elfocrypt

Phase I

Holly Dinh
Mani Maghsoudlou

Application Properties

- Client-server paradigm
- One-to-one and group chat
- Cross-platform desktop application using Node.js and Electron
- Internal on-disk NoSQL (key/value) database for both server and client using LevelDB
- JSON Web Token for authentication
- Openssl cli to generate user's keys on the client side
- Node.js crypto module (wrapper around openssl lib) for encryption/decryption
- Users need to register online before using the client application

Restful https server using express

- Probably on JWS with Let’sencrypt certificates
- All POST data transferred through request body and not the URL (no query strings)
- NoSQL database => No SQL Injection
- Server acts as a distributor and cannot read user messages
- Server is not involved in the key distribution mechanism

Use-cases

- Register (create an account)
- Login
- Add a friend
  - *send a friend request*
  - *receive a friend request*
- Send messages
- Receive messages
- Logout

Assets/Stakeholders

- Stakeholders
  - *Registered user*

- Assets
  - *User credentials*
  - *Messages*
  - *List of friends*

Adversarial Model

- Active outsider
  - Man in the Middle attack
  - Evil Maid attack
  - Replay attack

- Passive outsider
  - eavesdropping
  - traffic analysis

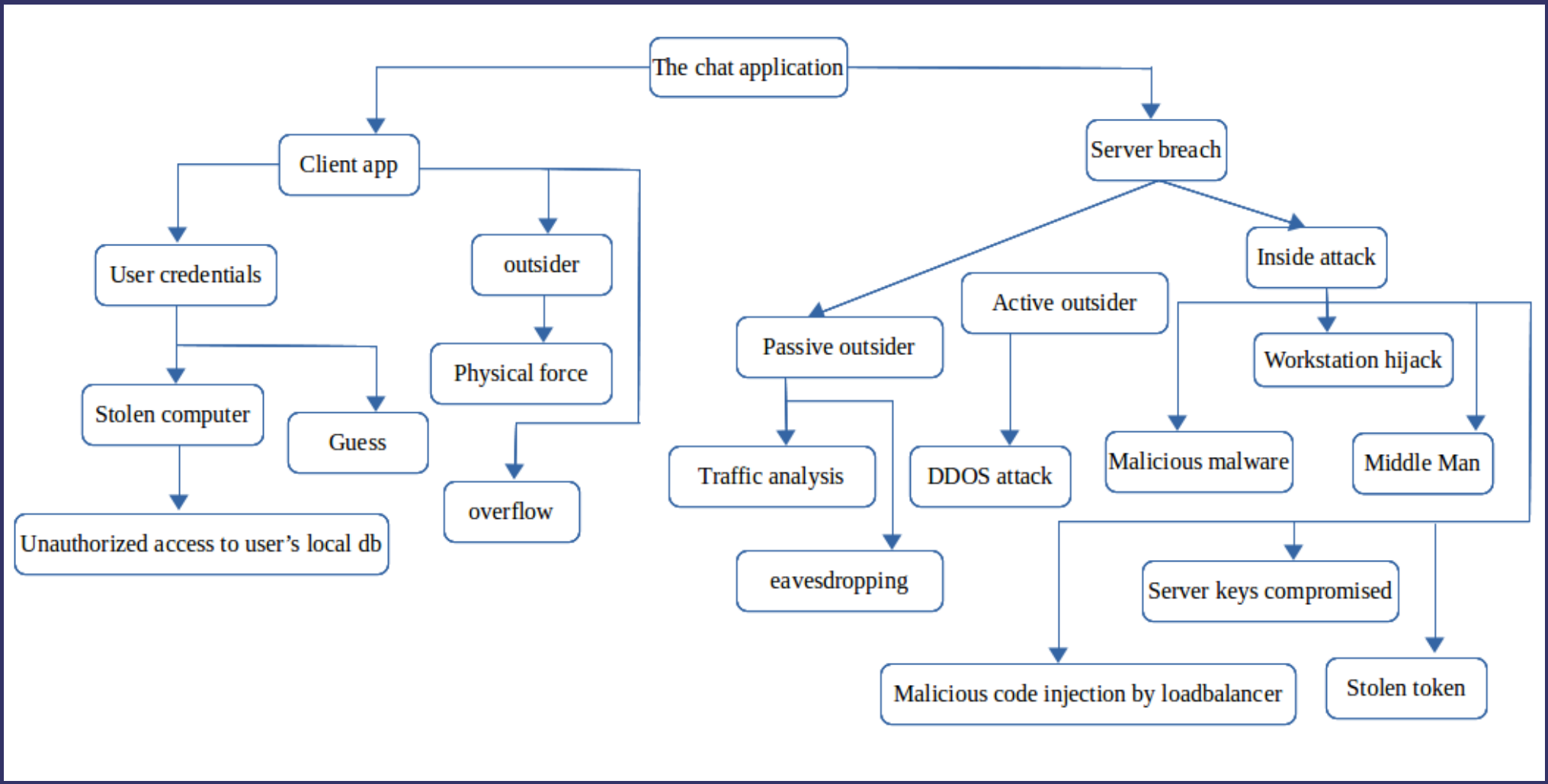
- Passive insider
  - packet sniffing between the *PaaS loadbalancers* and the server

Possible Vulnerabilities

- Taking down the server (DDOS)
- Stolen computer

  - *gain access to the user's local database on the user's machine*

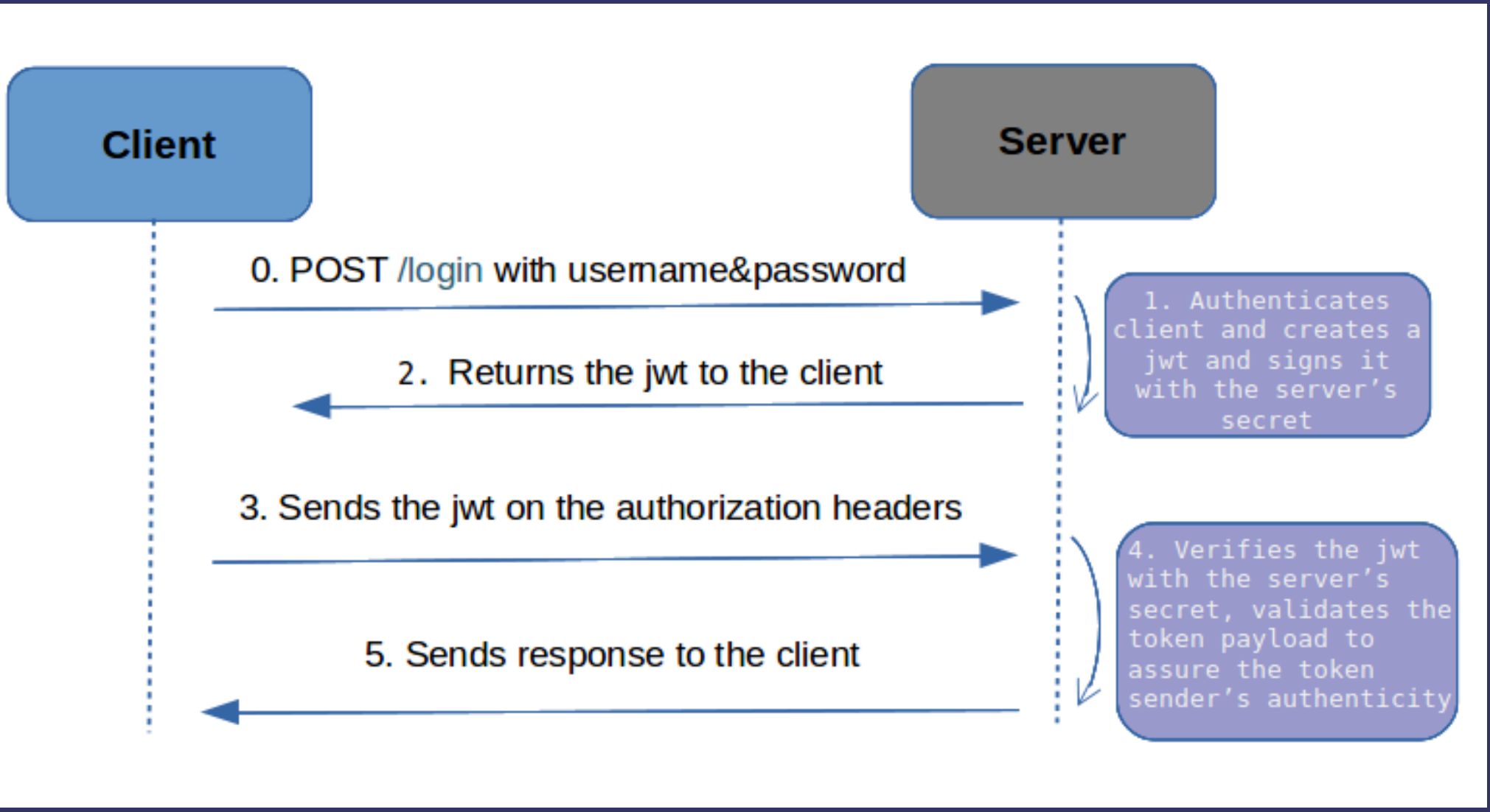
Attack Tree


The chat application
Client app
Server breach
User credentials
outsider
Inside attack
Active outsider
Passive outsider
Workstation hijack
Physical force
Stolen computer
Guess
Traffic analysis
DDOS attack
Malicious malware
Middle Man
overflow
Unauthorized access to user's local db
eavesdropping
Server keys compromised
Malicious code injection by loadbalancer
Stolen token

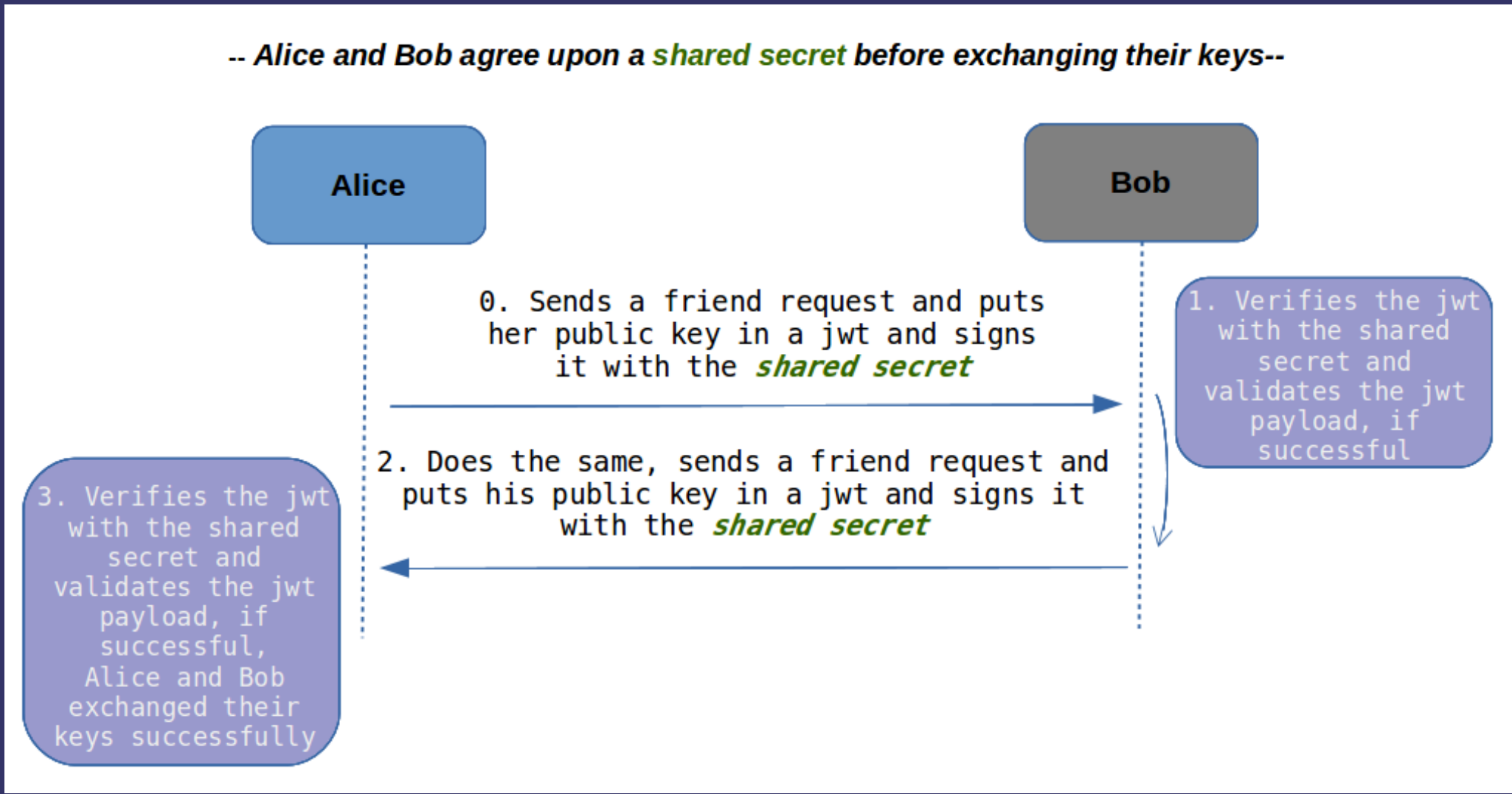
Previous Related Works

- Signal (https://whispersystems.org/)
- WhatsApp (https://www.whatsapp.com/)
- Telegram (https://core.telegram.org/api)

Application Authentication Flow



Key Distribution Scheme



Solution/Analysis

- Secure connection using the latest TLS
- Using JWT to authenticate requests
- JWTs expires after 60 mins by default
- Message confidentiality

  - encryption using openssl's aes-256-cbc cipher suite with two random keys, 256 bits for the message (may replaced with Diffie-Hellman secret) and 128 bits for the iv

  - message sent as a payload of a JWT that is signed by the user
- Message Integrity

  - using HMAC SHA-256 digest

  - each message iv is also protected by the HMAC as part of the digest