Key Exchange Outside of TCP/IP

Design Document

Group: sdmay20-52

Client/Faculty Advisor: Julie Rursch

Jacob Moody
Jack Potter
Andre Chickering
Jordan Svoboda
Joel Wacker
Logan Woolery

December 8rd, 2019

Executive Summary

Development Standards and Practices Used

Summary of Requirements

This project has fairly minimal hardware and software requirements necessary to produce our final product. The hardware requirements for this project consist of computers on which we will perform development work, a server that will host the backend infrastructure for our project, and several cell phones on which the application we develop can be deployed for testing purposes. The software requirements for the development of our project can be broken down into the integrated development environments (IDEs) necessary to build and deploy our application as well as the infrastructure software required for deploying and testing our backend code including a hypervisor and firewall.

The computers necessary for the development of our project will primarily consist of our own personal laptops and desktops, which will be largely be sufficient. However, we have found the need to source an additional Apple laptop/desktop of any kind, as there are numerous issues preventing the development of iOS applications on anything other than a computer running OSX. The server hardware necessary for hosting our project's backend will also likely consist of our own server(s). While we initially approached the university to request a virtual private server or a world-routable IP address that could forward traffic to a server we would provide, we encountered difficulties that rendered pursuing the issue untenable. Instead, we will simply use our own servers which already have hypervisors installed that will make the deployment of a new virtual server very straightforward. Finally, between the members of the group, we have a selection of various iPhone and Android cell phones, which will allow testing across numerous platforms.

All of the software required for the development of this project is free and/or open source, eliminating software costs as a constraint on our project. The Android Studio and XCode IDEs will be used for the development and deployment of the phone applications, and team members will be free to utilize their own editors and workflows for the rest of the project.

Applicable Courses from ISU Curriculum

We will draw from our experiences in the following courses throughout the completion of this project:

- Computer Science 309
  - Software development best practices
    - Version control using Git
    - Continuous integration using Git/Jenkins
    - Test driven development with unit testing
  - Android application development
    - Use of Android Studio IDE
    - Integration of external Java libraries
  - Client/server communications

        * Restful APIs
        * WebSockets
        * Client login and authentication

- Computer Engineering 308
    - Low-level coding practices
        * Multithreaded application development
        * Thread-safe programming
        * Understanding of Unix system calls
        * Network communications at low levels

- Computer Science 228
    - Data Structures
        * Storing messages, conversations, users efficiently
        * Ensure messages and conversations can be easily searched
        * Store key hashes efficiently
    - Algorithms
        * Searching conversations/messages
        * Connecting users for conversations
        * Generating/sharing keys quickly

- Computer Engineering 234X
    - Ethical Considerations
        * Ensuring we have no access to user messages/keys
    - Legal Considerations
        * Exporting cryptography suites internationally
        * Responses to law enforcement requests

- Computer Engineering 430/530
    - Network protocols and security
        * Network stacks
        * Protocol weaknesses and securities

New Skills/Knowledge acquired that was not taught in courses

The two primary technologies that we will use that have not been taught in a class before are Golang and Flutter. Golang is an extremely versatile language developed by Google that will make up the majority of our backend code. We chose to use Golang for several reasons. First of all, it is very extensible and makes it very easy to implement open source libraries for cryptography and secure network communications that have been thoroughly audited to ensure efficient and secure performance. Second, Golang makes developing test suites in real time very straightforward, and it contains a number of useful utilities to ensure complete coverage of all code by the test suites. Finally, Golang can easily export executable builds of our project that can be run on nearly any

architecture or operating system very easily.

We will be using Flutter for the development of our client applications. Flutter is a Google project that allows mobile applications to be developed once in the Dart language and then built into native executables for both Android and iOS. We chose Flutter as it will ensure that all versions of our client application will feel and work similarly, and while we may need to do some manual tweaking for each platform, it will generally increase the efficiency of our development and ensure all features are consistent across both applications.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Acknowledgement

We are grateful for the opportunity to work on a project envisioned, researched and designed by our group. As we are not professional engineers yet, it is a gift to be able to work with other students who have a similar desire to learn about and implement a topic that we share a passion for. Thank you to Iowa State University for allowing us to pursue this project and providing us the support necessary to complete the project. Thank you also to Dr. Rursch for her mentorship throughout this process.

1.2 Problem and Project Statement

1.3 Operational Environment

Our messaging app is intended to operate on any iOS or Android platform. This means that it could be used in any part of the world or any time of day, so we need to keep it uniform across all devices. The fact that it could be used in any part of the world means that the security of our app must be as strong as possible so that no one (from a single hacker sitting in a coffee shop to a hostile foreign government) is able to crack the encryption and read the messages that are being sent. In relation to time of day, we need to ensure that the app is up and running whenever a user wishes to send or receive a message.

1.4 Requirements

1.5 Intended Users and Uses

Our intended users are those that desire an extremely secure yet straightforward way to communicate with each other across networks that are known to be hostile. The use case is somewhat narrowed by the caveat that the users must physically meet in person in order to generate the encryption keys that will be used for their conversation, however, there are a number of use cases in which this is an acceptable compromise to ensure strengthened security.

One potential use case involves users who are actively protesting a federal government that has the capacity to intercept internet traffic and potentially launch man-in-the-middle attacks against communications used to establish or share encryption keys. Protesters would be able to meet in person to generate and exchange encryption keys before using the platform to coordinate direct actions against the government.

Another use case would be journalists traveling to a region where their communications will be monitored for espionage purposes. They would be able to coordinate encryption keys before traveling and then utilize the platform to transmit their reporting back to their organization.

Finally, a potential group of users for our project are privacy and security advocates. Many individuals who study or work in the security fields use secure communication platforms out of

principle, and it's believable that a large number of them would be interested in utilizing our platform for secure communication.

1.6 Assumptions and Limitations

Assumptions

- We will have many simultaneous users interacting with each other through the application at any given time.
- This product will be used on both iOS and Android devices, and these two platforms will be interacting with each other. In other words, iOS users should be able to communicate with Android users and vice versa.
- Our users will be in close proximity with each other at least once so that they can exchange keys in person.
- Our application will be used around the world, and not all communications (outside of the initial key exchange) will be occurring at short distances

Limitations

- As this concept has not been implemented in any messaging applications that we know of, we will have to do most of our troubleshooting and testing on our own with little prior art to reference.
- We will be limited in the amount of concurrent users we can have and test with due to the small size of our team and the logistical issues with distributing an application that isn't on an application store.

1.7 Expected End Product and Deliverables

We have developed a plan for the deliverables that we are expecting to complete throughout this project.

Our first deliverable that leads to this is a mockup of the application interface for the frontend and a server design for the backend. For the frontend, we want to make sure we have an idea of what our application should look like. This includes menus, screens and the functionality that we should be implementing on the client side. For the backend, we want to have the needed functionality and storage planned out for a prototype of the application to be able to function. This will include the amount of messages we want to store, how they will be stored and how they will be delivered. This will ensure we stay on the right track towards a working prototype. Our goal for this deliverable is October 18th, 2019. This date will give us ample time to begin work on a prototype that matches the designs we had.

Since we are just beginning the project, our ultimate goal is to have a working prototype by the end of the semester. This prototype will be able to exchange keys successfully, but may not have the messaging aspect working completely. This means will want to have any additional methods of key exchange ironed out and functioning at this time. If we are able to get some messages that can be sent and received, it will be a bonus but we are mainly focused on proving that our concept will work. This will demonstrate that we have a strong foundation going into the second semester

and that we will have a working project at the end of our time in senior design. Our date for this deliverable is December 13th, 2019.

By the middle of the next semester, we plan to have a working version of the application that is able to exchange keys, send encrypted messages, and decrypt received messages. The goal with this deliverable is to be able to show that we have developed a working application that may still need some polishing but is functional. It should be able to encrypt messages, send them, receive them and decrypt them, as well as exchange the keys in the ways that we implemented in the prototype. This will give us time to make minor improvements and adjustments to the overall application to improve its performance for the user as well as fix any minor issues or bugs that may have arisen. Our date for this deadline is February 10th, 2020.

Our final deliverable will be a fully working application that has been optimized and is running as well as we are able to make it. This should have all of the features that we planned implemented and running smoothly, and we will be able to demonstrate that we can communicate with multiple users who all have different keys. There will not be any major bugs/glitches, and the application should be easy to use and efficient. Our date for this deliverable is March 1st, 2020.

2 Specifications and Analysis

2.1 Proposed Design

2.2 Design Analysis

2.3 Development Process

2.4 Design Plan

3 Statement of Work

3.1 Previous Work and Literature

To our knowledge, this is the first attempt to make key exchange outside of TCP/IP possible. At the time of writing, the only way to do so would be to write down the key, and manually type it in to the recipients machine. Even the current maximum-paranoia applications using public key encryption still transfer the keys over TCP/IP.

Our wrapper application, a secure messaging application, is not a new concept. The major difference between our application and existing systems, like Signal or WhatsApp, is that we allow users full control of their encryption keys, as well as non-standard means of transferring keys in a practical manner. These allow for a greater level of security in the transfer of messages. ("Messages" herein is used in the cryptographic context, referring to any piece of data, not necessarily a textual one.)

3.2 Technology Considerations

3.3 Task Decomposition

The major problem is the sharing of unique cryptographic keys in a zero-trust environment, followed by the use of these keys to secure communications across a network that is known to be hostile. This can be broken into these steps:

1. **Generate a secure, cryptographically random symmetric key, with each client drawing from cryptographically secure sources of randomness.** The key generation must be completed before any exchange can occur. We will want to use a cryptographically secure source of randomness to avoid the key generation being predictable or reverse engineered. We will need to make sure that the source of our randomness does not allow malicious users to predict values that are generated, and that they cannot deduce any previously generated keys should a key be compromised. There are libraries with functions that implement this functionality for random number generation, so we will make use of those rather than write our own methods that may be insecure.

2. **Establish a secure, non-TCP/IP communication channel with recipient device.** We will be passing information directly from one device to the other using QR codes and a camera on the device. The keys to be exchanged will be translated into a QR code which can only be scanned in person, as this will avoid the need to transmit the information over a network connection. This channel will be secure from attackers trying to sniff the traffic on a network, but it does bring about other security concerns. In order for this channel to be truly secure, the users will need to ensure that there is no way for an attacker to view the exchange occurring, since if they can view the QR code they could take it and use it in future attempts to gain the key.

3. **Verify the identity of the recipient device.** Verification of device identity should be a straightforward process. If two users trust each other enough to exchange keys in person, then it will be easy to verify that the recipient device is correct since the communicating devices will be in the same room as the parties performing the exchange.

4. **Perform key exchange.** As the symmetric key that will be used for the conversation is generated collaboratively by the client devices, the key exchange takes place throughout the

process of creating the key. The devices are placed with their screens facing each other, so that each device may see the other's screen using its front-facing camera. As the devices draw on their own cryptographically secure sources of randomness, they collaboratively generate a single symmetric key that will be used for their conversation.

5. **Verify key integrity.** Once the exchange has been completed, it is crucial to verify the key integrity. This step will ensure that usable keys have been exchanged and that no information was lost during the exchange. If a usable key has not been generated, the exchange will need to happen again before the two parties go their separate ways because otherwise it will not be possible to successfully encrypt and decrypt messages that need to be sent in the future. Each of these steps is absolutely necessary for the protocol to be viable. Each step depends to some degree on the previous, but each step can be treated as it's on functional unit in the program.

3.4 Possible Risks and Risk Management

3.5 Project Proposed Milestones and Evaluation Criteria

3.6 Project Tracking Procedures

We will utilise the Kanban boards that are built into GitHub to establish and track goals and progress. Andre has been elected to serve as a manager tasked with ensuring that we keep achieving goals at an acceptable pace. He will ensure that our metrics regarding percentages of tasks completed are reported back to Dr. Rursch and that corrective actions are taken if necessary to ensure the prompt realignment of our team with our original goals.

3.7 Expected Results and Validation

Our goal is to have a secure, user friendly method of exchanging keys in a zero-trust environment, and then having a viable means of communicating with these keys, all with no trust in the internet infrastructure. In order to confirm that we have met these goals, we will perform extensive security testing of our platform. More details of these tests can be found in section 5 of this document, which outline some of the tests necessary to ensure that our project goals have been achieved. These goals can be summarized as follows:

1. First and foremost, we will have functional cell phone applications for both the Android and iOS mobile operating systems. We will also have a very portable server application that can be deployed on a variety of operating systems. The client applications will be designed such that they require a form of authentication such as a passphrase before they can be unlocked, and all data stored on the client devices will be kept encrypted at rest and decrypted on the fly after the user has authenticated themselves.

2. Second, we will have two cell phones with our client application collaboratively generate and share a symmetric encryption key. This will occur with no transmissions over any network, including Wi-Fi, GSM, etc. The key will be generated with cryptographically secure sources of randomness, ensuring that not only can the key never be intercepted, but it can not be recreated by an adversary. This collaborative process will also be when the clients exchange their respective public key information for message signing and validation.

3. Third, one of the client devices will compose a message, encrypt it with the appropriate public key, attach the public key of the second client, sign it with their own private key, and then send it to the specified server.

4. Fourth, the server application will receive the incoming message, verify that it has been signed by any user who has previously uploaded their public key, verify that it is destined for another user who has uploaded their public key, and then attempt to route the message to its intended recipient. Depending on each user's predetermined settings, the message may be kept or automatically deleted from the server once it has been received by the intended recipient. If the message is kept on the server, it remains in the strongly encrypted state it arrived in.

5. Finally, once the second client has received the message, it will be decrypted once the second user inputs their passphrase to unlock the symmetric key used for decryption.

At every point in this process, we will be testing to ensure no adversary would be able to intercept or produce a plaintext copy of any message or the keys required to encrypt/decrypt messages being sent by legitimate users. To aid in testing, we will distribute our application to several volunteers to send and receive messages while other volunteers who are experienced network security professionals attempt to intercept communications, reproduce the keys of other users, and generally disrupt the service.

4 Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

4.2 Feasibility Assessment

4.3 Personnel Effort Requirements

Table 1: Task/Time Breakdown

Task	Time
Requirements	36 Hours
Specifications	24 Hours
Prototyping	80 hrs
Development	180 hrs
Stabilization	30 hrs
Final Release	30 hrs

4.4 Other Resource Requirements

This project will require development and testing for our clients and servers. The iOS and Android clients will be tested on an iOS and an Android device respectively, and we have the devices necessary to develop and test our project. We have independently sourced a MacBook to assist in the building and deploying of the iOS client code.

4.5 Financial Requirements

As this entire project will be developed and completed with free and/or open source software run on hardware that is already owned by our group, we do not anticipate any financial costs at this time. Group members will use their own computers to develop the project, and we have borrowed a MacBook to assist in building and deploying iOS code. The project will be deployed on server hardware and phones owned by various group members.

5 Testing and Implementation

5.1 Interface Specification

No special software testing interface will be necessary for the development and deployment of testing suites for our project. The Go programming language testing library will be used to assist in the generation and use of tests of our server infrastructure code. The Flutter testing library will be employed for functional testing of each module in our Android and iOS applications. As no hardware is being developed in this project, we do not need to develop any hardware testing specifications.

5.2 Hardware and Software

Testing for our Flutter application will require us to have Android and iOS devices to place the app on. One of the main features of Flutter development is the "Hot Reload" feature, which allows us to make changes to code and then instantly load the new code and changes to a running device connected to the computer. This will be extremely useful for testing as it will allow us to test the application on real devices, make necessary changes and then test those changes without wasting time compiling new code and moving it around to get it onto devices. This should be the extent of the hardware needed for Flutter testing as other tests will be carried out in software.

Flutter also has a testing library built in, so we will be making use of this. It has built in unit testing, widget testing and integration testing, with documentation provided for each. The unit tests will be used for testing smaller functions throughout the application, like display values and the logic in the functions. These will be helpful when making smaller changes or when trying to determine the source of bugs. Widget tests are used to test each of the screens in our application and their functionality. These tests will allow us to simulate users interacting with the various screens that we designed and allow us to ensure that button presses result in the correct actions, check that needed elements on the screen are present and test out any other functionality relating to the screens.

We will be using unit tests to test our Go server code. Specifically, we will be using Visual Studio Code and GoLand to develop the code needed for these tests. We will also use these two programs to compile and develop the server-side code.

5.3 Functional Testing

Unit-Based Testing

For testing our server-side code, we will create unit tests written in Golang, taking advantage of the Go programming language's built-in testing library. Unit tests will be created incrementally with the server-side code to ensure that our code doesn't introduce vulnerabilities or bugs that would compromise the security or functionality of our application. To test the IOS and Android clients, we will be using Flutter unit tests.

Integration Testing

Due to the relatively small size of this project, integration testing will come somewhat naturally as we continue to develop both the server and client software. Some automated tests will be used to ensure that client/server communication continues to work throughout development, i.e. ensure

that APIs are not modified unexpectedly. Beyond these few tests, further integration tests are not specifically planned at this time, although they may be added if a need becomes obvious.

System Testing

After our application passes the integration testing phase, we will then ensure that our clients and server communicate and behave as expected. This will include sending malformed requests to ensure the server handles errors properly, verifying that all communications are encrypted and decrypted when appropriate between the clients and the server, and general usability tests.

Acceptance Testing

After our application passes system testing, we are going to have people test our application to make sure that it works. We will attempt to recruit people of all technical skill sets to ensure that our application can be used by everyone. We will be creating documentation of how to create new servers and how to use our application as a whole, and we will also have our users verify that our documentation is articulate, and easy to understand. We will also involve a small group of external users as beta testers once our project is in a feature complete state.

5.4 Non-Functional Testing

Performance Testing

Our tests for performance will be performed by the users we distribute the application to. We will need to make sure that users can send messages without a noticeable delay and this test is best accomplished through heavy use by real users. Heavy use by our users will also allow us to test server performance. Since we plan to have users all around the world and have many users at once sending and receiving messages, we will want to perform some "stress tests" to ensure that servers can adequately handle the traffic of multiple users attempting to connect. Periodically we will get feedback from our user base and we will use this to determine where performance is lacking, then refocus our efforts towards fixing these performance issues.

Security Testing

In order to ensure the security of our platform, we will conduct periodic code reviews of our server and client code. We will also conduct at least one penetration test against our service, attempting to circumvent security controls as well as attempt a denial-of-service that leverages a weakness in our platform (distributed denial-of-service is considered outside the scope of our testing). We also intend to invite friends who work as professional penetration testers to launch attacks against our platform to see if they are able to intercept communications or disrupt the service in a preventable way.

Usability Testing

As soon as our project is in a semi-workable state, we will begin deploying development builds to our server and our personal cell phones. This will give us the opportunity to constantly use the application, finding bugs and noting usability issues in the process. Closer to the end of the development of the project, we will give access to the application to a group of friends and associates who have agreed to test our application and give us feedback on the interface, functionality, and other usability factors.

Compatibility Testing

As our project is not designed to communicate across platforms other than our own, there are very few points for compatibility issues to arise. We will perform testing with an Android emulator to ensure that our android application works across a variety of devices, and beyond that, strenuous compatibility testing will not be necessary. Due to the nature of iPhone development, our app will seamlessly work on any iOS phone platform. Our server code can easily be built and run for nearly any operating system on nearly any architecture, thanks to the versatility of the Go programming language, and specifically the power of the ‘go build‘ command, which allows for effortless cross-compilation.

5.5 Process

5.6 Results

6 Closing Material

6.1 Conclusion

6.2 References

6.3 Appendices