Team Apollo Release 1 Summary

Team members

Name and Student id	GitHub id	Number of story points that member was an **author** on.
Daniel Yoo 26623875	Dan-Yoo	8
Erin Benderoff 27768478	erinb90	6
Gabriel Harel 40006459	g-harel	6
Feken Baboyan 27754507	veken1199	8
Shawn Elbaz 27754132	shelbaz	5
Radu Raicea 27684037	Radu-Raicea	5

Mobile App summary (max one paragraph)

Many people tell us that one of their biggest concerns is that they don't know if the governments or third parties are spying on our messages and storing them to collect data. So we wanted to put your minds at ease by creating the most bulletproof end to end encrypted messaging application out there to date. We created an easy to use, lightweight and secure application that can encrypt and protect all your messages, without the need to learn anything about encryption. How did we do this? By using such an advanced encryption algorithm, we needed to make it ourselves from scratch. It's called the Signal Protocol, and its so good, it's already used by the biggest messaging applications in the world, like Whatsapp and Skype! It works right out of the box with all your contacts, no setup required!

Velocity

Total: 6 stories, 38 points over 4 weeks
Sprint 1 (3 stories, 17 points) over 2 weeks
Sprint 2 (3 stories, 21 points) over 2 weeks

Plan up to next release

Total: 8 stories, ~66 points, over 5 weeks
Sprint 3 (8 stories, 36 points)
Sprint 4 (1 epic, ~30 points)

Overall Arch and Class diagram

All the diagrams can be observed in the **wiki**.

Signal Android is split into 3 main layers: Presentation, Business, and Data. The presentation layer is composed of **activities** that represent pages in the application (one screen). Activities can be further broken down in multiple **fragments**. Both activities and

fragments contain UI elements called **views**. The elements from the presentation layer are usually defined in XML files. The business layer has all the **services**, **managers**, and **listeners**. These take care of performing the business logic and associating UI elements to actions. The data layer has the **loaders** that loads data from the database. The database has its tables modelled like the **models**. There are other components like the Android API and the Signal API that interact with multiple layers as well.

Infrastructure

Android Support Library

RecyclerView and RecyclerView.Adapter:

We are using this User Interface Layout from the Android Support Library because it provides the ability to load only the data that is being displayed instead of loading the entire dataset which would be heavy as the number of pinned messages grow. As well, whenever the user scrolls his pinned messages view, each view that is scrolled off-screen doesn't get destroyed but recycled and populated with the new data that is entering the screen. This provides an efficient solution when wanting to display large data sets. The only drawback is the complexity of implementation.

Alternatives that were rejected:

The alternative to the RecyclerView is the ListView. RecyclerView is much more powerful, flexible and an upgrade to the ListView.
Reasons are:
-provides flexibility in choosing a LayoutManager.
-supports item animators/decoration compared to the lack of support in the ListView
-supports on item touch listener

Overall, the RecyclerView provides much more flexibility and freedom compared to the traditional ListView.

Testing Libraries:

Mockito and PowerMockito:
Those both testing frameworks are used exchangeably to mock and create stubs of the methods and the classes that were implemented in order to extend the functionality of a number the application interfaces. It is crucial to use such frameworks in Signal since each component is coupled with many other interfaces and therefore using the traditional Unit Tests becomes hard to maintain and develop. Mockito helps to mock the interface of a class and reduces the cost of creating new instances of the mocked classes for the testing purposes. It allows to focus on the unit and the logic flow through that unit after mocking the coupled interfaces.

PowerMockito is another great library that helps the developers to mock the static methods of a type. Both those frameworks were used during the testing process.

Stetho Facebook:
This is another great framework that helps us to bridge the running application in a debug mode with Google Chrome inspect tool. It helps to visualize the structure of the internal storage units such as local storage, cookies, and databases. Furthermore, it helps the developer to explore and inspect the structure and the behavior of the user interface attached to the running activity. It was intensively used during the testing and the development phases.

Code

File path with clickable GitHub link	Purpose (1 line description)
ConversationFragment.java https://github.com/Radu-Raicea/SignalAndroid/blob/master/src/org/thoughtcrime/securesms/ConversationFragment.java	Displays the correct toolbar icon (pin or unpin) on long-clicking message and calls the appropriate handler method to pin or unpin the chosen message.
PinnedMessageAdapter.java https://github.com/Radu-Raicea/SignalAndroid/blob/master/src/org/thoughtcrime/securesms/PinnedMessageAdapter.java	Creates the view for each pinned message item and binds data to it, which includes the message body, sender name and timestamp.

Testing and Continuous Integration
Each story needs a tests before it is complete. If some class/methods are missing unit tests, please describe why and how you are checking their quality. Please describe any unusually aspects of your testing approach.

Testing our stories was a major pain point for this sprint. The Signal project only had unit tests for a few utility functions and none for the layout. This means the necessary infrastructure to properly write and implement any tests was not there. We needed to fight against the gradle build tools and the android ecosystem at every step of the way, especially since our team is not yet familiar with the android platform. In the end, we have implemented some very surface level tests as a first step in the right direction.

Test File path with clickable GitHub link	What is it testing (1 line description)
testPinMessagesMethod( ) : PinnedMessagesTest.java https://github.com/Radu-Raicea/SignalAndroid/blob/master/test/unitTest/jav	Tests that behavior of the actual pin messages methods work correctly

a/org/thoughtcrime/securesms/database/PinnedMessagesTest.java	
testHandlerPinMessageSmsFailureOnAlreadyPinned( ) : PinnedMessagesHandlerTest.java https://github.com/Radu-Raicea/SignalAndroid/blob/master/test/unitTest/java/org/thoughtcrime/securesms/pinmessages/PinnedMesssagesHandlerTests.java	Tests the scenario of trying to pin a message that is already pinned, therefore showing a true value for that message column in messaging DB

https://travis-ci.org/Radu-Raicea/SignalAndroid

Our continuous integration environment is built on the best practices of the industry. We use the industry standard for revision control on for our code repository, so it lies on Github. Every single commit is self building, since it is integrated directly with TravisCI . TravisCI runs the build through gradle, as well as all unit tests to ensure that the build is self testing, and performs as it should in a developer environment. This environment configured through gradle is a clone of the production environment, ensuring consistency throughout the development process. To ensure that everyone is running the same build, even as they are making changes, everyone commits to the baseline frequently, usually at least once per day. Pull requests cannot be merged unless 2 conditions are met, the first being that the TravisCI build has passed, and the second is that at least 2 people have approved the build, ensuring the code is always tested and peer reviewed before it gets merged to the production branch. However, the one aspect that we need to work in the next releases for the continuous integration to go smoother, is to reduce the time it takes to build down from an average of 20 minutes, so that mistakes can be found quickly and fixed.