

**LAB 12**

**Part 3: Public Key Cryptography**

**Copyright © (2022-2025) All Rights Reserved**

**Dr. Matthew Kisow**

VERSION 2

22-DEC-2024

**LAB**:NaCL Cryptography Programming – Part 3 Public Key Cryptography

**OBJECTIVE**: To develop a secure client application capable of communicating with a one-time pad server using Curve25519 elliptic curve cryptography, enabling students to practice key generation, public-key exchange, and decryption of encrypted messages in a controlled environment.

**INSTRUCTIONS**: In this lab, you will communicate with a PublicKey server. Follow the steps below to set up your client and construct requests. Ensure you follow these guidelines to interact with the server and decrypt the received message. Please follow the instructions for submitting this assignment on Blackboard.

1. **Overview of the Communication Process**
   1. **Request:** Your client sends an HTTP-like request to the server at **otp.kisow.org:12001**. The request must include your name and your public key.
   2. **Response:** The server responds with its public key and an encrypted random message (ciphertext).
   3. **Decrypt:** Use your private key and the server's public key to decrypt the ciphertext and retrieve the plaintext message.
2. **Using the Lab Python Environment**

Navigate to the “*LAB12/Part 3*” folder and enter the Python3 virtual environment.

1. Activate the virtual environment:

source venv/bin/activate

1. **Setting Up Your Python Environment**

Before you start programming your client, set up your Python environment:

1. Create a virtual environment:

Python3 -m venv .venv

1. Activate the virtual environment:

* **Linux/macOS:**

source .venv/bin/activate

* **Windows:**

.venv\Scripts\activate

1. Install the necessary dependencies:

pip install wheel pynacl requests

1. **Crafting a Client Request**

Construct your client’s request in the following format. Replace **<Your Name>** and **<Your Public Key>** with your details:

**CRYPTO 1.0 REQUEST**

**Name: <Your Name>**

**PublicKey: <Your Public Key>**

* **Your Name:** Your full name as a string.
* **PublicKey:** Your public key in hexadecimal format, generated using Curve25519 elliptic curve cryptography.

1. **Example Request and Reply**
   * **Client Request**

**CRYPTO 1.0 REQUEST**

**Name: Student Name**

**PublicKey: 7d14ceb0905065ce5b6e27a95d4c511ec2baf98887c74e662808043b85f1287d**

* + **Server Reply**

**CRYPTO 1.0 REPLY**

**Name: Dr. Matthew Kisow**

**PublicKey: 245fb800e3ec462d84389b4e01074cb85e6f49e18df30002c2da69d9a853e507**

**Ciphertext: 71927e1b6694fa3665b1d58ad9f13c4f63e5fcc087d10700212bbf554c57ecde40f3a42e75ea2ef1ce64cdccda875339b994a66bca347f5df2fb778a5f927386c8bced280418f53115e38086b8e54a443bd2c57957a2f29c8bd89e77def637e6**

1. **Steps to Write Your Client**
2. **Generate a Key Pair:** Use the **nacl.public** module to generate a private and public key pair:

from nacl.public import PrivateKey

private\_key = PrivateKey.generate()

public\_key = private\_key.public\_key

print("Your Public Key (hex):", public\_key.encode().hex())

1. **Send the Request:** Use the **requests** library to send your request:

import requests

url = "http://otp.kisow.org:12001/"

payload = (

"CRYPTO 1.0 REQUEST\r\n"

"Name: Student Name\r\n"

f"PublicKey: {public\_key.encode().hex()}\r\n"

)

response = requests.post(url, data=payload)

print(response.text)

1. **Decrypt the Response:** Parse the server's reply to extract its public key and the ciphertext. Use **nacl.public.Box** to decrypt the ciphertext:

from nacl.public import PublicKey, Box

import binascii

# Example response

server\_response = """

CRYPTO 1.0 REPLY

Name: Dr. Matthew Kisow

PublicKey: 245fb800e3ec462d84389b4e01074cb85e6f49e18df30002c2da69d9a853e507

Ciphertext: 71927e1b6694fa3665b1d58ad9f13c4f63e5fcc087d10700212bbf554c57ecde40f3a42e75ea2ef1ce64cdccda875339b994a66bca347f5df2fb778a5f927386c8bced280418f53115e38086b8e54a443bd2c57957a2f29c8bd89e77def637e6

"""

# Extract server's public key and ciphertext

server\_public\_key\_hex = "245fb800e3ec462d84389b4e01074cb85e6f49e18df30002c2da69d9a853e507"

ciphertext\_hex = "71927e1b6694fa3665b1d58ad9f13c4f63e5fcc087d10700212bbf554c57ecde40f3a42e75ea2ef1ce64cdccda875339b994a66bca347f5df2fb778a5f927386c8bced280418f53115e38086b8e54a443bd2c57957a2f29c8bd89e77def637e6"

server\_public\_key = PublicKey(binascii.unhexlify(server\_public\_key\_hex))

box = Box(private\_key, server\_public\_key)

ciphertext = binascii.unhexlify(ciphertext\_hex)

plaintext = box.decrypt(ciphertext)

print("Decrypted message:", plaintext.decode())

1. **Testing**
   1. Save the script as **part3.py**.
   2. Run it from the command line:

python3 part3.py

**TIP**

* Make sure your request follows the correct format, or the server will return a 400 BAD REQUEST.
* Log all responses and errors to debug any issues with your client.

**DELIVERABLE**

Write a Python3 script named **part3.py** that communicates with the server, receives the *public key* and *ciphertext* as a stream, then decrypts and prints the plaintext message to the screen.

*This is the working pseudocode solution to the program.*

BEGIN PROGRAM

# Source Code File: Part 3: CLIENT: Public Key Cryptography

# Name: part3.py

# Author: <students name>

IMPORT base64

IMPORT nacl.utils

IMPORT PrivateKey, PublicKey, Box FROM nacl.public

IMPORT requests

# Generate client key pair

ASSIGN PrivateKey.generate() TO client\_private\_key

ASSIGN client\_private\_key.public\_key TO client\_public\_key

DEFINE FUNCTION send\_request(server\_url)

"""Send a properly formatted request to the server."""

# Format the request

ASSIGN (

"CRYPTO 1.0 REQUEST\\r\\n"

"Name: Student Test Name\\r\\n"

"PublicKey: {base64.b16encode(bytes(client\_public\_key)).decode()}\\r\\n"

) TO request\_payload

PRINT "Sending request to the server:"

PRINT request\_payload

# Send the request to the server

TRY

ASSIGN requests.post(server\_url, data=request\_payload) TO response

ASSIGN response.text TO response\_text

PRINT "\\nResponse from the server:"

PRINT response\_text

# Parse and decrypt the response

IF response.status\_code == 200 THEN

CALL parse\_and\_decrypt\_response(response\_text)

ELSE

PRINT "Error from server: {response.status\_code} {response.reason}"

EXCEPT requests.RequestException AS error

PRINT "Request failed: {error}"

END FUNCTION

DEFINE FUNCTION parse\_and\_decrypt\_response(response\_text)

"""Parse the server's response and decrypt the ciphertext."""

TRY

# Split response lines

ASSIGN response\_text.split("\\r\\n") TO lines

IF lines[0] IS NOT EQUAL TO "CRYPTO 1.0 REPLY" THEN

PRINT "Invalid response format."

RETURN

ASSIGN None TO instructor\_name

ASSIGN None TO instructor\_public\_key

ASSIGN None TO ciphertext

# Parse headers

FOR line IN lines[1:] DO

IF line STARTS WITH "Name:" THEN

ASSIGN line.split(": ", 1)[1] TO instructor\_name

ELSE IF line STARTS WITH "PublicKey:" THEN

ASSIGN PublicKey(base64.b16decode(line.split(": ", 1)[1])) TO instructor\_public\_key

ELSE IF line STARTS WITH "Ciphertext:" THEN

ASSIGN base64.b16decode(line.split(": ", 1)[1]) TO ciphertext

IF instructor\_name IS None OR instructor\_public\_key IS None OR ciphertext IS None THEN

PRINT "Incomplete response from server."

RETURN

PRINT "Name: {instructor\_name}"

PRINT "Public Key: {base64.b16encode(bytes(instructor\_public\_key)).decode()}"

# Decrypt the ciphertext

ASSIGN Box(client\_private\_key, instructor\_public\_key) TO box

ASSIGN box.decrypt(ciphertext) TO plaintext

PRINT "Decrypted Message: {plaintext.decode('utf-8')}"

EXCEPT Exception AS error

PRINT "Failed to parse or decrypt the response: {error}"

END FUNCTION

IF THIS SCRIPT IS RUN AS MAIN PROGRAM THEN

# Replace with the appropriate server URL

ASSIGN "http://otp.kisow.org:12001/" TO server\_url

CALL send\_request(server\_url)

END PROGRAM