Advanced Topics in Online Privacy and Cybersecurity (67515), Spring 2025

Assignment 2: Building a Mixnet

Due date: July 7th, 2025

Overview

In this assignment, you will implement a **mix network**, based on the design introduced by David Chaum in his seminal 1981 paper, (DOI link). The goal is to gain hands-on experience with anonymous communication protocols that delink senders and recipients.

Your system will consist of multiple clients and three mix servers. All components must be containerized using Docker, and communication will occur over a simulated local network.

Functionality

System Components

Your system will consist of the following components:

- **Mix Servers** that process onion-encrypted messages
- **Clients** that send and receive anonymous messages

System Starting Parameters

Setup

Before running the protocol, your containers will need to know all necessary configuration data. Using a configuration json file in a fixed location within your containers is a simple solution for this. Configuration data you should be providing:

- mix server addresses and public keys,total number of messages expected per round.
- total number of messages expected per round.
- anything else that you think is necessary for the protocol

This setup phase is required solely for technical convenience and testing reproducibility. It is not considered part of the actual privacy-preserving protocol design that is the focus of this problem set.

The protocol itself should be agnostic to the number of clients, but your report must include benchmarks with varying numbers of clients (for example, 2, 5, and 10) with a discussion of latency and scalability trends. We also expect a short description of how to recreate such test runs, ideally with a run script.

You may implement this setup phase however you like (e.g., via shared configuration files or bootstrapping scripts), as long as the information is correctly distributed to all relevant parties before any set of round begins.

Round-Based Communication

To enhance anonymity, your system must implement a simple **round-based communication model**. In each round, all active clients are expected to submit exactly one message to the mix network, even if the payload is a dummy message. The mix servers should process messages in fixed-length batches corresponding to rounds, and only deliver messages once all inputs for that round are received.

API Specification

You must implement the following API, which defines the interactions between clients and the mix servers. Communication between components should be implemented using gRPC.

You may modify the exact method signatures, data formats, or communication details to suit your implementation, **provided that any deviations from the specification are clearly documented** in your write-up.

Client API

- `prepare_message(message, recipient_pubkey, mix_pubkeys, round)` Constructs a multi-layer encrypted message using the recipient's public key and a list of public keys for the mix servers (innermost to outermost). The message is structured such that each mix only learns the address of the next hop. The parameter `round` indicates the round in which the message should be processed. Dummy messages must also include a round number.

- `poll_messages()`
  Called by the client to retrieve any messages delivered to them after being routed through the mix cascade. This method returns encrypted messages that the client must decrypt locally.

Mix Server API

- `forward_message(payload, round)`
  Receives a payload encrypted under the mix's public key and tagged with a round number. The mix decrypts one layer to reveal the address of the next hop (either another mix or the final recipient) and stores it internally. When all messages for a round have been received, the mix forwards the inner encrypted payloads in a batch. If the next hop is a recipient, the message is stored for delivery via polling, Additionally, the final mix server stores messages to files by round for testing purposes.

Round Structure in the API

Each message must be associated with a round number. All submitted messages for a given round should be processed and delivered together. The final mix server should write the output of each round to a file for inspection and testing.

Relaxing Assumptions

For the purposes of this assignment, you may assume that all clients and mix servers have access to each other's public keys. You may also assume that clients know the address of the final mix server responsible for delivering messages to each recipient, that mixing paths are equal and static for all messages, and that recipients should know which mix server to poll for new messages.

Containerization and Setup

All components must be containerized and orchestrated using `docker-compose`. Your system must demonstrate the end-to-end operation of the mix network in a fully automated environment.

- **Each component (clients and mix servers)** must run in separate Docker containers.
- **Use** `docker-compose` to define and run a multi-container Docker application.
- You must define a Docker network within `docker-compose.yml` to enable secure and isolated communication between the mix and the clients.
- Provide a script `setup.sh` that builds and launches the containers, demonstrates at least one full message and reply round with 2 clients, and then cleans up the environment.

Programming Language. Your implementation must be written in Go or Python. Students that implement the assignment in Go will receive a 2 point bonus.

Security

Anonymity Guarantees

In your report, provide a detailed explanation of the standard threat model assumed for mix networks and how the system's design offers anonymity guarantees within that model. Discuss how layered encryption protects both sender and receiver anonymity, and the purpose of including random values (nonces) in encrypted messages.

Runtime Benchmarks

You must report the runtime performance of your system for varying workloads. Specifically, measure and report the following metrics for between two and ten concurrent clients:

- Message preparation time (client side)
- Decryption and forwarding latency (mix side)
- End-to-end delivery time

Perform these measurements across at least five different message sizes to evaluate the impact of payload length. Additionally, run and include results from an unencrypted baseline implementation (i.e., messages sent without any encryption or mixing) to serve as a performance comparison.

Plot each of these metrics against both the number of clients and message size to clearly illustrate performance trends. Ensure that all graphs are clearly labeled with appropriate titles, axis labels, and legends for easy interpretation.

Security Analysis

In your report, briefly describe (in a couple of sentences each) at least two plausible attack scenarios that could compromise the anonymity or integrity of this type of mix network, considering adversaries capable of observing or interacting with the system. For each attack, also discuss possible mitigations or defenses. You are encouraged to reference relevant literature to support your analysis.

Submission Guidelines

You must submit the following:

1. All source code and Docker related files
2. A script to build, configure, and launch all containers, demonstrate communication, and teardown
3. A report with the following sections:
    - System architecture and design decisions
    - Implementation of the API and encryption logic
    - Security analysis
    - Runtime benchmarks
    - Proposed attack scenarios