



# Network Security

# CS3403

## Lab 4

### Encrypted data over TCP Socket

**Note:** Mac users can do WSL experiments on their own machines and work with other classmates for Windows related activities or

If you have any VM installed try the same between Mac and the VM.

Can also try connecting from one machine to the other which are on the same subnet.

**Mouli Sankaran**

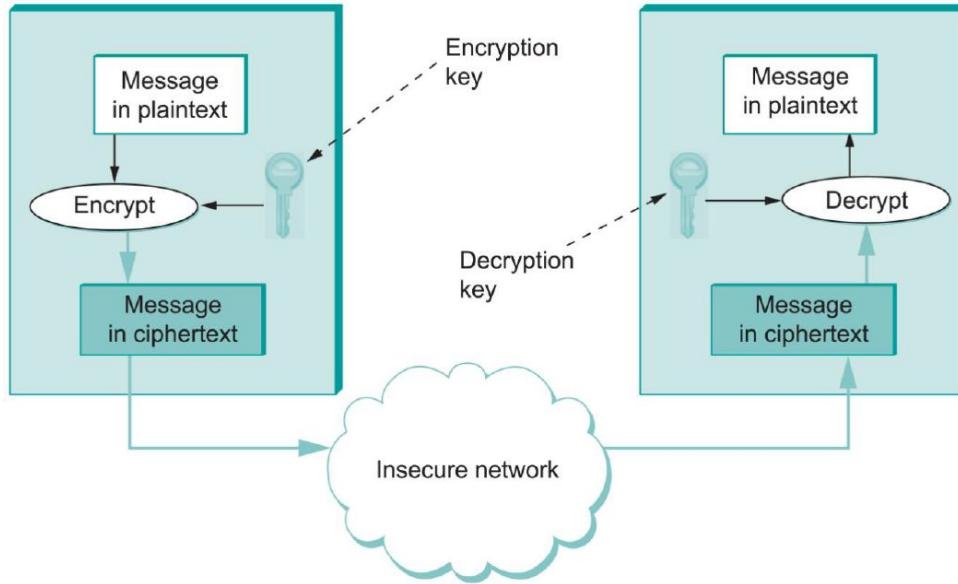
# Lab 4: Focus

- **Symmetric Key Ciphers**
  - OpenSSL Introduction
  - Library Installations on WSL and Windows
  - **Exp 1:** Server in C - WSL – Client in Python – Win
- **Asymmetric Key Ciphers**
  - Explanation about the **pem** format
  - **Exp 2:** Server in C - WSL – Client in Python – Win



# Symmetric Key Ciphers (secret-key)

# Secret-key Encryption and Decryption



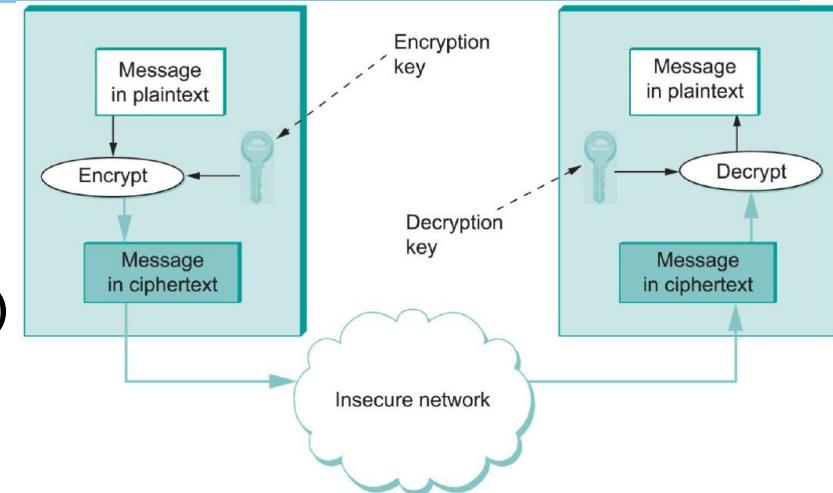
**Plain text** is the original, unencrypted information or data.

**Cipher text** is the result of applying an encryption algorithm to the plain text; it appears scrambled or unreadable without the appropriate decryption key.

- The **transformation** represented by an **encryption** function and its corresponding **decryption** function is called a **cipher**.
- Encryption and decryption functions have to be parameterized by a key and the functions are considered to be public knowledge—only the key needs to be secret.
- The **ciphertext** produced for a given **plaintext** message depends on both the **encryption function** and the **key**.

# Secret-key Encryption and Decryption

- Here, **both participants** in a communication **share the same key**.
- Secret-key ciphers** are also known as **symmetric-key** ciphers.
- Advanced Encryption Standard (**AES**) standard issued by NIST.

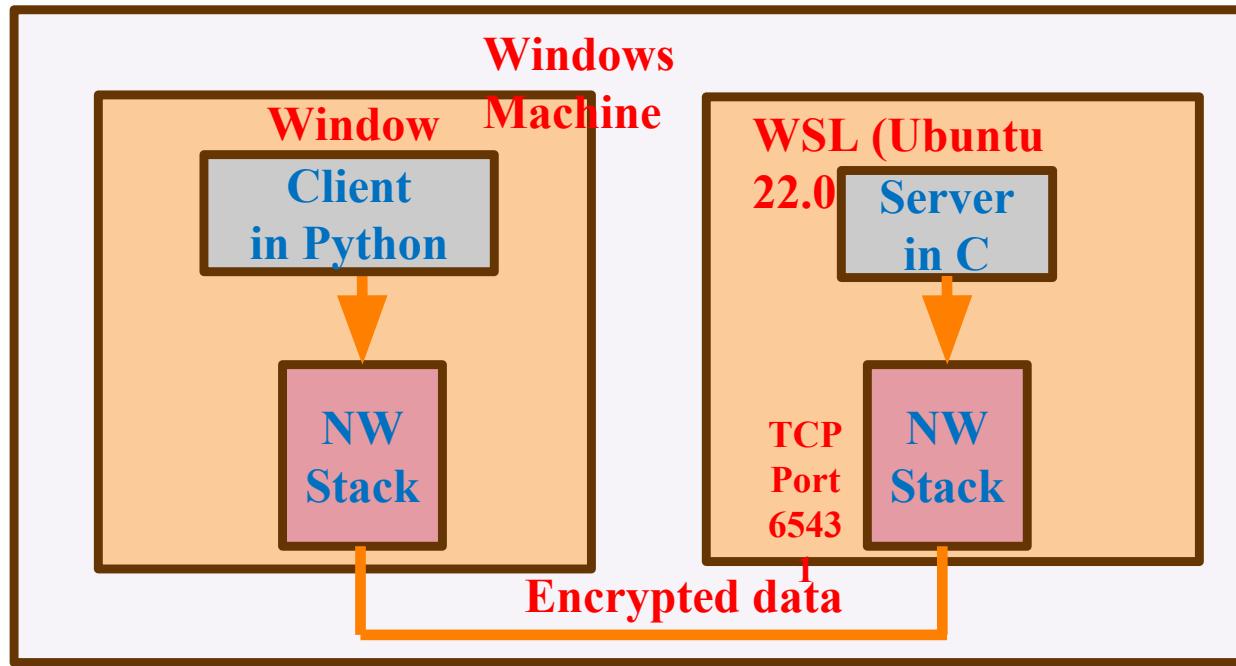


- AES supports key lengths of 128, 192, or 256 bits, and the block length is 128 bits. AES permits fast implementations in both software and hardware.
- It does not require much memory, which makes it suitable for small mobile devices.
- It is used to securely encrypt data in fixed-size blocks (often 128 bits).
- AES has some mathematically proven security properties and, has not suffered from any significant successful attacks.



## Exp 1 – WSL Server – Win Client (symmetric key)

# Exp 1: System Diagram: Symmetric keys (code shared)



## Server code:

```
// For AES-128, key and IV are 16 bytes.  
unsigned char key[16] = "0123456789abcdef";  
unsigned char iv[16] = "abcdef9876543210";
```

## Client code:

```
# Same key and IV as in Server written in C  
key = b"0123456789abcdef"  
iv = b"abcdef9876543210"
```

**IV: Initialization  
Vector**

# Open SSL: An Introduction

[Ref: OpenSSL](#)

- **OpenSSL** is an open-source software library that implements the Secure Sockets Layer (**SSL**) and Transport Layer Security (**TLS**) protocols
  - OpenSSL enables secure communication between clients and servers.
  - It's used by many web servers, including those that use HTTPS.
  - It generates and manages certificates that verify the identity of websites and servers
  - It encrypts data using symmetric/asymmetric encryption algorithms
  - It implements basic cryptographic functions and provides various utility functions
- **SSL** was the first widely adopted protocol to secure Internet, developed by Netscape in mid-1990s.
  - Due to vulnerabilities and design issues, SSL has been largely phased out in favor of its successor TLS
- **TLS** offers stronger cryptographic algorithms and better security practices
  - TLS is used for securing web traffic (HTTPS), email, VoIP, and many other applications.

# Pre-requisite: Library Installations

- **On WSL (Ubuntu):** using the Terminal:

- For running the Server program (in C) on WSL do the following:
  - ***sudo apt-get update***
  - 
  - **Building the server executable on WSL**
  - ***gcc Lab4\_Exp1\_Sym\_Server\_WSL.c -lssl -lcrypto -o sym\_server -Wno-deprecated-declarations***
  - **-l** links dynamic (shared object) libraries libssl.so and libcrypto.so.
  - **-W** for suppressing deprecated declarations while compiling

- **On Windows:** using the Command Prompt:

- For running the Client program (in Python) on Windows do the following:
  - ***pip install cryptography***
  - **Compiling and running the client on Windows**
  - ***python Lab4\_Exp1\_Sym\_Client\_Win.py***

# Exp 1: Execution and output

*gcc Lab3\_Exp1\_Sym\_Server\_WSL.c -lssl -lcrypto -o sym\_server*

smouli@Mouli: ~/NetSec/Lab

```
smouli:Lab3$ ./sym_server
Symmetric server listening on port 65431...
Decrypted message from client: Hello from symmetric client
smouli:Lab3$ |
```

Build the server executable on WSL

Run the Server 1st on WSL

*pip install cryptography*      Install Python cryptography package on Windows

```
G:\Users\Mouli\RVU\Courses\NetSec\PMaterials\Labs\Lab3\code>python Lab3_Exp1_Sym_Client_Win.py
Received from server: Hello from symmetric server
▼ Data (32 bytes)
  Data: 0f7e0c009fd3a915bd4740405dd1a277145fab962818f65bffb7e2b4b9a18fff
  [Length: 32]
```

0000	00 15 5d 19 e6 63 00 15 5d bf 1f f3 08 00 45 00	... ]...c... ].....E.
0010	00 48 b0 cf 40 00 80 06 00 00 ac 18 d0 01 ac 18	-H-@... -----
0020	d7 31 f6 0b ff 97 de 26 6a d9 13 b8 3d 88 50 18	-1.....& j....=P.
0030	00 ff ff 9e 00 00 0f 7e 0c 00 9f d3 a9 15 bd 47	.....~ .....G
0040	40 40 5d d1 a2 77 14 5f ab 96 28 18 f6 5b ff b7	@@]...w_- ...([...
0050	e2 b4 b9 a1 8f ff	.....

Run the client on Windows

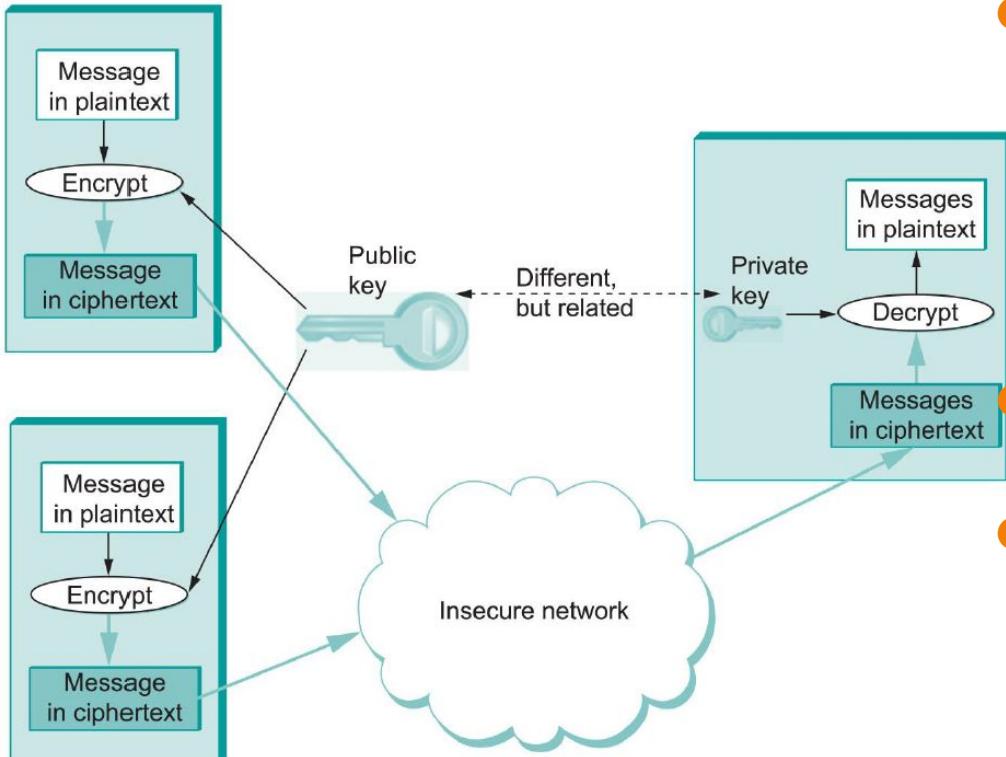
Using the Wireshark grab the encrypted data traffic to the Server port **65431**.

The data being sent on the network is encrypted and not readable.



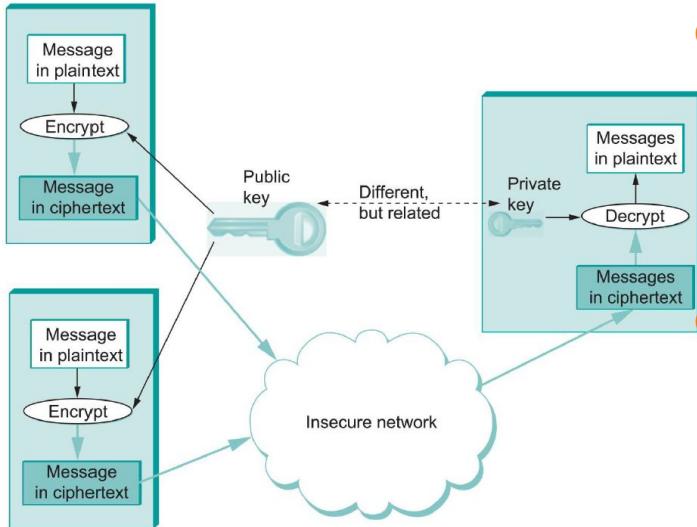
# Asymmetric Key Ciphers (private and public keys)

# Public-key Encryption and Private-key Decryption



- Instead of a single key shared by two participants, a **public-key cipher** uses a **pair of related keys**, one for encryption and a different one for decryption.
- The pair of keys is “owned” by just one participant.
- The **owner** keeps the **decryption key secret** so that only the owner can decrypt messages; that key is called the **private key**.
- The **owner** makes the **encryption key public** so that anyone can **encrypt messages** for the owner; that key is called the **public key**.
- Obviously, for such a scheme to work, it must not be possible to deduce the private key from the public key.

# Public-key Encryption and Decryption



- Any participant can get the public key and send an encrypted message to the owner of the keys, and only the owner has the private key necessary to decrypt it.
- If we think of keys as defining a communication channel between participants, secret-key cipher provides a channel that is two-way between two participants.
- In secret-key, each participant holds the same (symmetric) key that either one can use to encrypt or decrypt messages in either direction.
- A public/private key pair, in contrast, provides a channel that is one-way and many-to-one: from everyone who has the public key to the unique owner of the private key.
- In public-key, for two-way confidentiality between two participants, each participant needs its own pair of keys, and each encrypts messages using the other's public key.

# Exp 2: Generating pem files

- On WSL (Ubuntu): using the Terminal:
  - Generate **private.pem** file on WSL, using the below command
    - *openssl genpkey -algorithm RSA -out private.pem –pkeyopt rsa\_keygen\_bits:2048*
    - **genpkey utility is used to generate 2048 bits wide private key using RSA algorithm**
  - Generate **public.pem** file on WSL, with the below command by giving **private.pem** as input and ask openssl tool to extract its **public key**
    - *openssl rsa -pubout -in private.pem -out public.pem*
- **public.pem** and **private.pem** files generated on WSL, need to be copied to Windows
- **Client** is to be run on **Windows**.
  - It uses the **public.pem** to encrypt the data before sending it to the server running on WSL
- **Server** is to be run on **WSL**.
  - It uses the **private.pem** file to decrypt the data encrypted using the **public.pem**

# Exp 2: Running this program

- Build and run the Server on WSL (Ubuntu): using the Terminal:
  - Building the server executable on WSL
  - `gcc Lab4_Exp2_Asym_Server_WSL.c -lssl -lcrypto -o asym_server -Wno-deprecated-declarations`
  - For running the `./asym_server` keep `private.pem` in the same directory
- Run the Client on Windows: using the command prompt
  - `python Lab4_Exp2_Asym_Client_Win.py` keep `public.pem` in the same directory

```
smouli@Mouli: ~/NetSec/Lab  × + ▾
smouli:Lab3$ ./asym_server
Asymmetric server listening on port 65432...
Decrypted message from client: Hello from asymmetric client
smouli:Lab3$
```

← *asym\_server* running  
on WSL:

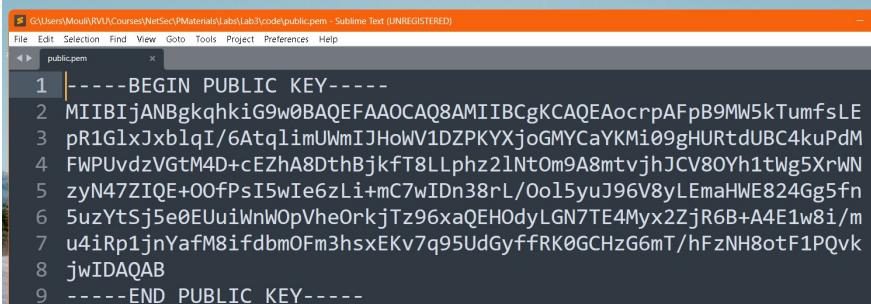
```
Command Prompt × + ▾
G:\Users\Mouli\RVU\Courses\NetSec\PMaterials\Labs\Lab3\code>python Lab3_Exp2_Asym_Client_Win.py
Received encrypted data from server: 9a19d2b916f4422c89316a78af47438bdbba34a614852bc3285b4cb6e870d40
f24ecc16f388261892b2bed1c8ed66cbf447c0ce412e9dbaefda815be32398761d3daf38d6860c5c6fb48faa67c788053c25
226d436e6138ad8fb0fe013bc9a333f7ec1128b90e9d3c861648f20227d87f484faa60f958be0ed9998a88c010167700d434
ea4e73d945378d99e6fa8ac427380e866a6758e75960d543064df8037f03e1c26ad25dbf0c8b6ad26d0cd22c178e603f1f7c
8253e0ea4e8721cd112fd5cd27bf6399f2d57a9009884232e3c60d88b858b5e600c97820b40d7b749bbcd04c4cc79872c1f9
e11b3822acd68f54186798135fd04c5650a084908ffcc5068
G:\Users\Mouli\RVU\Courses\NetSec\PMaterials\Labs\Lab3\code>
```

← *asym\_client*  
running  
on Windows:

# pem format: Explained

- The ".pem" extension stands for "**Privacy Enhanced Mail.**"
- It denotes that the file's contents are encoded in the **PEM format**: a **Base64 encoding** of binary data
- **Base64** encoding is a method for converting binary data into an ASCII string format by representing it using 64 different readable characters
- Encoding involves taking the binary data, breaking it into groups of 3 bytes (24 bits), and then splitting those 24 bits into four 6-bit groups
- Each 6-bit group is then mapped to a specific character from a set of 64 readable characters:
  - Uppercase letter (26)
  - Lowercase letters (26)
  - Digits (10)
  - Two additional symbols, '+' and '/' (2)
  - **In total 64 unique readable characters**

Sample **public.pem** contents



```
G:\Users\Mouli\RVU\Courses\NetSec\Materials\Labs\Lab1\code\public.pem - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
public.pem x
1 -----BEGIN PUBLIC KEY-----
2 MIIBIjANBgkqhkiG9w0BAQEFAOCAQ8AMIIIBCgKCAQEaocrpAFpB9Mw5kTumfsLE
3 pR1G1xJxblqI/6AtqlimUwmIJHwV1DZPKYXjoGMYCaYKMi09gHURtdUBC4kuPdm
4 FWPUvdzVGtM4D+cEZhA8DthBjkfT8Llphz21Nt0m9A8mtvjhJCV80Yh1tWg5XrWN
5 zyN47ZIQE+0ofPsI5wTe6zLi+mC7wIDn38rL/Oo15yuJ96V8yLEmaHWE824Gg5fn
6 5uzYtSj5e0EuuiWnWOpVhe0rkjTz96xaQEHOdyLGn7TE4Myx2ZjR6B+A4E1w8i/m
7 u4iRp1jnYafM8ifdbm0Fm3hsxEkv7q95UdGyffRK0GChzG6mT/hFzNH8otF1PQvk
8 jwIDAQAB
9 -----END PUBLIC KEY-----
```

# Lab 4: Summary

- **Symmetric Key Ciphers**

- OpenSSL Introduction
- Library Installations on WSL and Windows
- **Exp 1:** Server in C - WSL – Client in Python – Win

- **Asymmetric Key Ciphers**

- Explanation about the **pem** format
- **Exp 2:** Server in C - WSL – Client in Python – Win