Daisy Barbanel, Davaajav Ganzorig,
Miranda Grisa, Connor Hopcraft, Troy Hu,
Ryan Li, Gray Mackall, Zené Sekou

Final Design Review

- **See Milestone 4b README or Final Release README on how to install and use the app.**
- **If you want suggested acceptance tests, see Milestone 4b README.**
- **Moreover, the Final Release README's tutorial can also guide you on creating your own acceptance tests.**

Source Code Checkpoint:

We've created a final release branch for this submission specifically. Checking out this branch will provide access to the full-featured build of the smart and secure messaging app. Its information is provided below:

Branch Name: **master** OR **final-release**
final-release URL:
https://github.com/ze-ne/cs220-smart-and-secure-messging/tree/final-release
master URL:
https://github.com/ze-ne/cs220-smart-and-secure-messging

How to Install and Run Unit Tests

- How to Install and run app
  - If you just want to install the app, you can follow the "Final_Release_README.pdf" which involves downloading an APK.
  - You can still use milestone 4b's instructions to run the app as well. The github branch you pull must be **master** or **final-release** (link to repository found at end of document).
- Unit Tests
  - Same instructions as milestone 4b but with github branch **master** or **final-release**

Note Regarding Network Connectivity:

Use of this app assumes an internet connection, either through WiFi or mobile data. Though a lack of connection will not cause the app to crash, basic functionality such as logging in, starting conversations, sending messages, and searching for users will not be available. That is,

behavior will differ somewhat from the acceptance tests' suggested/correct behavior. Offline behavior is detailed below:

Adding a Conversation

- If the other user's data has been cached:
    - The conversation will appear on the conversation screen, but will not be added to the server or the recipient's device until network connection is restored.
        - *However:*
            - If the user leaves either the user search list or start conversation dialog (text box to input userId) before internet connectivity comes back, the current user will not be redirected into the conversation after clicking "Start Conversation".
            - If the user does not leave the search list or start conversation dialog before internet connectivity comes back, the user will still be redirected to the conversation.
- If the other user's data has not been cached:
    - A "user not found" error will be displayed.

Logging In

- An error message will be displayed and the user will be unable to log in.

Sending a Message

- The message will appear in the sending user's conversation, but will not be delivered to the database or sent to the receiving user until the network connection is restored.
- If the app is closed before network connection is restored:
    - The message will be sent when the app is reopened, assuming network connection at the time of reopening.

Searching for a User

- No search results will appear.

Blocking a User

- Nothing will happen. The blocked user will be added to the block screen when connectivity is restored.

Note Regarding Tonal Analytics:

The secure functionality uses an IBM tonal analysis service, for which we are subscribed to the free tier. This means we only have 2,500 available calls to the IBM API, after which it will stop sending data. We don't anticipate hitting this cap, but be aware if you choose to run a high volume of tests on the tone analytics. Contact us if you run into this problem.

Key Changing Due to Reinstallation Behavior (Noted in Milestone 4b)

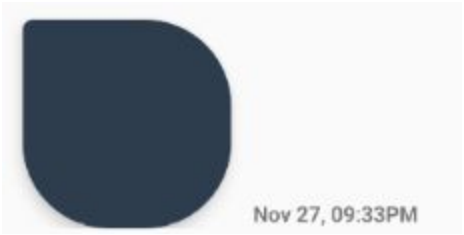
Background

When a user is created, a key pair is generated and stored on the local device. Note that if a key pair is already on the device, the user uses that key pair. The public key of the key pair is also uploaded to the database with the user as the owner.

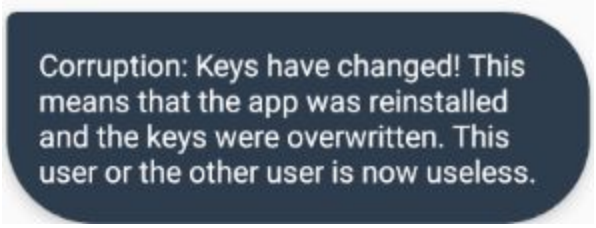
When you do a major reinstall or delete the app and then reinstall, the key pair files are deleted. This means that the next time you log into the device, new key files will be generated for you completely different from the old key files. However, the keys will not be updated in the database.

After this key change, whenever you receive a message, the encrypted message assumes that your private key is the OLD private key and NOT the new private key. This is because the message was encrypted with your OLD public key. Thus, the message is not decryptable for you and decryption will throw an exception if not caught (we catch such exceptions).

If this happens, the user sees in their messages screen:
Image Message



Text Message



What we are doing now

After discussing this particular behavior with the TAs, we were told that it is fine to not make changes to this behavior. Our main reasons for not changing this behavior are:

- For Facebook messenger, if you delete and then reinstall the application, any previous encrypted conversation is wiped clean because the local keys have changed.

- We originally envisioned that in ideal conditions, you should not be able to reinstall the app and use the same user on the newly installed app. Moreover, in our opinion, uninstalling and reinstalling the application is equivalent to installing the app on another phone and using an already created user.
- We also thought that it would be a big change to the database in order to allow for users to keep using the same account after reinstalling. Both the DB and Client need non-insignificant modifications.

Use-Cases Implemented:

From Milestone 3a

User

- Login
- Logout
- Register
- Manage Account

Conversation

- Start Conversation
- View Conversation
- Send/Receive Text Messages
- Send/Receive Image Messages

From Milestone 4a

User

- Search for User
- Manage Block List
- Change First and Last Name

    Listed as 'end to end manage information' in the 4a README.

Conversation

- Search for Conversation
- Delete Sent Message
- Delete Conversation
- Drag to View Part of Hidden Image Message
- Send Message With Destruction Timer
- Send 'Click to Reveal' Text Message

    Listed as 'drag and reveal' in the 4a README.

Security

- End to End Encryption

- Screenshot Message Prevention

Tone Analytics

- "Smart" Tone Analytics

Use-Cases Not Implemented or Changed:

Proposed in Milestone 3a

Contacts

As development progressed, we realized that all the functionality we wanted to implement through contacts was already available to the user through their list of conversations. From the main conversation tab, users can see the users with whom they have communicated, send messages to those users, or block those users. With this already in place, there was no reason to add contacts as a separate feature.

Proposed in Milestone 4a

End-to-End Contact Management

See *Contacts* above.

Send Image with Picket Overlay

Adding a picket overlay proved both difficult to implement and ineffective as a means to securely view an image. Instead, we implemented a press-to-reveal feature where the image is sent completely hidden, and the recipient can press regions of the image to temporarily reveal it. This way, the image can remain invisible until the receiving user is free from onlookers, at which point they can choose to view it privately. This guards more effectively against screen watching than the picket would have.

Proposed in Original Design Document

Group Messaging

We decided that group messaging would have been too time consuming, and that as a feature, it was nonessential to the core vision of a smart and secure messaging app. We felt that development time would be better spent fleshing out the security features which give our app a reliable standard of privacy.

Use-Cases Implemented But Not Proposed:

Phone Authentication

In order to keep user profiles secure, we decided it was important to have a reliable registration process by which we could ensure that no user could register another person without their permission. We implemented phone authentication to solve this problem; when users register now, their account data is linked to their phone number. The number is

verified to be theirs through a passcode sent to that phone number at the end of the registration process.

Bonus Use Case (Push Notifications - NOT PART OF FINAL SUBMISSION):

In addition to our specified final submission feature above, during testing and debugging after Milestone 4b, the server team found they had the bandwidth and technical capability to add a new push notification feature for notifying users of new messages sent to them in real-time, as we figured this would provide a minor user experience improvement. However, since this was implemented **for fun** after our Milestone 4b submission (wherein we could only submit small changes afterwards), we decided to put this moderate change of a new separate feature into its own separate branch ('notifications') **that is not the same as the final-release branch**. It is important to note that the code in this branch is there for demo purposes for those curious, but is otherwise **not going to be part of our final submission for grading/testing**.

To see this feature in action (assuming the appropriate source code is pulled and built), simply have another user send you a new message while your app is closed (i.e. in the background, as is the typical use case for basically all messaging clients with push notifications enabled) and wait for the short receipt of the new push notification (which should say something like 'user xxx has sent you a new message'). Since push notifications are generated server-side and the messages are end-to-end encrypted, the push notification will just notify the user who sent the message but not display the message itself, which follows our paradigm of usable security/privacy.

Bonus feature source code:
https://github.com/ze-ne/cs220-smart-and-secure-messging/tree/notifications

Member Contributions:

Frontend Team
Zené Sekou

- Conversations list including adding and deleting conversations and searching
- Conversation screen and functionality within it including sending text messages, sending text and image messages with overlay, deleting messages, the deletion timer set by user, and some of the Smart display
- Login, registration, and phone authorization through Firebase
- Home activity
- Required adapters and xml files for above screens

- General UI design and formatting

Miranda Grisa

- Settings, Contacts, Screenshot Prevention, Smart Tone Toggle, Connecting Various Back and Front Ends To Each Other

Daisy Barbanel

- Block list UI including adding and removing blocked contacts and searching
- Sending and displaying image messages from UI (without overlay)
- Get analytics toggle: populating the messages with individual analytics and displaying general analytics
- Storing sent messages in firestore database
- Unit tests for delete message and delete conversation
- Deletion toggle

Backend Team

Troy Hu

- Encryption/Decryption of sent messages
- Device, Conversation (most of it), and ImageHandler Classes
- Parts of the User class/Database handling
- Wrote tests for everything I implemented
- Refactored/made fixes to existing backend or server code
- Minor refactors/changes to existing frontend code in order to fit code

Connor Hopcraft

- Worked primarily on client backend functionality, including local data manipulation, message sending, receiving, deletion, timed deletion, as well as fetching and updating user data from server (mostly in User.kt)
- Wrote listeners on the frontend to support real-time updates for messages and conversations
- Added smaller edits to other frontend code and backend message class to support this
- Most unit tests for *User.kt*

Ryan Li

- messages and blocklist testing and functionality
- testing

Server/Smart Team

Davaajav Ganzorig

- Watson api functionality
- Image deletion timer
- Deletion on database
- Image storage on the database

- Conversation listeners for receiving new messages
- Helped with encryption
- Database functionality and design

Gray Mackall

- Worked on Watson api functionality to get sentiment analysis for each message in a conversation, as well as unit tests for related functions and setting up associated IBM cloud platform account
- Also worked on functions that interacted with firestore database for conversations and users
- Helped connect front and back end during first iteration