SDMAY20-52: Key Exchange outside of TCP/IP

Logan Woolery
Jacob Moody
Jordan Svoboda

Joel Wacker
Jack Potter
Andre C

Under the Advisement of Dr. Julie Rursch

INTRODUCTION

In a world where it is increasingly difficult to trust even the very backbone of modern communication, it is necessary to develop a system through which encryption keys can be communicated in absolute security, irregardless of the sanctity of the networks across which they might be transmitted.

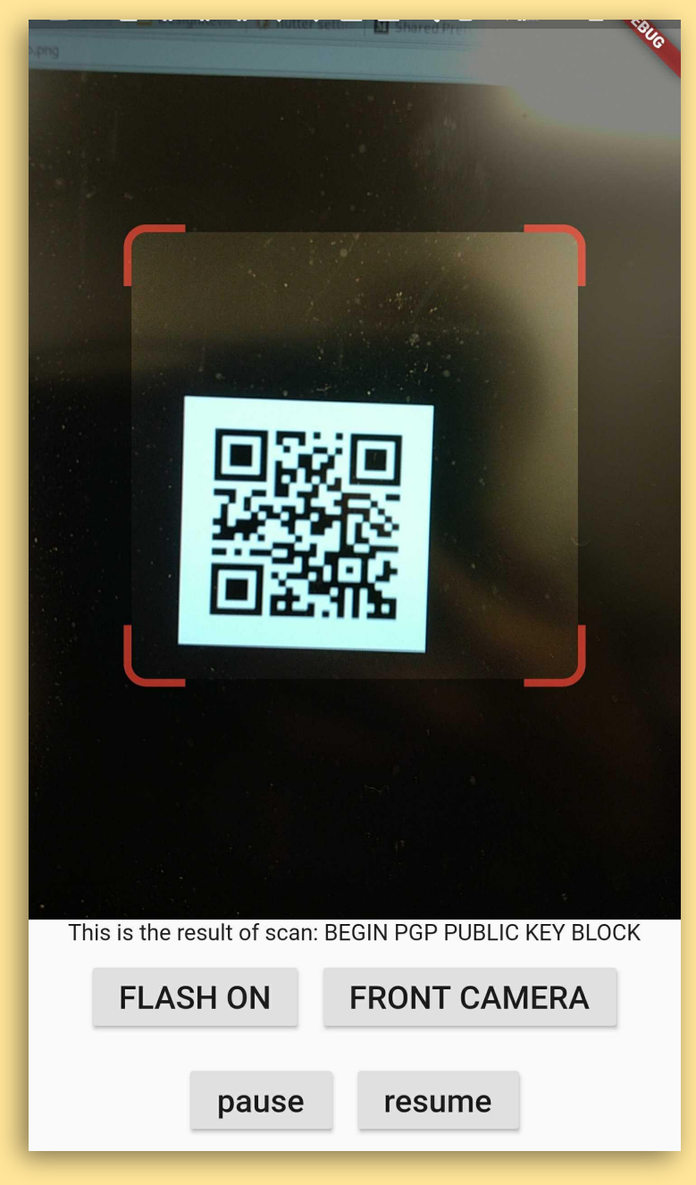
This requires a method for the generation and pre-sharing of cryptographic keys in a completely offline fashion, enabling these keys for use in communication. Our solution to this problem is to exchange keys in person using QR codes.

OBJECTIVE

- Secure framework
- Key exchange outside of IP
- Self hosted server
- RESTful API
- Entirely FOSS

ENGINEERING STANDARDS AND DESIGN PRACTICES

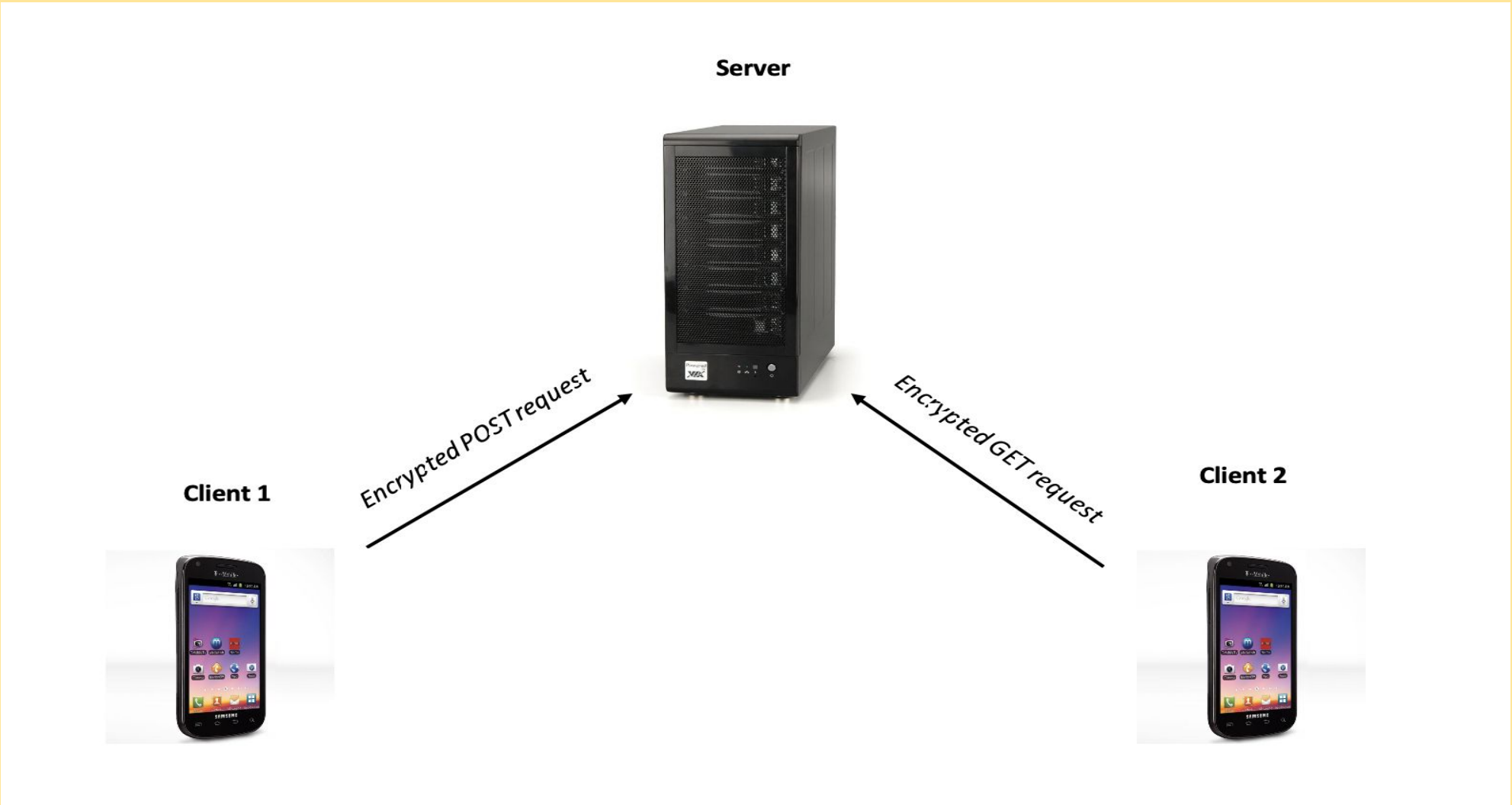
- Git for version control software
- Agile for work distribution and management
- RFC 8017 RSA
- RFC 2818 HTTPS
- RFC 8018 AES
- RFC 4122 UUID


This is the result of scan: BEGIN PGP PUBLIC KEY BLOCK

The first prototype, attempting to read a QR code

TECHNICAL DETAILS

- RSA 2048-bit keys
- AES-GCM 256-bit encryption
- HTTP over TLS using JSON
- RSA PKCS1v15 signatures
- UUID for challenges
- SHA256 hashing
- RSA PEM encoding

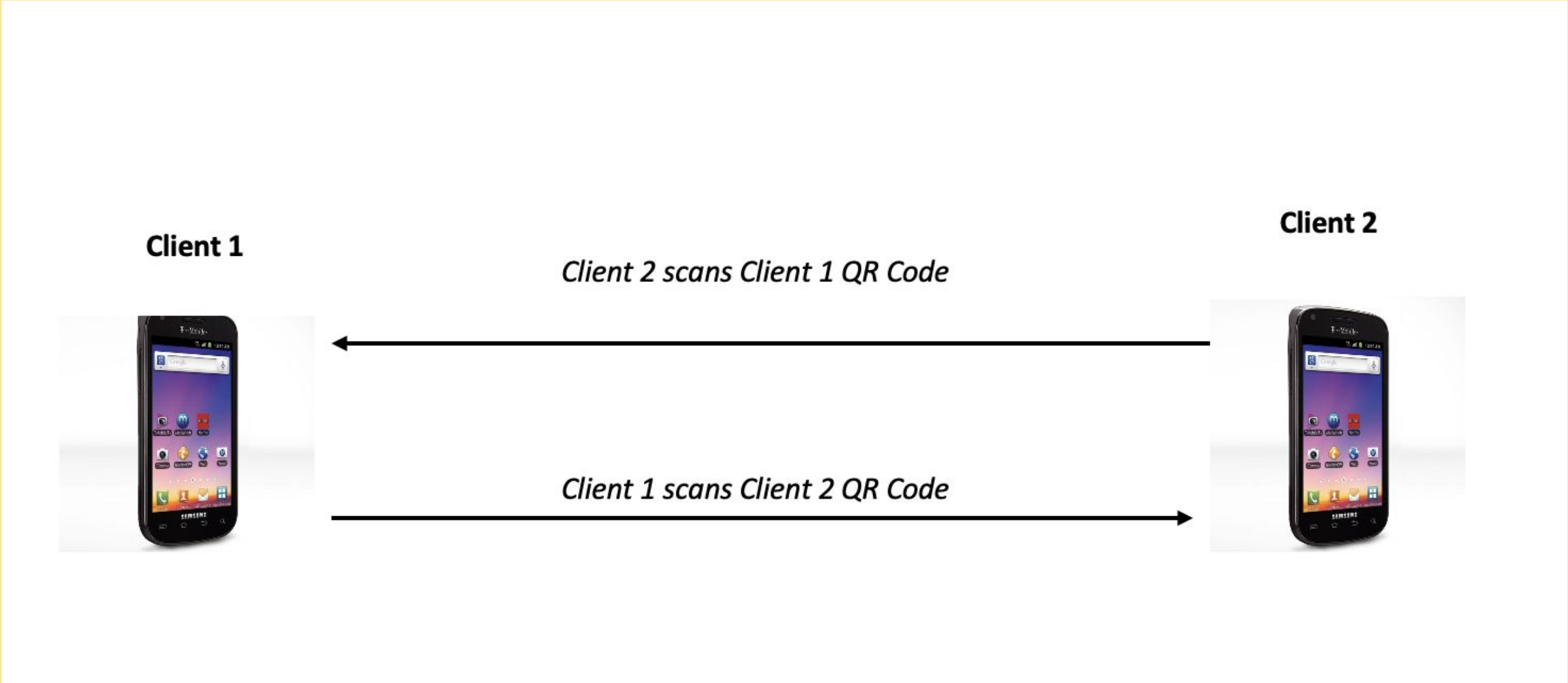


Paranoia
My Key
Paranoia
Generate Key
My Key
Chats

Early concept designs

FUNCTIONAL REQUIREMENTS

- AES keys exchanged over QR codes
- Talks to server over HTTPS endpoints with JSON
- Clients are authorized through the use of RSA signing
- Clients must perform cryptographic challenges for access

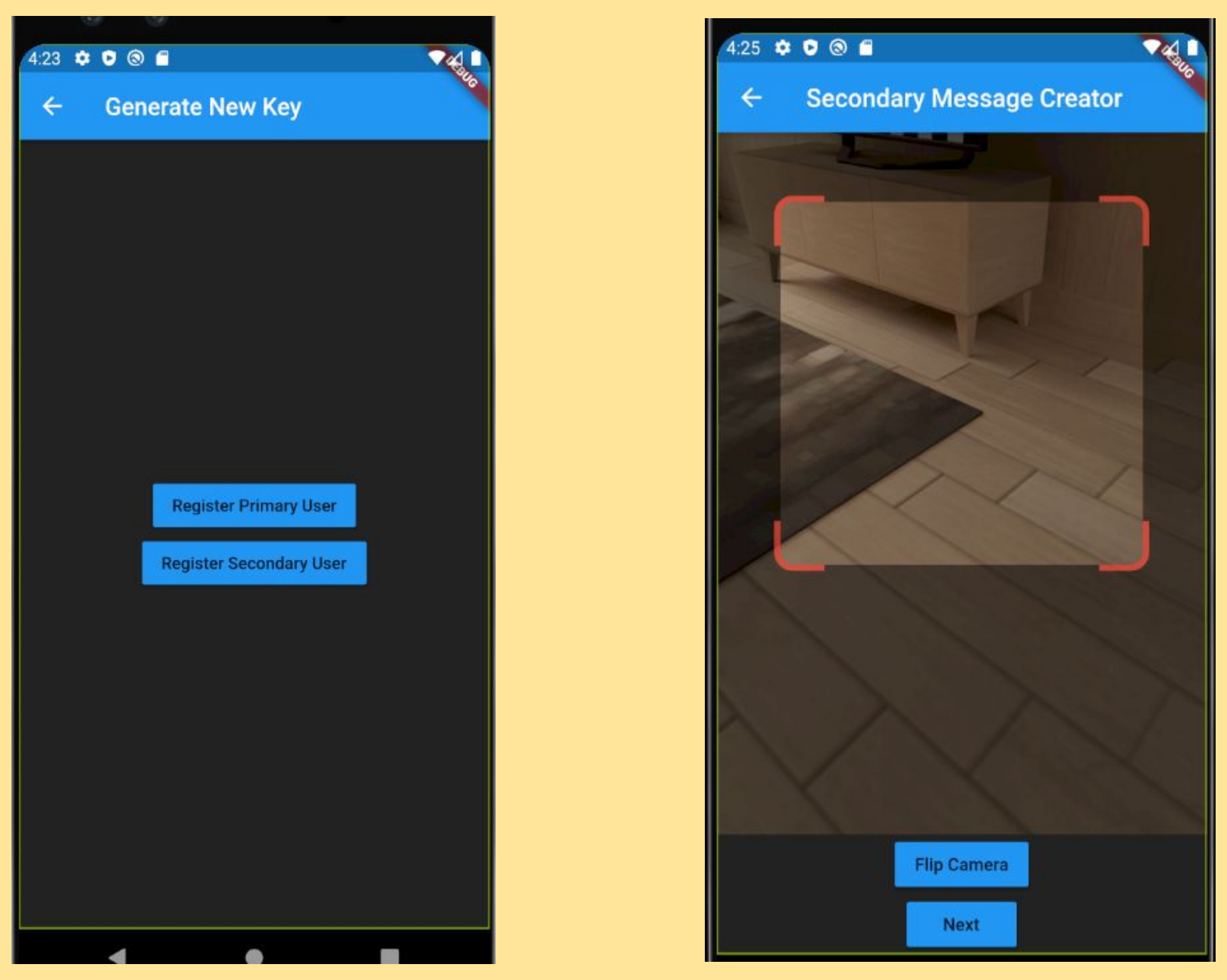

Client 1
Client 2
Client 2 scans Client 1 QR Code
Client 1 scans Client 2 QR Code

NON-FUNCTIONAL REQUIREMENTS

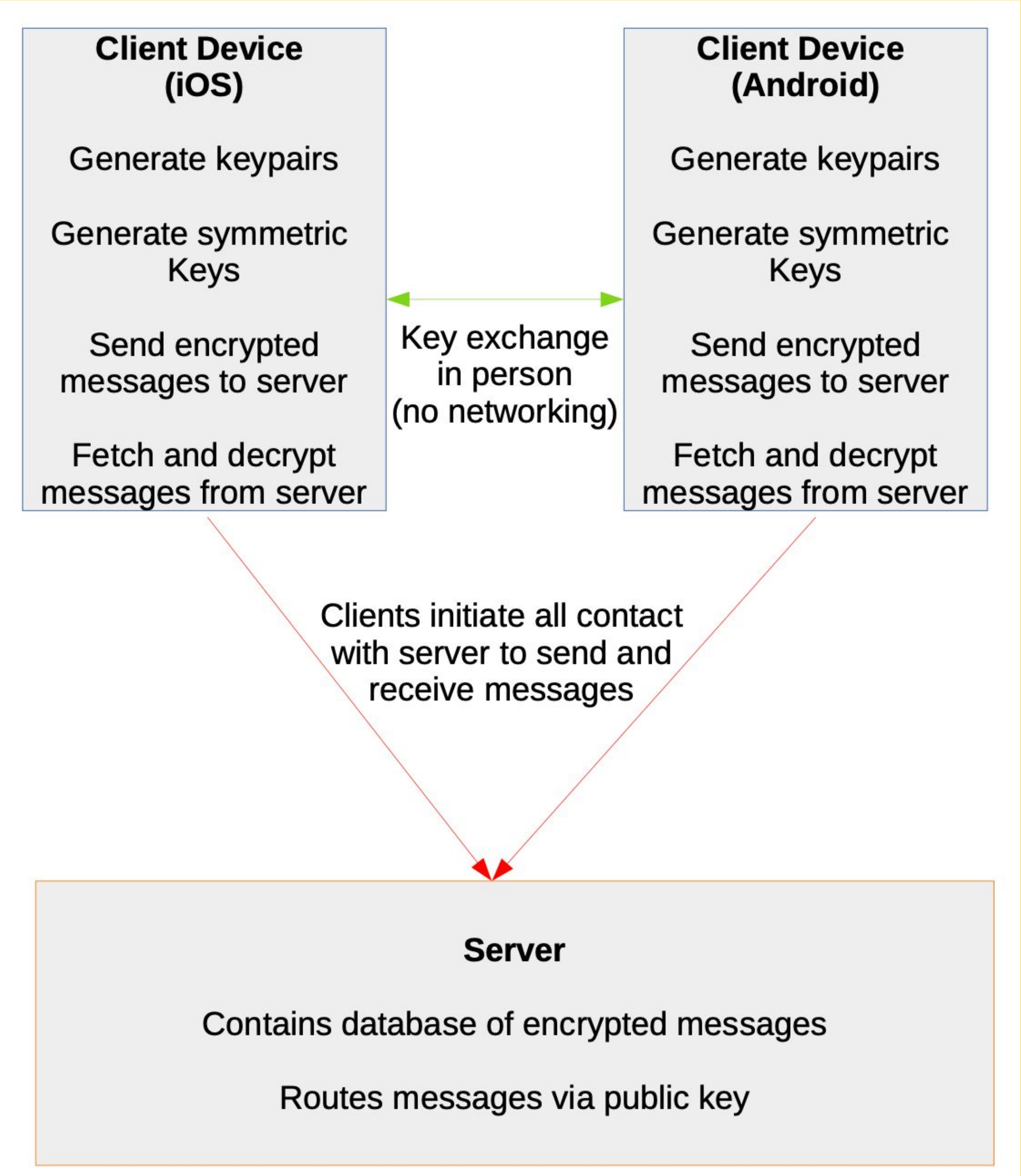
- No PII associated with users for addressing or authorization to the server
- No plain text messages nor method for decrypting those messages passed over IP
- Server must have ability to be self hosted
- Cryptographically safe method for communication between client and server.
- Resistance to Man-in-the-Middle attacks.
- Cross platform clients(iOS and Android)

TESTING

- Server - continuous integration unit testing
  - Entire functionality of all endpoints
  - Possible error scenarios
  - Tests run after every commit and pull request
- Client - rapid manual testing
  - Emulated Android and iOS devices
  - IDE hot reload allows testing after every change



Final Design



Client Device (iOS)

Generate keypairs

Generate symmetric Keys

Send encrypted messages to server

Fetch and decrypt messages from server

Key exchange in person (no networking)

Client Device (Android)

Generate keypairs

Generate symmetric Keys

Send encrypted messages to server

Fetch and decrypt messages from server

Clients initiate all contact with server to send and receive messages

Server

Contains database of encrypted messages

Routes messages via public key