Privately (and Unlinkably) Exchanging Messages Using a Public Bulletin Board
Electronics-ICT

(B-KUL-JLI1QZ)

Arne Vanmarcke, Thibault Dekock

December 11, 2024

Instructor: Dr. Vincent Naessens

1 Introduction

This project implements a Privacy-Friendly Bulletin Board (PBB) application. The system leverages Java RMI for communication and JCA/JCE to ensure secure cryptographic operations. The application is designed to enable clients to anonymously and securely store and retrieve messages from a centralized bulletin board without compromising privacy or metadata.

The implementation includes a proof-of-work (PoW) mechanism to mitigate abuse, such as spamming or Denial-of-Service attacks.

2 Design decisions

To create a bump between two clients, bumpfiles are generated by logging into the webchat. The client then searches for another bumpfile and connects to this client. Bumpfiles contain the publicKey, index and tag of the client. All the contents of these files are then BASE64 encoded. This adds an extra barrier to prevent an attacker from easily reading the contents of this file. The file name contains the client's username and a random number to distinguish two clients with the same username.

For sending messages we utilize an AES-encrypted byte buffer for transferring all message data between the bulletin board and the client. Rather than using a delimiter. This has the advantage that no intruder can break the system by making use of a simple delimiter.

To encrypt message information using AES encryption we need a certain secret key, this secret key is generated using Elliptic Curve Diffie-Hellman (ECDH), and is derived using HMAC (with hash function SHA-256). Using Diffie Hellman for generating this key has as advantage that there is no need to ever send this key over the network (reduces risk of interception).

3 Advanced features

Although we only implemented one advanced feature (protection against a Denial-of-Service attack), we'll briefly mention how the others can be implemented as well.

- Supporting recoverability of corrupted states:
  Encrypting and storing a serialized version of the state on the server can create backup states to revert to after the state has been corrupted. Using an encryption key known only to the client will ensure security for this implementation.
- Increasing scalability through partitioning:
  By partitioning the bulletin board across multiple servers or chunks, we can help scale the system to allow more users to send messages. The partition design could be either static by dividing the index range into chunks and managing these ranges by separate servers or dynamic by monitoring the load and dynamically redistributing indices to new servers.
- Tackling Denial-of-Service (DoS) attacks:
  We chose to implement the protection against DoS attacks. A few options are possible here:
  - Rate limiting: restrict the number of requests per user or IP over a time period.

  - Sessions: Introduce authenticated client access using tokens. This way, each client gets a quota for requests per session.
  - Proof-of-Work (PoW) mechanism: clients must solve a computational puzzle before processing their requests. This is the option we chose for implementation. Since the previous options, both require server-side monitoring, either to track a user's number of messages or create / follow up sessions and tokens, we wanted to reduce the server load as much as possible. By using a PoW mechanism, we move all computational requirements to the client. The server only needs to generate a hash as a 'challenge' after which the client will compute a valid nonce that satisfies the difficulty requirement of the PoW. The difficulty of this challenge can be set dynamically depending on the spamming behaviour of the user. For us, the perfect difficulty for the challenge was three 0s at the start of the hash, which resulted in a computational time of around 20-500ms. Spamming now becomes a significant computational burden on spammers attempting to flood the system with requests since they must solve a PoW challenge before every request. This is while the server can quickly and efficiently verify the correctness of a client's solution.

4 SWOT analysis

- Strengths
  - Puzzle lowers amount of request in a small time frame.
  - Sending messages in a buffer doesn't require a certain message format.
- Weaknesses
  - Missing exception handling. To handle unencountered errors.
  - PoW introduces computational delays for the clients. Although it is beneficial for our server, users of the app might not like this solution, especially when using mobile devices.
  - At this moment, the bulletin board is hosted centrally, making it a single point of failure.
  - The current implementation doesn't include persistent storage for messages, meaning data is lost upon server restart.
  - Possibility to break bumpfile parser by including a separator in the username.
- Opportunities
  - Implement distributed bulletin boards to handle higher loads.
  - Integrate group-based messaging for broadcasting to multiple clients at the same time
- Threats
  - If an adversary controls both the server and a mixing network, metadata unlinkability can be compromised.
  - Quantum computing could compromise SHA-256 and AES encryption.

Link to github