COMP4337/9337 Securing Fixed and Wireless Networks

Assignment Report: DIMY Protocol Implementation

Group: Team5 (Wed 4--6pm lab)

Group Members:

- Yuchen Bai(z5526405)
- Shukun Chen(z5466882)
- Mengyu You(z5471795)

Demo Video link: https://youtu.be/i955VYgFBVw

In case above link does not work: onedrive link

1. How to Run the Program

Prerequisites

Python 3.8 or higher
Required Python packages:
pip install cryptography bitarray mmh3 numpy

Running Instructions

1. Start the backend server:
   python3 DimyServer.py
2. Start client nodes (in separate terminals):
   python3 Dimy.py 15 3 5 127.0.0.1 55000
   Where the parameters are:
   - 15: Time interval (seconds) for generating EphID
   - (t value, can be 15,18,21,24,27,30)
   - 3: Minimum shares required for reconstruction (k value, k >= 3)
   - 5: Total shares to generate (n value, n >= 5, k < n)
   - 127.0.0.1: Server IP address
   - 55000: Server port
3. Run the attacker node (in a separate terminal):
   python3 Attacker.py 127.0.0.1 55000

2. Executive Summary

This report covers the implementation of the DIMY (Did I Meet You) protocol for privacy-preserving COVID-19 contact tracing. The protocol utilizes several cryptographic mechanisms including Shamir's Secret Sharing, Diffie-Hellman key exchange, and Bloom filters to achieve privacy preservation. Nodes periodically generate ephemeral identifiers (EphIDs)and create shares based on secret sharing and broadcast those. Once sufficient shares have been collected, nodes are able to reconstruct EphIDs and compute shared encounter identifiers (EncIDs) using the Diffie-Hellman protocol. These EncIDs are then stored in Bloom filters, which allows for privacy-preserving contact tracing.

Additionally, we investigate potential security issues in the protocol and present an example of false positive report attack where the attacker may violate the privacy promises of the system

3. Implementation Details

Key Implementation Features

- **Multi-threaded Architecture**: All components use threading to handle multiple tasks concurrently (e.g., generating EphIDs, broadcasting shares, listening for broadcasts, maintaining DBFs, sending queries)
- **Message Drop Mechanism**: Implemented 50% message drop rate to simulate nodes moving in and out of range
- **Bloom Filter Operations**: Efficient implementation of Bloom filters with configurable size and hash functions
- **Cryptographic Primitives**: Used established libraries for Diffie-Hellman, hashing, and cryptographic operations

- **Network Communication**: UDP multicast for node-to-node communication and TCP for node-to-server communication

Implementation Components

1) **Node Implementation (Dimy.py)**
    - Generates 32-byte ephemeral identifiers (EphIDs) at configurable intervals
    - Implements Shamir's Secret Sharing to split EphIDs into shares
    - Uses UDP multicast for broadcasting shares between nodes
    - Implements Diffie-Hellman key exchange for computing EncIDs
    - Maintains Daily Bloom Filters (DBFs) for storing encounters
    - Creates Query Bloom Filters (QBFs) for querying exposure status
    - Generates Contact Bloom Filters (CBFs) for reporting positive cases
2) **Server Implementation (DimyServer.py)**
    - Receives and stores CBFs from COVID-positive users
    - Processes QBF queries from nodes to check for exposure risk
    - Performs Bloom filter matching between QBFs and stored CBFs
    - Maintains CBFs with appropriate expiration (14 days)
3) **Attacker Implementation (Attacker.py)**
    - Eavesdrops on UDP broadcasts to collect shared secrets
    - Reconstructs EphIDs from collected shares
    - Implements a false positive report attack by creating fake CBFs

Successfully Implemented Features

- EphID generation and sharing using Shamir's Secret Sharing
- EphID reconstruction and verification - Diffie-Hellman key exchange for EncID
- DBF, QBF, and CBF creation and management
- Backend server for CBF storage and QBF matching
- Simulated COVID-positive reporting - False positive report attack implementation

Features Not Fully Implemented

The attacker successfully collected EphIDs from various nodes, reconstructed the EphIDs, and generated a fake positive CBF. However, this CBF was only uploaded to the server and successfully saved by the server, but it was not reflected in the DIMY nodes' frontend.

4. Design Trade-offs

Shamir Secret Sharing: Performance vs. Security

Trade-off:

We implemented the k-out-of-n Shamir Secret Sharing scheme for splitting the EphID into shares. The trade-off here was between security and performance. A larger n (the number of shares) means more shares must be combined to reconstruct the EphID making the system more secure. However, it also raises the computational cost and time taken to generate and transmit the shares.

Impact:

- Pros: k-out-of-n sharing (for k in 1...n) as a security measure whereby no single node has full access to the EphID. This means that an attacker would have to access several nodes to be able to rebuild the EphID.
- Cons: Increasing n results in more network traffic due to the need to broadcast more shares. Additionally, for each reconstruction, more computational resources are needed to process the shares and validate the EphID hash, which could lead to performance bottlenecks.

Use of Bloom Filters for Storing Encounters vs. Data Accuracy

Trade-off:

Bloom Filters were used to efficiently store and query encounter data (EncID) in the form of DBFs, QBFs, and CBFs. While Bloom Filters are efficient and offer constant time complexity for insertion and querying, they have the inherent trade-off of false positives. A Bloom Filter might indicate a match even when no actual match exists, leading to the risk of false positive notifications in the context of contact tracing.

Impact:

- Pros: Bloom Filters are highly space-efficient, allowing us to store encounter data without consuming excessive memory.
- Cons: The probability of false positives must be carefully managed by tuning the number of hash functions and the size of the filter. While we minimized this by using 3 hash functions, there remains a small chance of a false positive, which could lead to incorrect risk alerts.

5. Special Implemented Features

1) **Efficient EphID Reconstruction**
    - The implementation efficiently tracks received shares and attempts reconstruction only when sufficient shares are available
    - Share verification ensures only valid shares contribute to reconstruction
2) **Robust Bloom Filter Implementation**
    - Our Bloom filter implementation supports operations like merging filters and efficiently checking for matches
    - This enables effective privacy-preserving contact tracing
3) **Comprehensive Debugging Output**
    - Detailed terminal output shows the state of each operation
    - Time-stamped logs make it easy to follow the protocol flow

6. Possible Improvements and Extensions

Implementing Encryption for Backend Communication (TCP)

Improvement: In the current implementation, the backend communication is not encrypted. A TLS/SSL layer should be added to the TCP communication between nodes and the backend server to prevent eavesdropping and man-in-the-middle attacks.

How to Realize: Integrating a secure communication protocol like TLS would require using libraries such as ssl in Python to encrypt TCP sockets. This would ensure that all data exchanged between nodes and the server is secure.

Optimizing Bloom Filter Performance

Improvement: The current implementation of Bloom Filters can be optimized by dynamically adjusting the size of the filter and the number of hash functions based on the volume of encounters.

How to Realize: Implementing adaptive Bloom Filters that automatically adjust the filter size and hash functions depending on the number of data items would reduce the false positive rate and make the system more efficient as the number of encounters increases.

7. Explanation for Task 11

We choose a false positive report attack.

Explanation on how this attack can be launched on the DIMY protocol:

Attack on Node-to-Node Communication (Implemented; More details in video demo)

Our implemented attack (Attacker.py) demonstrates a false positive report attack:

- The attacker passively collects broadcast shares from legitimate nodes
- When sufficient shares are collected, the attacker reconstructs EphIDs
- The attacker creates a fake CBF containing these reconstructed EphIDs
- The attacker submits this CBF to the backend server claiming COVID-positive status

This attack causes false positive notifications to legitimate users who have been in proximity to the targeted nodes. The impact is significant: it undermines trust in the system by creating false alerts, potentially causing unnecessary quarantines and testing.

Attack on Node-to-Server Communication (Not implemented)

A potential attack on the node-to-server communication is a man-in-the-middle (MITM) attack:

- The attacker positions themselves between nodes and the backend server
- The attacker intercepts QBF queries and CBF uploads
- The attacker can:
    - ➢ Modify query results to create false negatives (hiding exposure risk)
    - ➢ Track specific users by analyzing their QBF patterns
    - ➢ Collect QBFs to build a database of user movements

8. Code Sources

- Implementation of Shamir's Secret Sharing is based on examples from the secrets library documentation
- Diffie-Hellman key exchange implementation uses the cryptography library
- Multicast UDP socket code was adapted from Python documentation examples

Assignment Diary

Week	Tasks	Yuchen Bai (z5526405)	Shukun Chen (z5466882)	Mengyu You (z5471795)
1	Project kickoff and understanding requirements	Studied DIMY paper and assignment requirements		
2	Continue learning	Learning Shamir's Secret Sharing and Bloom filters		
3	Design and architecture planning	Designed message format and protocol flow	Planned Bloom filter implementation	Sketched Shamir's SS implementation
4	Core function implementation	Created Diffie-Hellman key exchange mechanism	Developed Bloom filter operations	Implemented EphID generation and Shamir's SS
5	Network communication	Implemented UDP broadcast mechanism	Developed TCP client-server communication	Added message serialization and parsing
6	Integration and attacker implementationg	Combined components and fixed integration issues	Implemented false positive attack	Debugged Shamir's Secret Sharing implementation
7	Security analysis and testing	Analyzed security mechanisms;Resolved byte-int conversion problems	Fixed UDP multiport listening issues	Prepared demonstration video and final report