bits, makes it harder for anyone to be able to crack the encrypted messages if they don't have a powerful computer. Also when it comes for signing, RSA allows to create larger hashes than DSA. It also gives faster signature checking compared to DSA. And it is a widely used encryption algorithm.

### Question 9

**Are there any security issues with the secret key file you generated? Why/why not?**

The issue with the secret key file, is if someone manages to steal it, and get access to it, then, this person would be able to decrypt any encrypted files related to this key pair. The person would have access to the private key, and therefore, every file encrypted using the public key would be decrypted thanks to the private key. Another issue is when creating this file, we had to type a passphrase, and so if we forget the password we would not be able to use the private key to decrypt messages coming to us. Furthermore, it is the weakest point, as if we don't put a strong password, then someone would be able to brute force it. So a good password here is important [7].

### Question 10

**How did you make sure that Alice really used Bob's public key? If you cannot check Bob's public key, how could someone that has complete control of the communication channel between Alice and Bob perform a man-in-the-middle attack?**

There is no certain way Alice knows about the authenticity of Bob's public key if there is not a trusted third party involved. The alternative, could be Bob delivers his public key personally to Alice personally which is not practical in real life scenario. In case if Alice can not check the authenticity of Bob's key, then a third person, we say named X, can pretend to be Bob by providing a public key with Bob's name/identifiers attached to it. As a result, Alice will be sharing information with X, assuming she is securely communicating with Bob over a public-key cryptographic solution [8].

In real world scenario, public key certificates are used to verify the identity of public keys. So in this case, Alice would not accept X's key since it might not have a public key certificate, identifying that public key belongs to Bob [9].

## Question 11

**You've only done encryption so far. Alice can be certain that only Bob can read the message (if precautions against man-in-the-middle attacks have been taken), but can Bob be certain that the message comes from Alice? Why/why not?**

No, Bob cannot be certain that the message comes from Alice at 100%, except if he has her public key, and assuming she has digitally signed her message. Then Bob can be sure it is her. In other words, this could be the same procedure as we did in Question 6-7.

But to make it more secure, Alice must hash the message as well as encrypt the message, encrypt the hash sum with Bob's public key so that only Bob can decrypt it. On receiving the message, Bob will be decrypting the message, hashing it and then decrypt the hash sum that came along from Alice. If the both hash sums match, only then it can be assumed that the message received is authentic [10].

## Public Key Cryptography – Digital Signatures

### Question 12

**Does Charlie have to be present when Alice and Bob communicate with each other? Do they have to communicate with Charlie beyond having trusted his key to sign for other keys? Why/why not? Were there any problems with your understanding of what was going on, and how the web of trust works? What/why? Do you have any suggestions for improvements in that regard?**

No, Charlie does not have to be present when Alice and Bob communicate with each other. As Charlie has signed both Bob and Alice's keys, and he introduced his key to both of them, then they do not need to communicate with Charlie beyond having trusted his key, as they already know who Charlie is, and so they can trust him. No, we did not have any issues understanding the concept behind the web of trust, so what we understand is that if A knows B and B knows C then A and C can talk to each other securely. There has been a lot of improvement in this context, but a tiny contribution of suggestion could be that public keys must not be listed openly, instead they should be shared by participants via some other means like physical meeting etc.

# References


[1]"Transport Layer Security - Wikipedia", *En.wikipedia.org*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security. [Accessed: 25- Feb- 2022].

[2]"Secure Shell - Wikipedia", *En.wikipedia.org*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Secure_Shell. [Accessed: 25- Feb- 2022].

[3]"HTTPS - Wikipedia", *En.wikipedia.org*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/HTTPS. [Accessed: 25- Feb- 2022].

[4]"ROT13 - Wikipedia", *En.wikipedia.org*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/ROT13. [Accessed: 25- Feb- 2022].

[5]"Advanced Encryption Standard - Wikipedia", *En.wikipedia.org*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. [Accessed: 25- Feb- 2022].

[6] "Secure Hash Algorithms - Wikipedia", *En.wikipedia.org*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Secure_Hash_Algorithms. [Accessed: 01- Mar- 2022].

[7]"Getting Started", *Gnupg.org*, 2022. [Online]. Available: https://www.gnupg.org/gph/en/manual/c14.html#AEN25. [Accessed: 01- Mar- 2022].

[8] "IBM Docs", Ibm.com, 2022. [Online]. Available: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=processes-public-key-cryptography. [Accessed: 01- Mar- 2022].

[9] "IBM Docs", Ibm.com, 2022. [Online]. Available: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=processes-public-key-certificates. [Accessed: 01- Mar- 2022].

[10] "IBM Docs", Ibm.com, 2022. [Online]. Available: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=cp-cryptographic-hash-functions-message-authentication-codes-digital-signatures#cryptographichashetc__digitalsignatures. [Accessed: 01- Mar- 2022].