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. **Android Studio.** The environment I tested and developed my software in was Android Studio. This is official integrated development environment (IDE) for Google’s Android OS. This IDE was built on JetBrains’ IntelliJ IDEA software and tweaked specifically for Android app development. I thoroughly enjoyed using the software because of the IntelliSense usually implemented in all of Jetbrains products. This helps you learn more about the code you are using, keep track of the method parameters, and call properties or methods in just a couple keystrokes instead of typing the whole thing. I’ve noticed it is pretty good about sensing which method or property I am trying to access based off the code already written (or sometimes needs to be written). Android Studio also has a nice layout editor which allows drag-and-drop capabilities for GUI design.
   2. **Programming Language.** 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 libraries 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 it is a lot less verbose. This makes programming and debugging much simpler than it’s Java counterpart.
   3. **Database.** The method of storage I used is Firebase. Firebase is a web application that was developed by Google that offers a wide range of services such as Analytics, Authentication, Encryption, Cloud Messaging, Application Hosting, Storage, and a Real-time Database. Firebase’s real-time database is a NoSQL database that works primarily with key-value pairs. This database has many useful features such as the ability to save backups, set rules for read/write/validation, and measure usage. Android Studio has a nice setup tool for implementing and importing your Firebase database into the application by instantly updating your Gradle build files.

1. **Methodology.** ​The current scheme I am working with is based on secrecy. It works the following way; 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.
2. **Security.** The whole purpose of this application is to implement a secure way for the user to authenticate to the banking application. There are four rings of security an attacker must break through in order to break the cryptosystem.
   1. **Password Secrecy.** The strength of this authentication scheme is that the password is not stored within the NoSQL database or in any file on the system. To authenticate to the system, you must know the email and password combination. The authentication key is generated by encrypting the plaintext key using AES with the format “EMAIL:password”. The email is stripped of all special characters and converted to uppercase as an additional minor added measure. It is then concatenated with the case-sensitive password with a ‘:’ in middle. The ‘:’ acts a delimiter to separate the two Strings later on.
   2. **Advanced Encryption Standard (AES).** The second ring of security being used is AES-ECB at 128 bits. AES-ECB is secure against brute force attack from modern super computers as it would take 1.02 x 10^18 years to crack. However, AES-ECB is weak because it will always produce the same ciphertext from plaintext if encrypted with the same key. In the future I will change my encryption scheme to that of either CBC or GCM.
   3. **Hyper Text Transfer Protocol Secure (HTTP)** The third ring of security is HTTPS. HTTPS is the secure channel that Google uses (as well as many others) when transferring data back and forth from Client to Firebase. HTTPS is only responsible for keeping your data encrypted in transit. HTTPS uses the cryptographic protocols Transport Layer Security (TLS) and Secure Sockets Layer (SSL).
   4. **Firebase Security.** The fourth ring of security used the rule suite offered by Firebase’s real-time database. This suite lets you customize your database by setting read/write/validation rules to people and systems using the database.

1. **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.

* 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 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 (ALEXCAMPONEFAKEMAILCOM)

**Password:** ABCDEFG

**Plaintext:** ​ALEXCAMPONEFAKEMAILCOM:ABCDEFG

**Ciphertext:** ​E7C34A4D952894D93F12C6C5CD752DC9240E3DAD0E9D40414D394B55C8545F9B

1. **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.
2. **Future Points and Features.** There are many features I was not able to implement within the timeframe that I would have liked to. The biggest battle was learning to develop an application in Android Studio and learning it’s various libraries. Some features I will implement after the class is over are the following:
   1. **Two Factor Authentication (2FA).** Two factor authentications are a nifty tool when it comes to combatting cyber crime and internet fraud. It acts as a third token for logging into a system, but that token is usually transmitted to a secondary device. Therefore, the only way to break that would be to obtain access to the physical device.
   2. **Extra Security Questions.** During password recovery my application only has two set security questions. Most applications have at least 10 to choose from and of the ones selected only a couple are chosen at random to be filled out. These questions and answers ideally would be encrypted at rest in the database.
   3. **Account Lock.** I would like to add feature to combat brute-force by limiting the number of attempts to authenticate to the system. After a set number within a certain time-frame the account would be locked and they would have to reset the password (ideally through a process sent to the user by email).
   4. **Transactions Data Model.** Modern banking applications track every transaction and logs them in their database. I would like to add a feature in the main activity that would show the user the activity in each account. I would also like to implement a way to transfer money from one account to another.

1. **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). After numerous tests to authenticate to the system, it has proved to be crude yet capable method of authentication. There are currently more secure, existing APIs out there for such tasks including Firebase’s authentication service, but I want to create my own system to gain a better understanding of how to circumvent vulnerabilities that arise when developing cryptosystems.