Android Banking Application

Kevin HAYDEN*

Marist College

Professor Pablo Rivas

Security Algorithms and Protocols

April 3rd, 2018

Abstract. In this paper, I propose a method for using the Advanced Encryption Standard encryption algorithm for authenticating a user for banking software. The problem at hand is: How can I efficiently and effectively authenticate a user given only a user's email and password without storing their password in the database? How do I use the AES algorithm to accomplish my goal? Firebase offers many services for encryption and authentication but I wish to explore this on my own in order to learn the process. This paper will explain my thought process on the subject and how I tested my ideas using an Android banking application called Vault. I expect my scheme to ensure that data can be stored and retrieved securely from the Firebase database.

1 Introduction. The point of this project is to test the usage of AES encryption when using it to authenticate a user using data encryption. The project is a banking application which authenticates a user using their email and password. These two values are concatenated together and encrypted using AES-128. The environment I am developing in is Android Studio which is the premier IDE for developing, debugging, and testing Android applications. The primary programming language I am using is Kotlin which runs on the Java Virtual Machine.

1.1 Technology. My goal with this project is to work with and research technologies that I haven't worked with before. The primary programming language I used for this banking application is Kotlin. Kotlin is a statically typed object-oriented programming language that runs on the Java Virtual Machine. While Kotlin runs off of code from the Java class library and runs on the JVM, the syntax is much different in just about every way. Variables in Kotlin are statically typed and inferred from the expression. Kotlin is an extremely efficient language to program in because while it uses Java libraries, it is a lot less verbose. This makes programming and debugging much simpler than it's Java counterpart.

2 Methodology. The current scheme I am working with new users is the following; once the user completes all of the fields during registration, the email and password are concatenated together with a colon as a delimiter set between them. The scheme then encrypts that string using a randomly generated and stored 32-bit hex key and stores the ciphertext as the user's password in the database separately. The user's actual password is never stored in the database and can only be retrieved when the key is decrypted with stored 32-bit hex key. When authenticating, the user enters in the correct email and password, the software queries the database for the encrypted email address which serves as the primary key. Once retrieved, the email and password are encrypted with the stored key. If the generated ciphertext matches the stored key, the user is authenticated and the user's account information is loaded.

3 Experiments. I tested many different sets of input parameters that get generated when entered into the AES algorithm. The AES algorithm I developed takes two input parameters; the plaintext and 32-bit hex key. The algorithm parses the plaintext into a collection of length 16 strings and converts them to their 32-bit ASCII values. It then encrypts the ASCII string and concatenates them all together.

3.1 Test Data. The key is a 32-bit hex string which breaks down to 128-bits in binary. This a privately-stored, randomly generated key that is saved as a part of the user's attributes. The email is a string that is formatted to uppercase and is then stripped of its special characters. The password is a value that is not stored and must be remembered by the user. It is then formatted to all uppercase and then concatenated at the end of the formatted email with a ':' in between as the input for the AES encryption. The ciphertext is the output of the AES encryption algorithm. The plaintext is both the input for the AES encryption algorithm as well as the output of the AES decryption algorithm.

Key: 54776F204F6E65204E696E652054776F
Email: kevphayden@fakemail.com (KEVPHAYDENFAKEMAILCOM)
Password: 123456789
Plaintext: KEVPHAYDENFAKEMAILCOM:123456789
Ciphertext: 3CB483CAFE6801F749C5DB349F14123C701B9E45EE137B68C779724334B7C1D3

Key: 5468617473206D79204B756E67204675
Email: alexcampone@fakemail.com
Password: abcdefg
Plaintext: ALEXCAMPONEFAKEMAILCOM:ABCDEFG
Ciphertext: E7C34A4D952894D93F12C6C5CD752DC9240E3DAD0E9D40414D394B55C8545F9B

4 Discussion/Analysis. When running these test vectors on the encryption and decryption algorithms, it really shows the strength of not storing the password in the database. Without knowing the password you cannot generate the authentication key that allows entry. However without a mechanism in place that restricts access after a failed number of login attempts, this software is vulnerable to brute-force password attacks. I plan on implementing this feature in the coming weeks.

5 Conclusion. In this paper I have claimed that the scheme I have in place will be an effective tool for authentication because the Advanced Encryption Standard has yet to be broken (to our knowledge). The biggest battle will be developing the Android app that implements this scheme on the Firebase Database. Hopefully

5.1 Complete. So far what I have completed are the Kotlin class models, the AES algorithm in Kotlin, the Graphical User Interface for the login screen, and tests for the encryption/decryption.

5.2 To Be Completed. Fully functional login screen controller, the GUI and controller for account management screen, the GUI and controller for the register screen, the GUI and controller for the forgot password screen, brute-force protection, and test data for the other models in Firebase.