# University of Cape Town
# Department of Computer Science
# CSC4026Z
# Network and Internetwork Security 2022
# Practical

## 1 INTRODUCTION

The following tutorial is the practical component for CSC4026Z and is to be completed in groups of four. Please send the names of your group members to Keegan White (WHTKEE004@myuct.ac.za), who is acting as the TA for the course this year.

The objective of the practical is to gain experience with cryptographic functions and exercise your knowledge of protocols, specifically exchanging encrypted messages in a group chat by implementing a small Pretty Good Privacy (PGP) cryptosystem that combines shared key encryption, public-key encryption and certificates. Thus, the system should focus on validating key authenticity and simulating/replicating message confidentiality and authentication aspects of PGP. The Certification Authority role required is more X.509-like than PGP-like.

## 2 THE TASK

Create client applications (minimum 3) that form a group chat and initially exchange and validate each other's public keys issued by a Certification Authority that they trust. Thereafter, messages should be transmitted within the group chat, using the shared key, private key, public key, hashing and compression functions, in the same manner as PGP.

In a communication session, the sender should be able to send a message that is received by all other members of the group. The emphasis is on group messaging. See Section 3 for the rubric.

To send a message, a Client communication system (based on UDP or TCP) should be established. In a one-way communication session, the clients can be separated as a sender and multiple receivers. Enabling Clients to participate in a continuous chat (act as both a sender and receiver), i.e., a two-way communication session, earns full marks. See Section 3 for the rubric. Communication sessions are initiated in various manners (typically the receiver should listen while the sender initiates communication). Moreover, a Client application may represent a class or separate instance of the system running, as long as there are three or more entities that may communicate.

It is important to note that the trusted third-party interaction (obtaining certificates from the Certification Authority) does not have to exist as a third Client in your final submission. However, certificates must be generated and exchanged for public-key authentication.

The sending and receiving applications are expected to have:

- A private and public key pair of their own
- The public key of the Certification Authority

- A certificate (containing the client's own public key) signed by the Certification Authority

The sending and receiving applications are expected to:

- Setup a connection for group message communication
- Exchange certificates
- Encrypt, compress, hash messages (and the reverse)
- Exchange encrypted messages

For testing purposes, please include debugging statements, i.e., output the encrypted message, session key, hashed message etc., along with the decrypted message, decrypted session key etc., to the console so that the system run is clearly documented.

A short write-up (no more than 5 pages) is required to explain and document your cryptosystem implementation, communication connectivity model, key management, choice of cryptographic algorithms, testing procedure and assumptions made. Diagrams are recommended.

Document how to execute/run the program(s) submitted (this can be written as a separate document i.e., README if needed). Code comments earn one mark in the "Order documented and justified" and the "Overall documentation quality" sections. A one-way communication system earns five marks in the "Messages sent and received both ways" section.

# 3 ASSESSMENT

To be determined based on running the system and write-up:

- Communications implementation
- Security implementation
- Overall system design and functionality to achieve the stated goal
- Evidence of testing

| Rubric | Total (130) |
|---|---|
| | |
| Communications Implementation | 30 |
| Sender and receivers connected (Code) | 5 |
| Messages sent and received both ways (Code) | 10 |
| Documentation on communication process (Write-up) | 5 |
| Keys generated, exchanged, decrypted and validated (Code) | 5 |
| Documentation on key exchange process (Write-up) | 5 |
| | |
| Security implementation | 30 |
| Message integrity implemented (Code) | 5 |
| Documentation on message integrity (Write-up) | 5 |
| Message authentication implemented (Code) | 5 |
| Documentation on message authentication (Write-up) | 5 |

| | |
|---|---|
| Message confidentiality implemented (Code) | 5 |
| Documentation on message confidentiality (Write-up) | 5 |
| | |
| **Overall system design** | **50** |
| Usage of compression (Code) | 5 |
| Documentation on compression (Write-up) | 5 |
| Shared key encrypted appropriately (Code) | 5 |
| Documentation on the shared key used (Write-up) | 5 |
| Correct order of PGP implementation (Code) | 5 |
| Order documented and justified (Write-up and Code) | 5 |
| Overall state of application (does it run to completion) (Code) | 10 |
| Overall documentation quality (Write-up and Code) | 10 |
| | |
| **Evidence of testing** | **20** |
| Debug statements (Code) | 10 |
| Documentation on testing procedure (Write-up) | 10 |

# 4 CRYPTOGRAPHIC AND IMPLEMENTATION DETAILS

The prescribed programming language for this practical is Java. We do **not** expect you to code your own encryption libraries and thus encourage you to use encryption libraries such as Bouncy Castle for Java. The Java API has some excellent documentation on the security libraries included in the SDK. Public / private key pairs should be created for use with RSA (for public-key encryption) and a shared key generated for use with DES, AES etc. (for shared key encryption).

**Cryptography Details:**
For asymmetric encryption, we encourage you to use the RSA algorithm in ECB mode with PKCS1 padding. The algorithm specification string in Java is “RSA/ECB/PKCS1Padding”. For symmetric encryption, we encourage you to use AES algorithm in CBC mode with PKCS5 padding. The algorithm specification string in Java is “AES/CBC/PKCS5Padding”.

As mentioned previously, for evidence of testing and insight, you should ensure that sufficient ‘debug’ statements are included so that the ciphertext and message components can be viewed.

# 5 DUE DATE

The system, along with the write-up, is due on **Friday, May 27th, 2022** at **23:59** and must be submitted via Vula.

This practical component makes up **40%** of the module assessment.

# 6 QUERIES

Any queries should be directed to Keegan White (WHTKEE004@myuct.ac.za).