Satellite Project Final Report
EC544 Networking the Physical World
April 29, 2020
Cameron MacDonald Surman, Hairuo Sun, Yi-Wei Chen

Overview

Our project is to implement “microsatellites”, a lightweight extraterrestrial probe can obtain sensor data and send it back to the control center. The control center and a microsatellite are treated as 2 different entities that can communicate with each other and send information. This process includes 4 parts: Encryption/decryption, authentication, API and communication. In our implementation, we first authenticate both the control center and satellite using RSA key pairs and verify the secure IDs from both sides. After the authentication process is completed, we input a geographic location to the control center, and ask a microsatellite for real-time data on weather. After the satellite receives the request and decrypts the message to get the requested location, weather information was obtained from OpenWeather API via Ethernet. This weather info is then encrypted and sent back to the control center. However, due to limiting factors, we weren't able to implement the radio module.

Success of the project

Authentication:

The authentication method was successfully implemented in the command line tool using openssl and RSA key pairs. We used this method to exchange passwords and secure ID, aka securID, so that the source of the sender can be verified before any additional exchange of information can continue. Referring to Figure 1 flow chart. In this project, we simulate the process of 2 entities, which are the control center B1 and the satellite B2, exchanging information.

In this process, we implement the RSA key authentication method similar to the way we implement lab3, but we exchanged securID on top of the sender message. First, we generate 2 key pairs and 2 random hexadecimal 8 bytes passwords for B1 and B2. Then, both B1 and B2 encrypt their password with the counterpart's public key so that the counterpart can exchange passwords that are decryptable.

B1 and B2 are also each given an 8-digit token to be used as its securID. Each securID is encrypted using its own public key and password so that, once either party has received the encrypted password from the counterpart and decrypted it, the identity of this counterpart can be authenticated. However, in real life, there is an additional procedure, which is to look up coumterpart's securID in the database so that the source of the message can be officially authenticated to be the right and trustworthy source. After both parties have exchanged passwords and securID, the whole authentication process is completed.

Encryption:

For the encryption/ decryption part, we have done it by AES-128 standard in CBC mode. The reason we chose 128 rather than 256 is that it is easier to implement, and the processing time is also shorter. Basically, each step (add key (XOR) -> byte substitution -> shift rows -> mix column) is the same for both 128 and 256, the only difference is just how many loops going through (9 for 128 and 13 for 256). The reason for selecting CBC over ECB is obvious in the sense that there is a possibility of leaking information when using ECB and CBC is actually the later version of AES standard.

The success of this section is that we can actually make encryption/ decryption working in correspondence with what we stated in the proposal. We can take a string at any length (reasonably not maliciously created) to be encrypted and then the decrypted message is exactly identical to the input.

We used crypto's library to fulfill this process because there are lots of functions regarding encryption/ decryption are pre-defined in it and it is an open-source material which means it is more accessible for everyone. For implementation of this library over different operating systems, please refer to "How I make it working on visual studio 2019.pdf" and "Implementation on ubuntu.txt" provided on GitHub.

In module 2 and lab 3, we have seen some examples on cryptography and tried hands-on implementation. Although it did not take a big role in lab 3 (key exchanges under RSA structure played the main role), we still get a taste of AES and how magical it is between an encryption/ decryption process. After having a little taste of it, we then had to search for a detailed mechanism behind the scene because we decided to include this perspective into our project. This learning process (little introduction -> little hands-on practice -> motivation on project -> learn deeper outside the class -> better understanding) helps us to understand a certain topic better because there are many things to be covered in the lecture, it would be hard for instructor to fully dive into while also being hard for us to fully understand in the first place.

API:

Originally, our project focused on the collection of API data related to the COVID-19 outbreak, and would help to determine what states were affected. As the virus has progressed, this was no longer feasible, and we opted to use the OpenWeather API to demonstrate the function of our project. The documentation for OpenWeather can be found at [https://openweathermap.org/current]. In our project, we would prompt the user for a zip code, encrypt and transfer to another board where the api call would be made.

While we could not perform this on 2 different boards, we still show the process clearly in our final version. In order to facilitate API calls in C++ we utilized libcurl, which allows for requests to be made to a specified endpoint. This returns a JSON string, and jsoncpp was used to parse the API response. From the response, we could pick out the information we wanted to present, craft a new output string containing the data, and return it to the "first board". In this way, we presented the flow of data from "Board 1" to "Board 2" to OpenWeather and back to "Board 1".

Discussion of Failure

Figure 2 flow chart shows the original workflow of our project. Compared to our current implementation, we have failed in the following area.

Originally, we included communication in our proposal. However, it would be harder for the hardware to synchronize in the case of being unable to meet in person (distance/ settings). Therefore, we thought of a substituting way to simulate our process: do demo on two containers. The failure of encryption/ decryption under this setting is that the key is generated differently from time to time and they are not going to be identical in separate containers. This will result in being unable to decrypt successfully given that AES is symmetric which means the shared secret should be known for both sending and receiving parties before the process. We tried to solve this issue by making the key fixed for the script, but it did not work well because the strings going through the AES process are not in ASCII format which makes pre-defining and printing harder. Instead, we go through the process in just one container to simulate how it should look like if two containers are involved.

The other failure is in the authentication implementation. We initially want the receiver to authenticate the sender's identity, securID every time a message is sent; however, we realized that this is redundant; therefore, our authentication only needs to be run once in the beginning of a program. However, One of our members that was working on the authentication section was unable to successfully compile any C++ code on her computer due to some technical problem with the command line tool and library problems. Therefore, the team and instructor instructed

her to use command line tools on her Macbook, which can run openssl, to directly demonstrate the authentication process. Since this authentication process only runs once in the beginning, the separation of the authentication process from the other processes doesn't affect the concept learning and implementation.

Supporting Figures and Diagrams

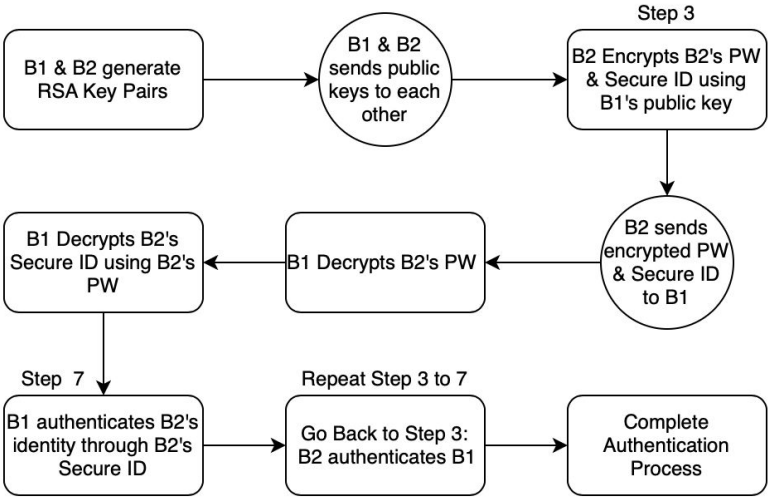

Figure 1. Flow Chart of How Source 1 (B1) checks the SecurID of Source 2 (B2)

Figure 2. Original Flow Chart of the Implementation Process

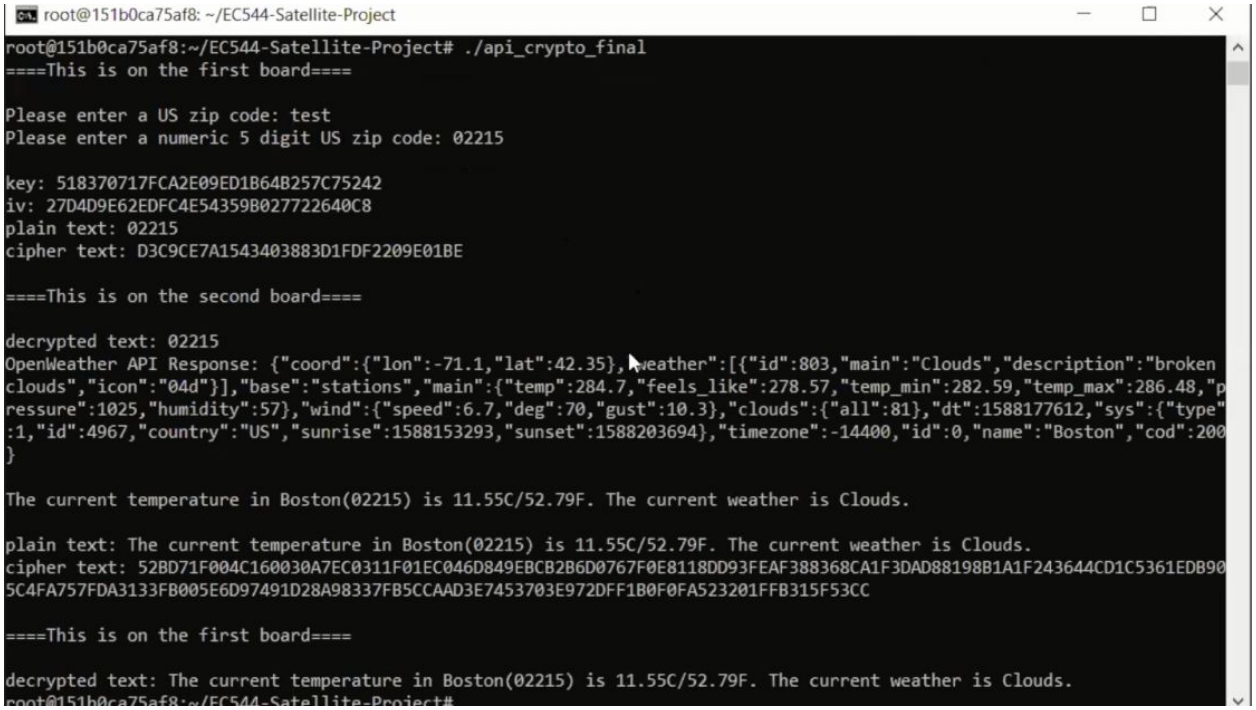

Figure 3. Sample output from combination of API and encryption/ decryption