Designing, Building, and Testing a Secure Android Utility Library

Nasrin Noor Ahmad, Jacob French, Christina Gomez-Sejas, David Haynes, Vu Nguyen

Sponsor: Leidos Inc.

Customer: David V. Szczesniak

Subject Matter Experts: Yaron Eidelman, Ammar A. Palla

Background

- ❏ There are many security vulnerabilities that occur due to improper design and implementation of security tooling.
- ❏ Our project seeks to eliminate these concerns by integrating with the standard library once, and exposing a single reusable interface for other applications to connect with.
- ❏ Users of our library want a single source of truth for Java encryption that is extensible, easy to implement and maintain across multiple platforms, and provides a common interface between implementations.

Software/Tools

We utilized multiple technologies in the development of our library, outlined below:

- ❏ GitLab / Git - The hosting platform for our code which allowed us to coordinate assignment of tasks as well as cooperate on coding.
- ❏ JUnit - The testing framework that we utilized to ensure that the functionality of the library could be verified and validated.
- ❏ Java Cryptography Architecture (JCA) - The foundational baseline for all cryptographic operations that occur in the Java language. We tapped into this extensively to leverage the cryptographic primitives that are exposed to high level applications.

Figure A. Gitlab

Figure B. JUnit

Objectives

The primary objective of this project is to provide the customer with a single source for Java encryption that may be integrated across multiple projects. This primary objective is accomplished over three primary phases, with testing and documentation along the way. Therefore, there are four project objectives, outlined below:

1. Development of reusability security library that implements security control functionality
2. Continuously test the code that is written and compose documentation that adequately details the code
3. Extension of the library to include additional functionality as suggested by the Customer and Subject Matter Experts
4. Implementing the library on an Android proof of concept application that demonstrates that the library functions as needed

Approach

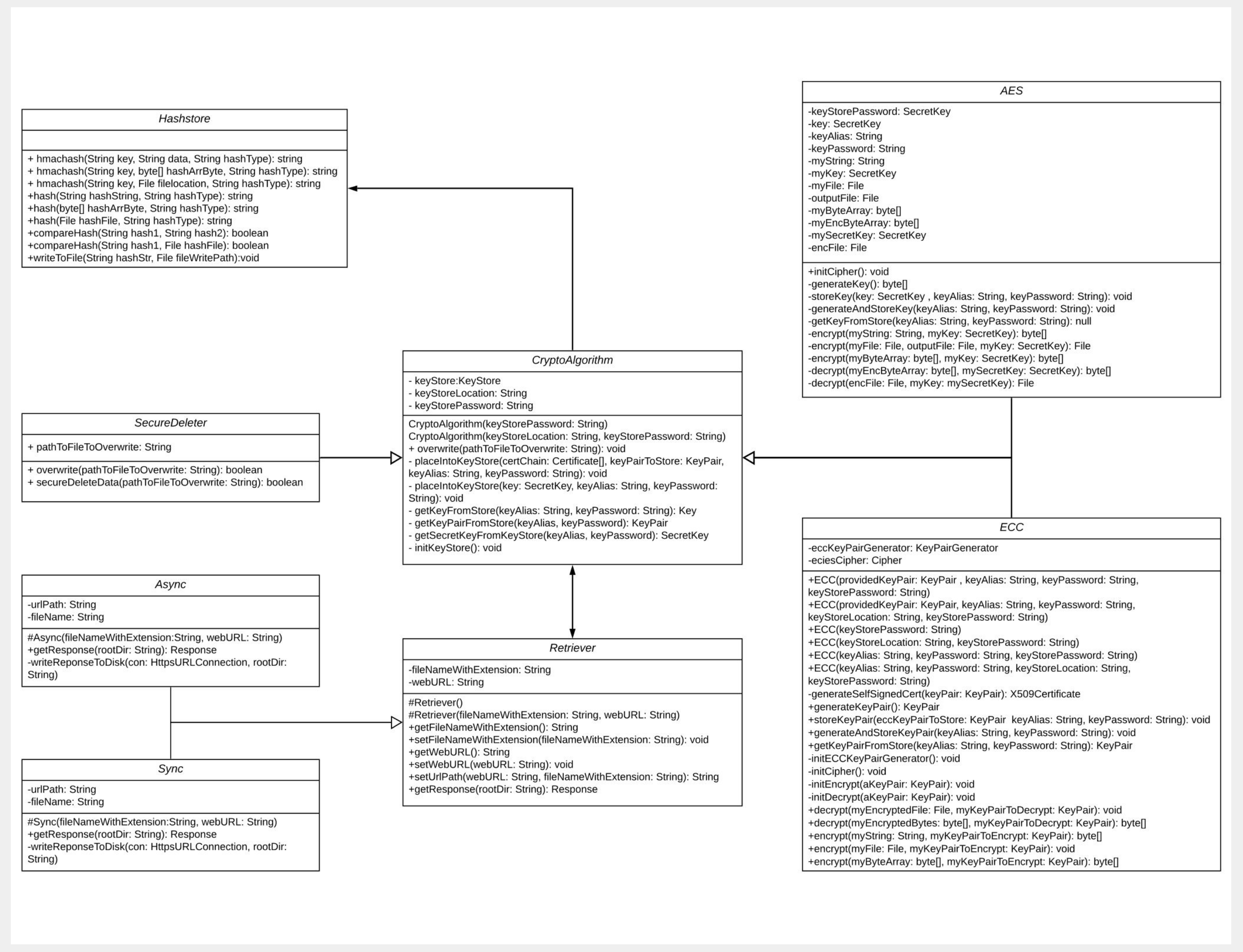

Hashstore
+ hmachash(String key, String data, String hashType): string
+ hmachash(String key, byte[] hashArrByte, String hashType): string
+ hmachash(String key, File filelocation, String hashType): string
+hash(String hashString, String hashType): string
+hash(byte[] hashArrByte, String hashType): string
+hash(File hashFile, String hashType): string
+compareHash(String hash1, String hash2): boolean
+compareHash(String hash1, File hashFile): boolean
+writeToFile(String hashStr, File fileWritePath):void
AES
-keyStorePassword: SecretKey
-key: SecretKey
-keyAlias: String
-keyPassword: String
-myString: String
-myKey: SecretKey
-myFile: File
-outputFile: File
-myByteArray: byte[]
-myEncByteArray: byte[]
-mySecretKey: SecretKey
-encFile: File
+initCipher(): void
-generateKey(): byte[]
-storeKey(key: SecretKey , keyAlias: String, keyPassword: String): void
-generateAndStoreKey(keyAlias: String, keyPassword: String): void
-getKeyFromStore(keyAlias: String, keyPassword: String): null
-encrypt(myString: String, myKey: SecretKey): byte[]
-encrypt(myFile: File, outputFile: File, myKey: SecretKey): File
-encrypt(myByteArray: byte[], myKey: SecretKey): byte[]
-decrypt(myEncByteArray: byte[], mySecretKey: SecretKey): byte[]
-decrypt(encFile: File, myKey: mySecretKey): File
CryptoAlgorithm
- keyStore:KeyStore
- keyStoreLocation: String
- keyStorePassword: String
CryptoAlgorithm(keyStorePassword: String)
CryptoAlgorithm(keyStoreLocation: String, keyStorePassword: String)
+ overwrite(pathToFileToOverwrite: String): void
- placeIntoKeyStore(certChain: Certificate[], keyPairToStore: KeyPair, keyAlias: String, keyPassword: String): void
- placeIntoKeyStore(key: SecretKey, keyAlias: String, keyPassword: String): void
- getKeyFromStore(keyAlias: String, keyPassword: String): Key
- getKeyPairFromStore(keyAlias, keyPassword): KeyPair
- getSecretKeyFromKeyStore(keyAlias, keyPassword): SecretKey
- initKeyStore(): void
SecureDeleter
+ pathToFileToOverwrite: String
+ overwrite(pathToFileToOverwrite: String): boolean
+ secureDeleteData(pathToFileToOverwrite: String): boolean
Async
-urlPath: String
-fileName: String
#Async(fileNameWithExtension:String, webURL: String)
+getResponse(rootDir: String): Response
-writeReponseToDisk(con: HttpsURLConnection, rootDir: String)
Retriever
-fileNameWithExtension: String
-webURL: String
#Retriever()
#Retriever(fileNameWithExtension: String, webURL: String)
+getFileNameWithExtension(): String
+setFileNameWithExtension(fileNameWithExtension: String): void
+getWebURL(): String
+setWebURL(webURL: String): void
+setUrlPath(webURL: String, fileNameWithExtension: String): String
+getResponse(rootDir: String): Response
Sync
-urlPath: String
-fileName: String
#Sync(fileNameWithExtension:String, webURL: String)
+getResponse(rootDir: String): Response
-writeReponseToDisk(con: HttpsURLConnection, rootDir: String)
ECC
-eccKeyPairGenerator: KeyPairGenerator
-eciesCipher: Cipher
+ECC(providedKeyPair: KeyPair , keyAlias: String, keyPassword: String, keyStorePassword: String)
+ECC(providedKeyPair: KeyPair, keyAlias: String, keyPassword: String, keyStoreLocation: String, keyStorePassword: String)
+ECC(keyStorePassword: String)
+ECC(keyStoreLocation: String, keyStorePassword: String)
+ECC(keyAlias: String, keyPassword: String, keyStorePassword: String)
+ECC(keyAlias: String, keyPassword: String, keyStoreLocation: String, keyStorePassword: String)
-generateSelfSignedCert(keyPair: KeyPair): X509Certificate
+generateKeyPair(): KeyPair
+storeKeyPair(eccKeyPairToStore: KeyPair keyAlias: String, keyPassword: String): void
+generateAndStoreKeyPair(keyAlias: String, keyPassword: String): void
+getKeyPairFromStore(keyAlias: String, keyPassword: String): KeyPair
-initECCKeyPairGenerator(): void
-initCipher(): void
-initEncrypt(aKeyPair: KeyPair): void
-initDecrypt(aKeyPair: KeyPair): void
+decrypt(myEncryptedFile: File, myKeyPairToDecrypt: KeyPair): void
+decrypt(myEncryptedBytes: byte[], myKeyPairToDecrypt: KeyPair): byte[]
+encrypt(myString: String, myKeyPairToEncrypt: KeyPair): byte[]
+encrypt(myFile: File, myKeyPairToEncrypt: KeyPair): void
+encrypt(myByteArray: byte[], myKeyPairToEncrypt: KeyPair): byte[]

Figure C. A Class Diagram depicting the class structure, methods, and dependencies within the library

- ❏ We broke the composition of the project down into individual sections based upon the cryptographic functions that they represent.
  - ❏ Crypto - There are two primary algorithm schemes for standard encryption and decryption operations: symmetric key and asymmetric key. We implemented both schemes into our library through the use of the AES and ECC algorithms.
  - ❏ Retrieval - This section included components that allow for secure network communication over the HTTPS protocol. In addition, users can choose to utilize synchronous or asynchronous methods in order to retrieve data from remote servers.
  - ❏ Stores - We implemented a class to handle hash operations on files, strings, and byte arrays and additionally provide a mechanism for users to be able to verify and validate provided hashes.

- ❏ The utility library by itself does not provide any usefulness until it is integrated into a practical application that utilizes it.
- ❏ We considered how users of our library would integrate it and modeled our solution and cryptographic abstractions based on that.
- ❏ One of the simplest baseline functions we used as a base case to analyze was the flow of data through the HashStore function.
  - ❏ There are multiple sub-algorithm choices that users could potentially select.
  - ❏ Data needs to come into the library from the user in order to be hashed properly.
  - ❏ Valid hashes must be ran through the HashStore properly.

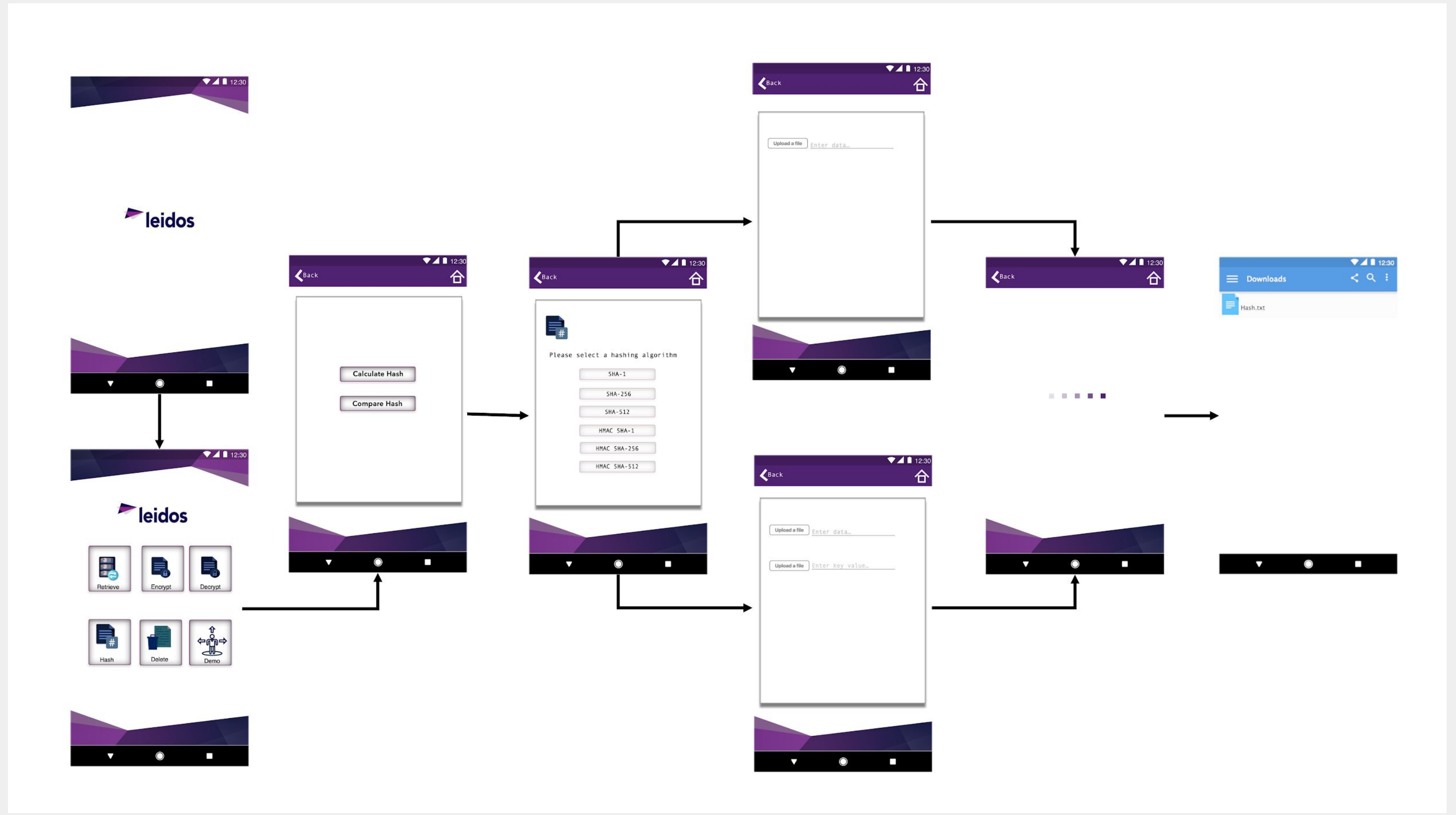
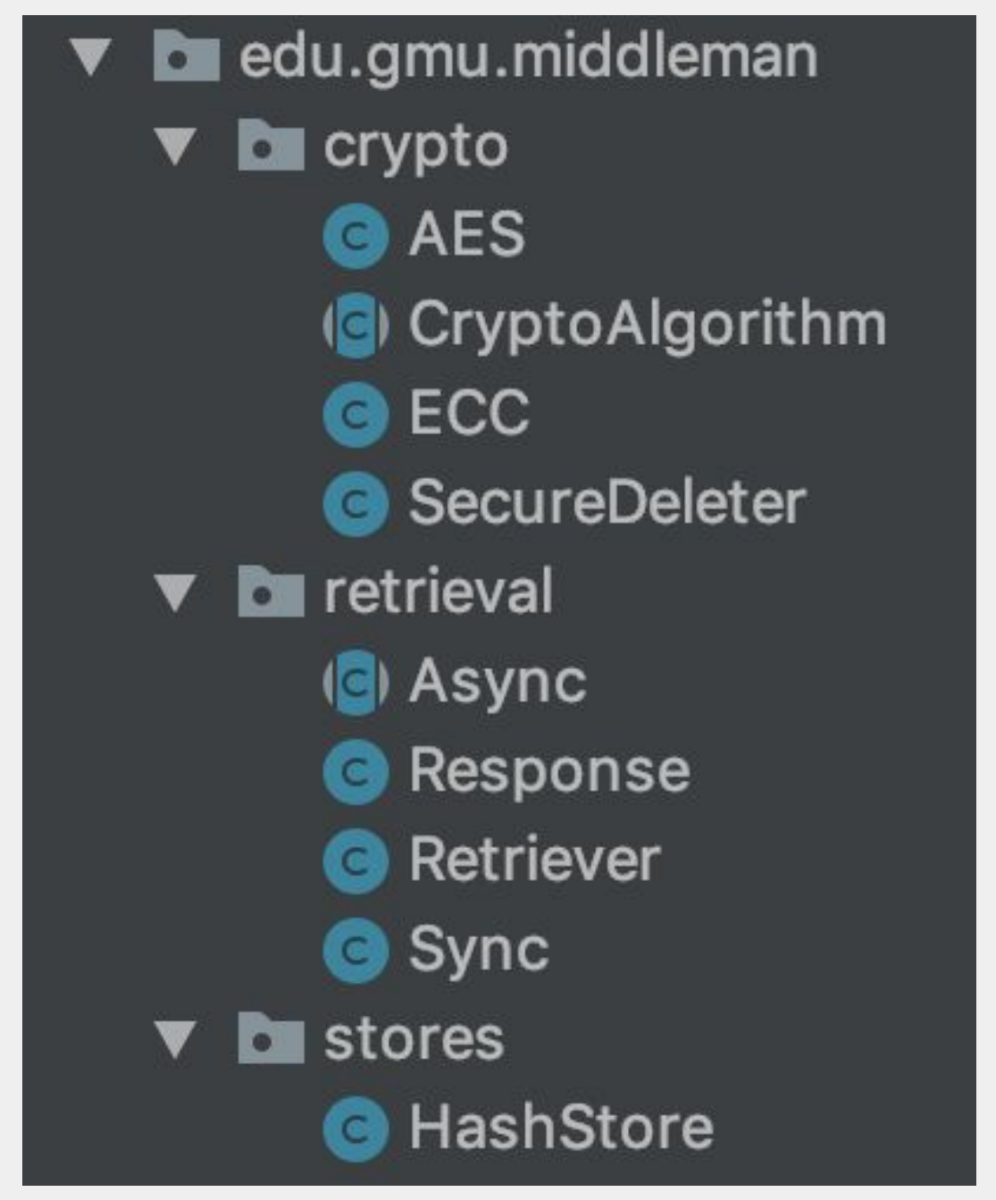


Figure D. An Implementation Concept Diagram displaying the flow of the library's intended use case.

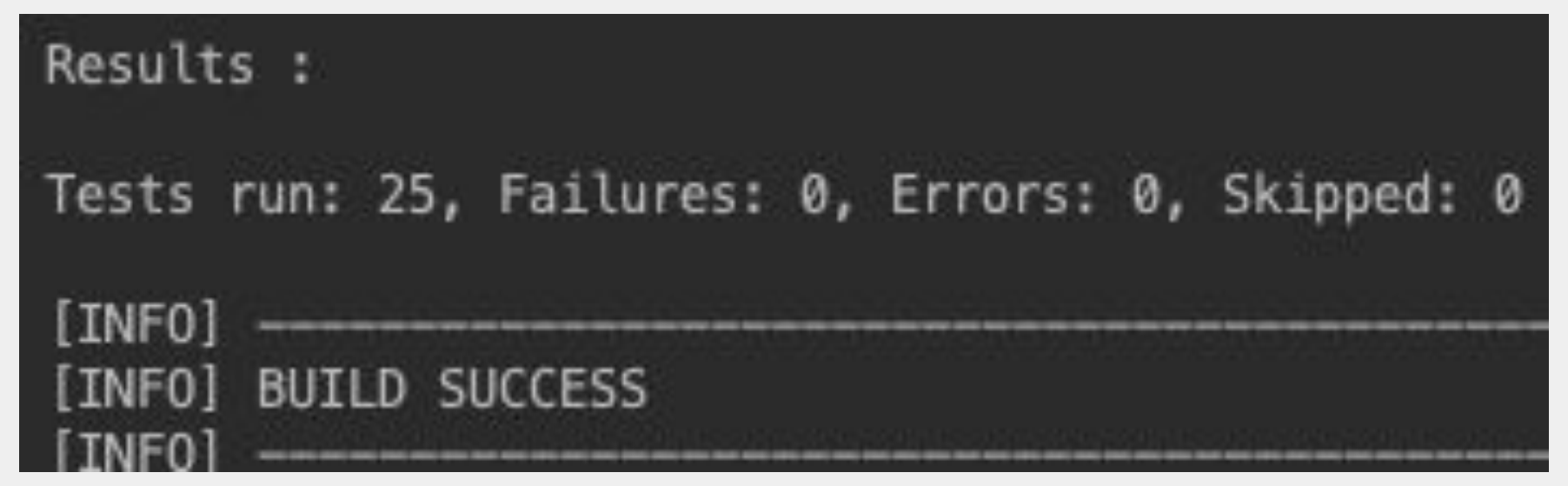
Results

- ❏ Core functions of the library that encompass encryption, hashing, secure deletion, and secure connection were developed.
- ❏ Abstraction of functionality allows for data flow between components to enable cooperation of disparate classes to achieve common goals such as key storage.
- ❏ Unit tests that assist in the verifying and validation of the code that was written were composed and ran on a constant basis to ensure consistent quality of library functionality.



```
▼ edu.gmu.middleman
  ▼ crypto
      AES
      CryptoAlgorithm
      ECC
      SecureDeleter
  ▼ retrieval
      Async
      Response
      Retriever
      Sync
  ▼ stores
      HashStore
```

Figure E. The library's directory

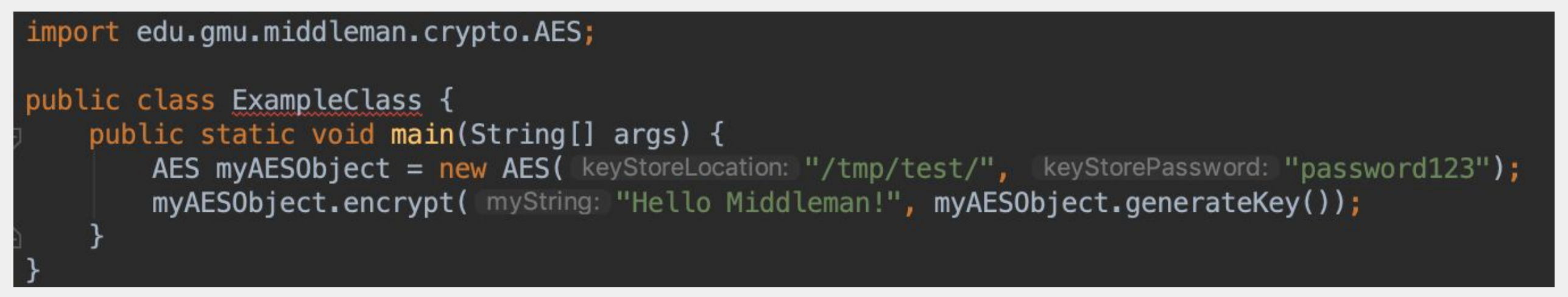


```
Results :

Tests run: 25, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
```

Figure F. JUnit test results for the entire library

Conclusion

- ❏ The library that we developed contains a variety of commonly used functions/tools that have been designed to conform to the latest security standards.
- ❏ It is reusable, extensible and can be integrated into any Java-based project may be enhanced to provide tools by which the security posture of the target system may be enhanced.



```java
import edu.gmu.middleman.crypto.AES;

public class ExampleClass {
    public static void main(String[] args) {
        AES myAESObject = new AES( keyStoreLocation: "/tmp/test/",  keyStorePassword: "password123");
        myAESObject.encrypt( myString: "Hello Middleman!", myAESObject.generateKey());
    }
}
```

Figure G. An example of a class utilizing the library

Acknowledgement

We would like to gratefully thank our sponsor from Leidos Inc., David Szczesniak, and Subject Matter Experts, Yaron Eidelman and Ammar Palla, for the opportunity and assistance provided for this project. We would also like to thank Gino Manzo, our George Mason University Senior Contact, for guiding us through our class and project.

May 2,2019