



# Key Exchange Outside of TCP/IP

Team 52, under the advisement of Dr. Julie Rursch:

Andre C

Jacob Moody

Jack Potter

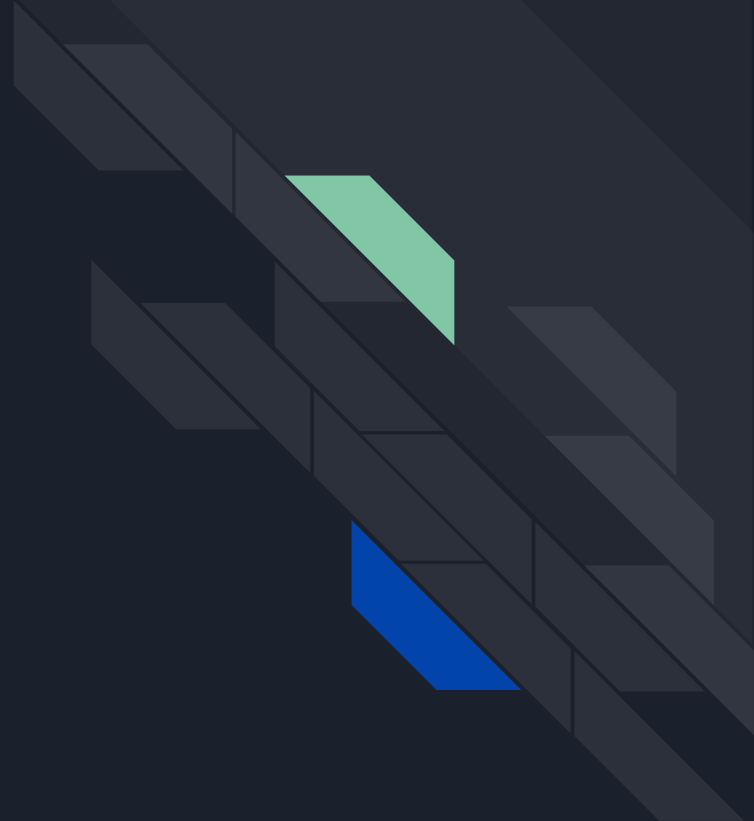
Joel Wacker

Jordan Svoboda

Logan Woolery

<http://sdmay20-52.sd.ece.iastate.edu/>

# Project Plan





# Project Overview

- Cross platform communication with zero trust in network
- Mobile application transfers cryptographic keys outside IP network stack,
- No personal identifiable information (PII) ever associated with users
- Centralized, self hosted server routes messages

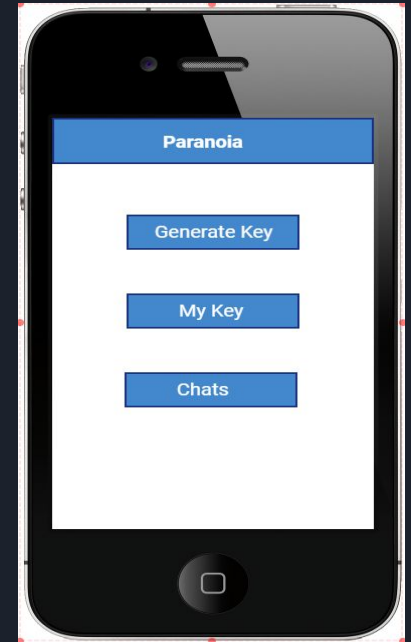
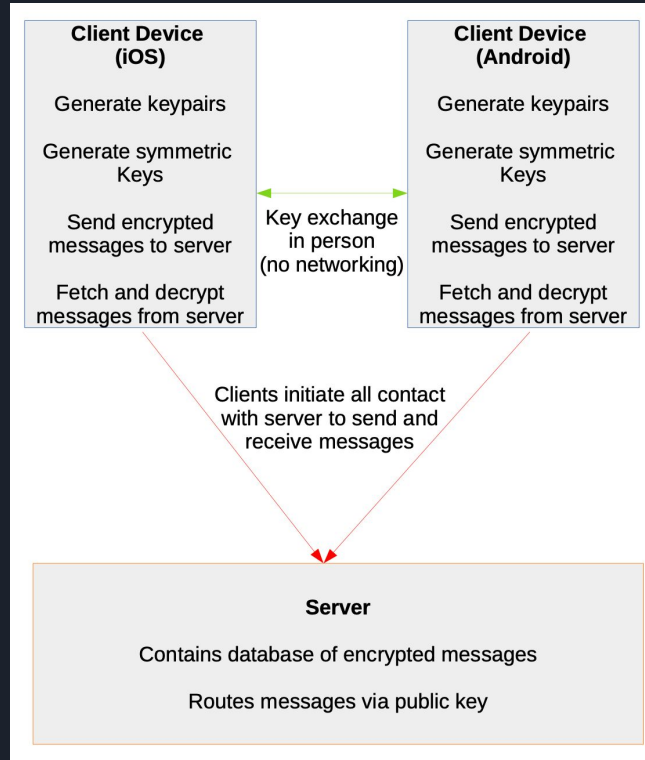


# Problem Statement

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 messages can be communicated in absolute security, regardless 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.

# Conceptual Sketches





# Functional Requirements

- No Personally Identifiable Information (PII) used to index or identify users
- Plain text messages must not be viewable by the messaging router
- “Method” for decrypting and accessing plain text messages must not pass over IP
- Users must be able to administer and run their own variant of the server
- User indexing (how users addresses others) must be cryptographically sound
- Server must not log connections from the client



# Constraints

- Clients may access the server from any source IP, and may become unavailable for large periods of time
  - Avoid synchronous communication
  - Avoid pushing (server -> client) communication
- Users must not be indexed/identified by PII
  - Usernames
  - Passwords
  - Email addresses
  - Device identification
- Server must have self hosted option
  - No closed source/non accessible server components
  - Server doesn't log IP connections of users
- Client and Server must both be open source



# Potential Risks and Mitigations

- Users impersonating others
  - Challenge based authentication (proves public/private key ownership)
  - In-person exchange of QR Codes (lowers social engineering attack feasibility)
- Man-in-the-middle attacks
  - Encryption happens before traversing network
  - Rogue server cannot decrypt messages sent or received





# Resource and Cost Estimates

- Mobile platforms to conduct testing with
  - Need both iOS and Android devices
- Computer in which to design and test program
  - Mac required for testing and pushing iOS application to devices
  - Must be capable of running flutter SDK
  - Must be capable of using git
  - Must be capable of compiling and running go programs
- Publicly addressable server computer with capability to run server



# Project Milestone and Schedule

12/06/2019: Working client prototype

1/18/2020: Successful message encryption

1/25/2020: Communication between two users

2/10/2020: Usable beta version of application

2/24/2020: Group chat fully functional

3/1/2020: All application functionality implemented

# System Design





# Functional Decomposition

- Graphical front end application
  - Generate keys and be able to transfer them through a system that is not over IP
  - Handle encryption/decryption and signing/verification of messages that travel over the network
  - Generate user fingerprints(addresses) and keep an index of them(contacts)
  - Provide a way for users to specify a specific backend for use
- Backend
  - Must provide an endpoint to allow clients to register and communicate with others
  - Must ensure authenticity and authorization of client for performing actions
  - Must keep messages on for devices to retrieve



# Detailed Design

- Backend server exposed a number of RESTful HTTP endpoints
  - Data structures are communicated over JSON
- Users are identified through the use of a RSA public/private key pair
  - Users register with a server using their public key and a signature of that public key to prove that they own the corresponding private key
- Check authorization and authenticity of clients
  - Clients are expected to pass a “challenge” in which a UUID string is give that they must sign with their private key for making actions with their corresponding public key
- Security of messages sent to and from the server
  - HTTP messages are encrypted using TLS
  - User messages are encrypted using AES, with the key being exchanged over QR codes



# Technology Platforms Used

- Flutter
  - Cross platform toolkit for using Dart
  - Parallelize Android and iOS development
- Go
  - Cross platform language
  - Specializes in network applications
- Github Actions
  - Continuous integration platform
  - Executes unit tests automatically
- Git
  - Version control
  - Used to manage multiple changes made to project



# Test Plan

- Server unit tests - continuous integration system
  - Entire functionality of each end point
  - Possible error scenarios
  - Run on each commit and pull request
- Manual client testing
  - Android and iOS applications
  - Emulated Android and iOS devices
  - Hot reloading - instantly test every change
- Large scale beta testing
  - Planned, not completed due to pandemic
  - Distribute applications for use/penetration testing
    - Cyber Defence Competition participants
    - Friends, classmates, coworkers

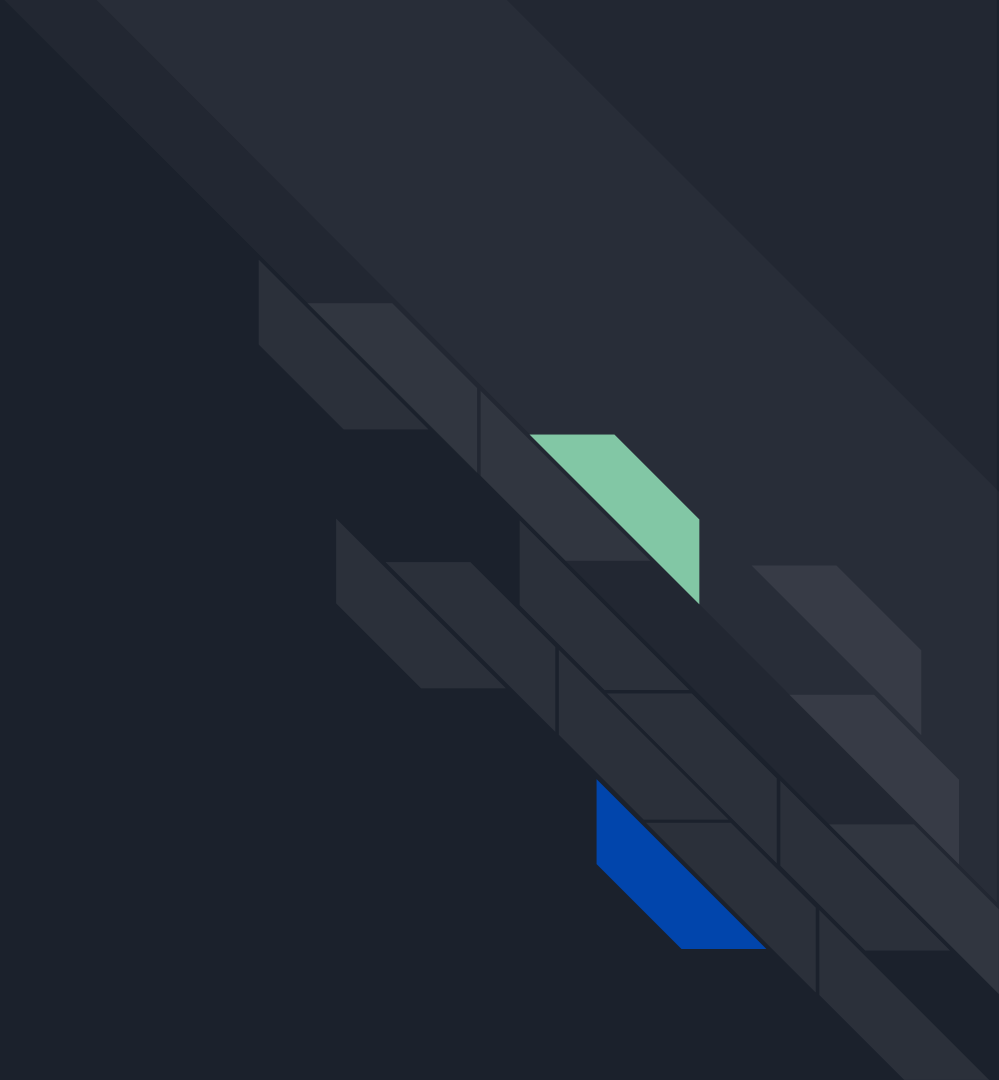


# Prototype and Building Block Implementations

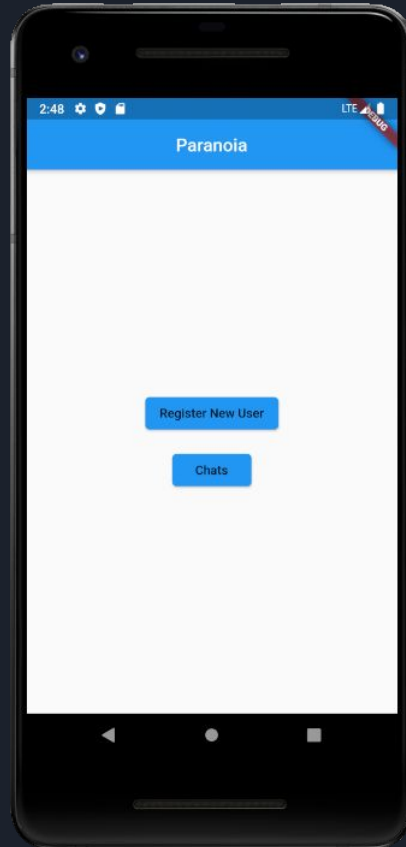
- Original implementation was using PGP with RSA being the backing cryptographic algorithm
  - We found PGP was overkill and difficult to implement
  - Switched to pure RSA signing
- Transferring keys using QR codes resulted in unexpected integers as an outcome
  - QR codes with several hundred characters of text would not scan correctly
  - During testing, we found QR codes of capitalized text had the least issues
  - Keys are converted to base32 before converted to QR codes



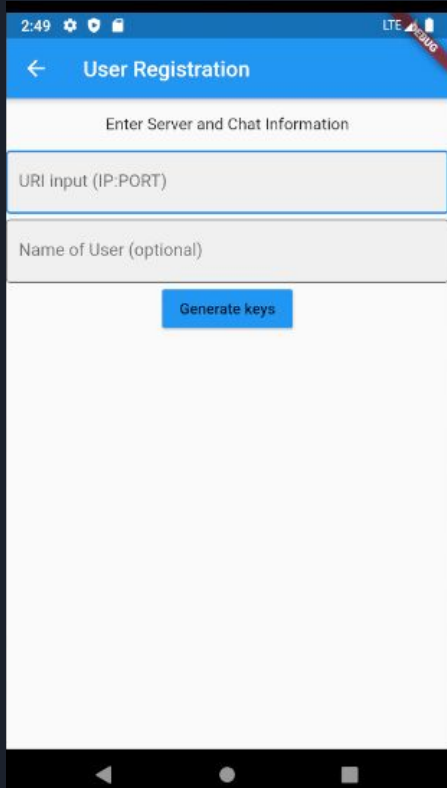
Demonstration



# Main Screen



# Bob Fills in Server Data, Generates Keys



2:49 LTE

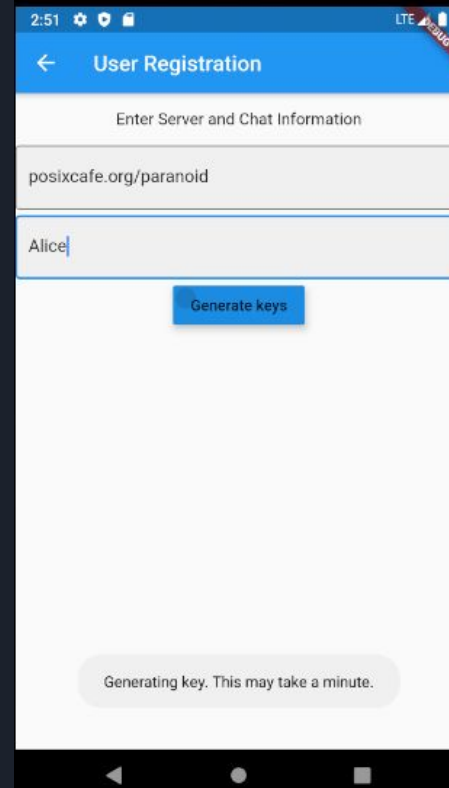
User Registration

Enter Server and Chat Information

URI input (IP:PORT)

Name of User (optional)

Generate keys



2:51 LTE

User Registration

Enter Server and Chat Information

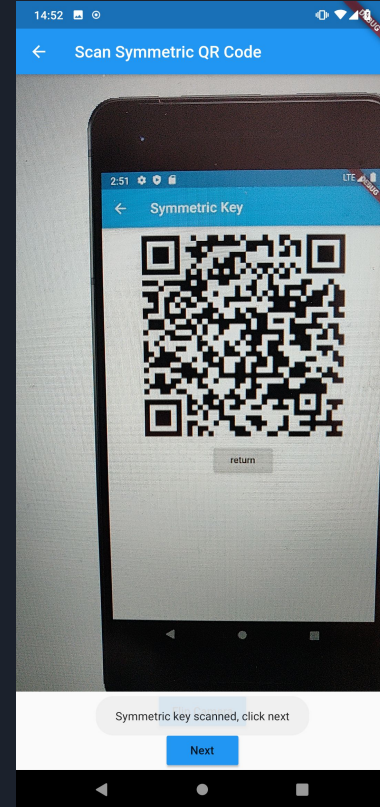
posixcafe.org/paranoid

Alice

Generate keys

Generating key. This may take a minute.

# Bob Displays Symmetric Key, Alice Scans It



# Alice Enters Server Info, Generates Keypair

14:53

← Secondary User Data Collection

Enter Server Information

posixcafe.org/paranoid

bob

ÿÿÿxàÆÿÿ¼4ÿÿÔ8<1ÿÿ%?ùlèÿÿÙ4û ,4ÿÿzÿÿ

Save Info

14:53


← Create a Group

Create a Group

Public Fingerprint of Primary user

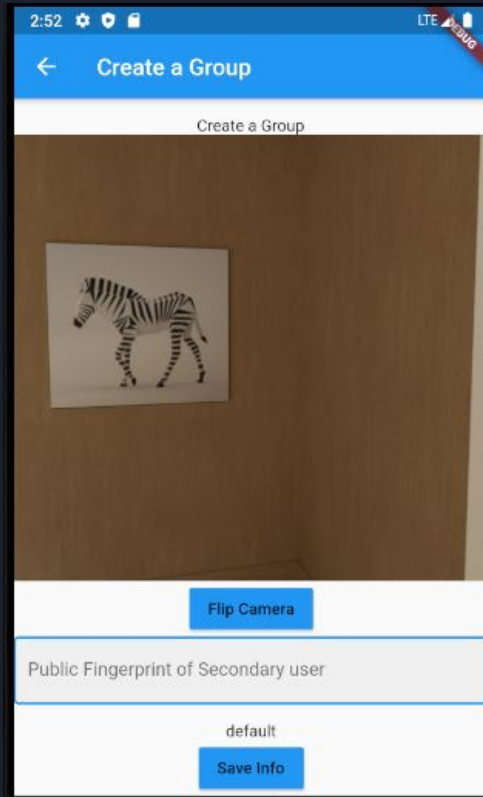
Scan QR Code

Save Info



Public key:  
5defadeed35b199ef9c8e997b5b5cdbea8858302d9bd7dee07c3542819190dc8

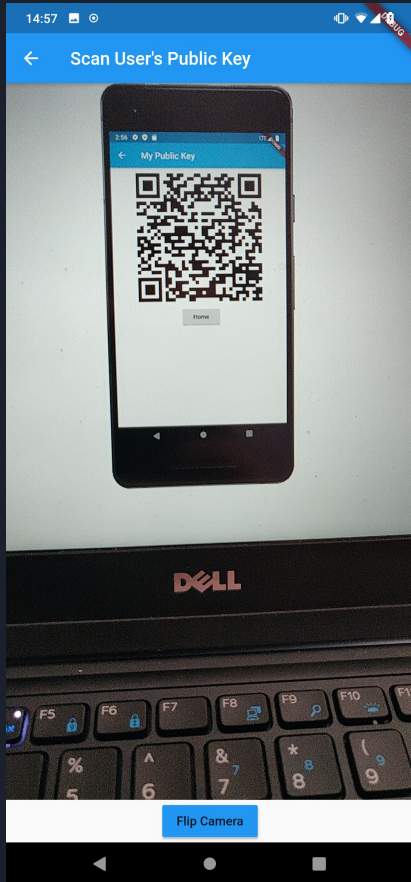
# Bob Scans Alice's Public Key



# Bob Displays Public Key

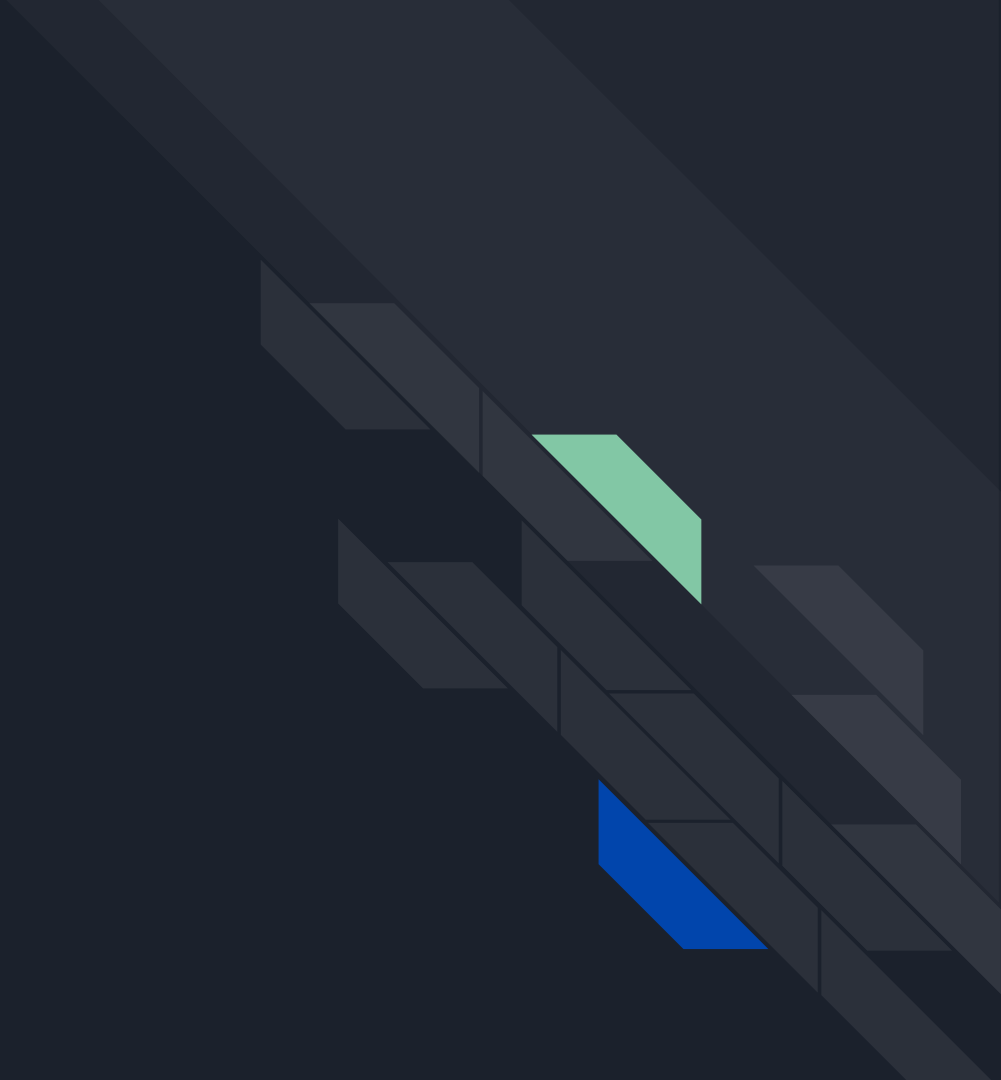


# Alice Scans/Saves Bob's Pubkey





Video Demo





# Engineering Standards and Design Practices

- Developmental Design Practices
  - Cryptographically secure random numbers
    - GroupID
    - Symmetric key
  - Version control
  - Encryption of data upon rest
- Engineering Standards
  - RFC 8017 RSA
  - RFC 2818 HTTPS
  - RFC 8018 AES
  - RFC 4112 UUID

Conclusion





# Member Contributions

Andre C

Frontend:

- Networking API
- Application compatibility testing
  - iOS/Android design
  - Front/backend communication
- UI Flow design and development
- Network Security Testing
- Database schema drafting

Jacob Moody

Backend:

- Unit testing
- Continuous Integration
- Design of API endpoints
- Design of challenge system
- Original PGP design
- Pure RSA rework
- Backend deployment
- Logging and performance testing



# Member Contributions (Cont.)

Jack Potter

Frontend:

- Client functional testing
- Flutter Android Application Developer
  - Navigation
  - QR code generation
- Code review
- Library research/integration

Joel Wacker

Frontend:

- Flutter Android Application Developer
- Client to Client implementation
- Client hardware integration
  - Camera access for iOS/Android
- QR Code Scanner for key sharing
- Physical Phone Tester



# Member Contributions (Cont.)

## Jordan Svoboda

- Client side implementation of crypto functionality
  - Symmetric key generation and storage, encryption, decryption
  - Asymmetric key pair generation and storage, message signing and verification
- Database implementation for Chats and Messages
- Implementation of Chat/Messaging interface on the client

## Logan Woolery

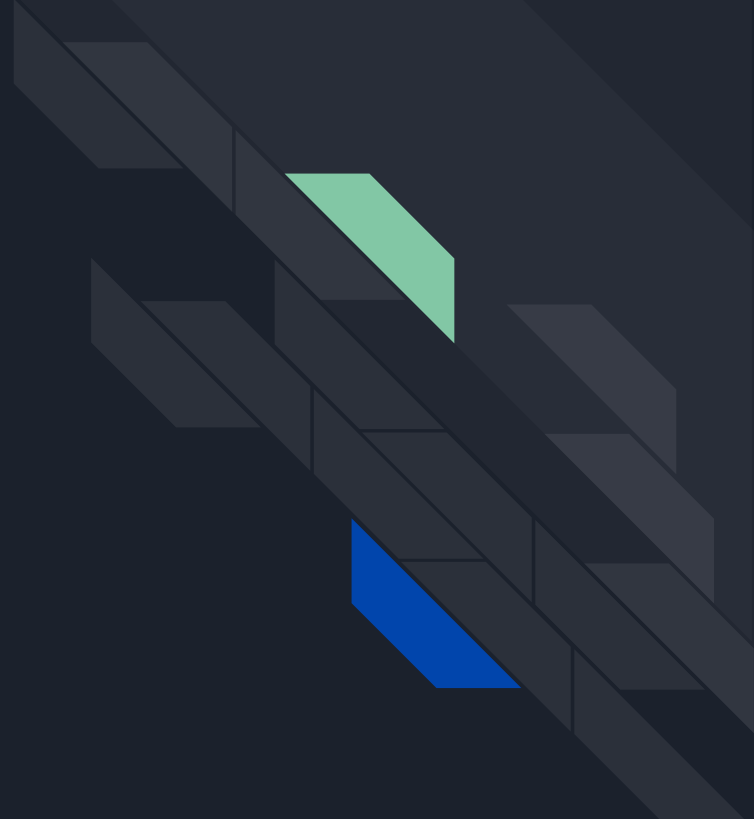
- User Interface development
- UX management
- Integrations testing
- CI/CD
- Automated testing development



# Future Prospects

- Method for verification that server is who they claim to be, and connection is not being MITM'd
  - While being MITM'd does not expose actual messages it may allow disclosure of who is messaging who at what times
- Support for other non IP key transfer methods
  - Use of audio to send and receive information
  - Use of camera flashes to send and receive information
- Federation between different self hosted server applications
  - This would allow two users using two different self hosted servers to communicate

Thank You  
Dr. Rursch!





Questions?

