Android Encryption App

By

MacKenzie Healy

CS 4395 - Senior Project Report

In partial fulfillment of a
Bachelor's of Science in Computer Science

Department of Computer Science and Engineering Technology
University of Houston – Downtown

Spring 2022

Faculty Advisor:
Dr. Emre Yilmaz _____

Faculty Advisor:
Dr. Ken Oberhoff _____

Abstract

Cryptography provides confidentiality by hiding information, and cryptanalysis breaks and deciphers cryptography. Symmetric cryptography uses the same key for encryption and decryption and has a history as far back as the Greeks with scytales to Julius Caesar and substitution ciphers. In block encryption, plain text is encrypted, producing ciphertext blocks at a time. DES became the first national standard for block encryption, but AES replaced it due to insufficient key size and inefficient software implementations with bit operations. AES' block size is 128 bits and is efficient to implement in software or hardware. Although AES is more secure (with key sizes 128, 192, and 256 bits), any block encryption in Electronic Code Book ECB mode is deterministic as the same blocks encrypt to the same cipher. This project provides a tool for substitution cipher encryption and the means to break substitution ciphers without a key using hill-climbing and quad-gram statistics for fitness measuring. The application also allows a user to select a Bitmap for encryption using ECB mode compared with probabilistic encryption using Cipher Block Chaining CBC. After header correction, the encrypted images display side by side, showing information leakage only in the ECB file. Next, e-mail file attachments can be encrypted, decrypted, and sent with an external e-mail application. Lastly, by encrypting files with both DES and AES ciphers, AES outperforms DES.

Keywords: CRYPTOGRAPHY; SUBSTITUTION CIPHER; HILL-CLIMBING; QUAD-GRAM FITNESS; AES; DES; MODES OF OPERATION; ELECTRONIC CODE BOOK; CIPHER BLOCK CHAINING

Acknowledgment

By the Fall 2021 semester, I knew I wanted to continue digging deeper into cryptography and use mobile application development as the vessel. Fortuitously, my mentor of choice, Dr. Emre Yilmaz, was teaching Cryptography in the Spring 2022 semester at the University of Houston-Downtown, and he was available to advise me on my senior project.

During Yilmaz's course, I learned the foundations and history of cryptography, cryptanalysis, substitution ciphers, symmetric encryption, and block ciphers. Under his guidance, I was able to implement substitution ciphers and much of the AES and DES encryption algorithms using C++. However, I utilized the built-in libraries within Java for block encryption using AES and DES for performance reasons.

Initially, I planned to provide user-assisted breaking of substitution ciphers by providing n-gram and frequency analysis. However, Dr. Yilmaz challenged me to take it a step further and perform automatic cracking. He led me to online tools that performed this capability and directed me to begin researching algorithms and solutions. As a result, I elevated my senior project and created a solution I would not have thought possible before.

Along with being accessible and engaging, Dr. Yilmaz again encouraged me to take the project further by investigating the performance advantage of AES over DES through a variety of test cases and measuring operation duration.

Chapter 1. Academic Research and Project Design

HISTORY

Cryptography continues to evolve as the need for privacy and security expands into the modern age. However, before exploring and experimenting with cryptography tools today, it's essential to step back and review its past. The art of cryptography is safeguarding information, such as transforming text using a secret code (cipher key) and cipher algorithm, and the art of cryptanalysis is deciphering and breaking cryptography. (Al-Sabawwi 2020). The message being encrypted is usually called the plaintext, and the encryption process results in cipher text. In the modern age, as the world is connected through the internet, vulnerable data roars across our networks in need of protection.

In an increasingly connected world, transformed by innovations in computing machinery and communication, the need for cryptographic security is not new. Thousands of years ago, ancient Greeks were believed to have encrypted messages using a transposition cipher tool called a scytale. The user wrote their note in rows by wrapping the cylindrical scytale with a thin strip of parchment. The resulting parchment contained all letters of the plain text messages, but the scytale transposed the letters into different positions. The recipient would need a scytale rod with the same cylinder diameter to decode the cipher. With unmodified letters, an attacker need only try rods of different diameters to attempt the discovery of the original plaintext. (Kuusisto 2015).

In Ancient and Modern Cryptography, Finn Kuusisto explains the similarity between the ancient scytale and symmetric keys. Just as you need the same cylinder diameter to decode the cipher, symmetric algorithms utilize the same secret key for encryption and decryption. (2015). In Understanding Cryptography, symmetric cryptography is akin to having a key to a safe. If Alice and Bob have a key, they can lock items inside, unlock the safe, and remove items. (Paar 2014). Even if Alice and Bob keep the key safe, if a key is weak, attackers can try every possible key shape until they find one that fits the lock. Today, if Alice needs to communicate with Bob across an unsecured channel, she can send that data encrypted with a symmetric key they both shared before transmission. (Nie 2009). As that data traverses from Alice to Bob, an attacker eavesdropping would be unable to steal that information unless they've also obtained the secret key, the key is weak, or they've found another attack vector on the algorithm or its implementation.

Stepping back in time, around 100 B.C., Julius Caesar utilized one of the first symmetric algorithms, known as a shift cipher, to keep messages secret as they traveled to his military generals. This simple substitution cipher replaces each letter with a different one, a fixed number of places away in the alphabet. The number of shifts the alphabet rotates is called the cipher key, and Julius Caesar always used a key of three. With this specific Caesar key (3), the letter 'A' becomes 'D,' 'B' becomes 'E,' and so on, to 'Z,' which becomes 'C.' (Singh 2000).

After classical cryptography matured, another simple symmetric encryption emerged. Each letter is substituted with another in this substitution cipher, rather than simply shifting the alphabet. In this case, the symmetric key is the mapping of each alphabet letter. For example, 'A' could be mapped to 'D' and 'B' could be mapped to 'Z'. (Paar 2014). In the modern-day, classical ciphers are a perfect starting place for learning about cryptography, but as we will see later, they are no longer safe enough to use in practice.

Information security has advanced significantly with the industrial revolution leading to the information age. Rather than information residing on paper alone, now data is stored in bits of 1 or 0 and bytes (8 bits). Unlike classical cryptography, stream ciphers encrypt data one bit at a time, and block encryption encrypts data in blocks. Instead of encrypting each letter separately, a block cipher could encrypt one or multiple letters depending on the block size. However, text isn't the only content cryptography protects. Now, we can encrypt images, videos, databases, and anything else digital.

Governments used cryptography and cryptanalysis for national security, but in the early 1970s, the US National Bureau of Standards recognized the growing commercial need for data security. Now known as the National Institute of Standards and Technology, in 1972, the NIST sought solutions from the public to find a secure, standard block cipher algorithm. IBM developed the winning algorithm in 1974, and by 1977, Data Encryption Standard (DES) became the national standard with an input size of 64 bits and a 56 bit key. DES begins with an initial permutation, followed by 16 rounds of encryption and a final permutation. (Yassein 2017).

The right 32 bits feed an 'f' function in each round. The 'f' function result is XORed with the left 32 bits. That result becomes the next right 32 bits, and the original right 32 bits swaps to the left. Therefore, the algorithm encrypts only the left side in each round. As a Feistel Network, decryption is the same, except the key schedule is reversed. Inside the f-function, the right's 32 bits are expanded to 48 bits, XORed with the round key, divided into eight groups of 6 bits, fed into an S-box substitution resulting in 4 bits each, and finally permuted. The DES' bit operations make implementation inefficient in software. (Paar 2014).

By 1998, the Electronic Frontier Foundation developed Deep Crack, a machine capable of breaking DES in 56 hours and proving DES is insecure against attackers. The National Institute of Standards and Technology advised using triple DES in older systems. Triple DES encrypts three times with different keys and effectively doubles the key strength of Single DES. (Paar 2014).

Near the end of the 1990s, and in the face of DES' defects, the National Institute of Standards and Technology knew the country needed a new block cipher algorithm. Once again, the NIST sought solutions from the public to become the Advanced Encryption Standard (AES). Following the requirements given by NIST, AES encrypts blocks of 128 bits and supports key lengths of 128, 192, and 256 bits. Among the five finalists, NIST chose Rijndael by Joan Daemen and Vincent Rijmen in 2001.

AES increases security, and hardware and software implementations are efficient. Unlike DES, which contains bit-level operations, AES operates on bytes. (Paar 2014).

During the Byte-Substitution layer (the only non-linear part of AES), the algorithm divides the input into 16 bytes, and each byte feeds into its own S-box. While DES's S-boxes were all different, AES S-boxes are identical. In cryptography, diffusion spreads out the influence of one plaintext symbol over many ciphertexts. AES obtains diffusion when the ShiftRows layer permutes the data on a byte level and the MixColumn mixes blocks of four bytes using Matrix multiplication. At the end of each round, AES adds the round key. The subkeys for each round are created recursively from the original key. Unlike the Feistel design of DES, AES decryption must invert the layers of each round. (Yassein 2017). The AES algorithm is detailed further in the Appendix.

Selecting a secure block cipher encryption isn't enough to guarantee confidentiality. To encrypt data, especially when that data is larger than the algorithm's input block size, developers must choose a mode of operation. Modes of operation detail how the block cipher will be used to encrypt blocks or create a secure stream cipher. Two options are the Electronic Code Book mode (ECB) and Cipher Block Chaining mode (CBC).

In Electronic Code Book mode, the cipher encrypts each data block separately. The input to the encryption algorithm is the plain text, and the block cipher output is ciphertext. ECB allows for parallelization, but this mode is deterministic. In deterministic encryption, identical blocks result in the same ciphertext. (Paar 2014). On the other hand, in Cipher Block Chaining mode, the first plaintext block is added modulo two (XOR) with an Initialization Vector (IV) nonce. This result becomes the input to the block cipher encryption algorithm. Subsequent blocks are XOR-ed with the previous ciphertext output. Therefore, CBC is securely probabilistic but incapable of parallelization. (Alomari 2009).

CRYPTANALYSIS

The security of Caesar's cipher depended on the secrecy of both the cipher key (3) and the algorithm (shifting letters along alphabet by key). Suppose an adversary learned of the algorithm of shifting letters along the alphabet a set number of times. In that case, they could shift the intercepted message one character at a time until it became readable. There are twenty-five potential Caesar keys in the English language with twenty-six letters. Therefore, in the worst-case scenario, an attacker need only attempt twenty-five different keys before decrypting the message. (Menon 2013).

More than ever before, mobile phones are prevalent in our society. In fact, people are often closer to their cell phone than any laptops or desktops. From real-time GPS navigation, social media, phone calls, and especially private messages, the widespread use of mobile phones means a significant need for security. In previous academic work in 2013, Vipin Menon and Manuel Rivero utilized the classical Caesar cipher to enable a user to encrypt and decrypt secret messages in a modern way. Menon and Rivero created an iPhone application to encrypt messages using Caesar ciphers. (2013). Additionally, their iPhone application offered a tool to decrypt intercepted messages through two methods. First, a brute force method requires user interaction with each attempted key to determine whether a message was readable. Second, the application checked each possible Caesar key and checked for words listed in an included dictionary file. (Menon 2013).

A cryptographic system's key strength relates to the system's vulnerability to brute force. As seen with the Caesar cipher, the keyspace is 25. One wouldn't even require a computer to break it. When NIST introduced the Data Encryption Standard, the key size of 56 bits was considered secure. By 1998, the Electronic Frontier Foundations' Deep Crack machine proved the DES key was too weak.

Unfortunately, brute-forcing a cipher by trying every key isn't the only vulnerability in substitution ciphers. Interestingly, in a substitution cipher where each character is matched with a different one, the keyspace is 26 factorial. Imagine brute-forcing by hand all 26 factorial keys. Would you consider it safe? (Paar 2014).

In a single, simple substitution cipher, a significant weakness emerges. The twenty-six letters that make up the English alphabet appear in words at different frequencies. For example, the letter 'E' is one of the most used letters in the alphabet, with a frequency of 12.02%. Next, 'T' has a frequency of 9.10%, 'A' has 8.12%, and 'O' has a frequency of 7.68%. (Cornell 2015). With a sufficiently long ciphertext, an attacker could count the number of times each letter appears and calculate the frequency. By comparing the calculated frequency with known English letter frequencies, one could begin to make intelligent guesses. Although Caesar and other simple substitution algorithms are the most vulnerable to this form of cryptanalysis, other substitution ciphers can also be affected. (Singh 2000).

Furthermore, letters are not the only thing with predictable frequencies. An n-gram is a sequence of n letters occurring contiguously in a sample text. For example, a list of bigrams for a text sample would include all the combinations of two letters that appear together. In the sentence, "HI WORLD," the tri-grams are: {HIW, IWO, WOR, ORL, RLD}. These various n-gram combinations occur in the English language at different frequencies. Researching quadgrams, I found "TION" to occur more frequently than any other. An attacker could make educated guesses at the substitution key by analyzing ciphertext for n-gram frequencies and comparing it against known frequencies in English. (Azimi 2012).

Returning to the discussion on symmetric block encryption systems, even the NSA has adopted the Advanced Encryption Standard as an algorithm secure enough to protect even top classified information. However, choosing the wrong mode of operation could undo any confidentiality benefits. Most notably, in the Electronic Code Book mode, ECB, with each block encrypted independently, patterns can emerge for an attacker without breaking the encryption. For example, when encrypting a bitmap image, any blocks containing the same pixels will always encrypt to the same cipher text. Therefore, an attacker could repair the encrypted file's header, open the image, and still be able to learn details about what image was encrypted. (Paar 2014).

PROJECT DESCRIPTION

Building upon the foundation of cryptography research and previous academic research work by Vipin Menon and Manuel Rivero in 2013, my senior project involves creating a mobile application with a set of cryptographic tools from substitution to block ciphers. While there are plenty of computer solutions for cryptographic needs, this project explores solutions available in a mobile environment. First, the user will be able to

encrypt plain text and decrypt ciphertext using a randomly generated substitution key or a key chosen by the user. The next plan was to provide the user with cryptanalysis tools such as frequency analysis of letters and n-grams to allow user-assisted cracking of substitution ciphertext without a key. After working with Dr. Emre Yilmaz, he encouraged me to take it a step further. Rather than user-assisted cracking, the mobile application will automatically decipher imported ciphertext. The application also examines insecure modes of operation for block ciphers. The user can select a bitmap to encrypt using AES in Electronic Code Book and Cipher Block Chaining modes. The app then displays the original image and both encrypted images side-by-side for comparison. Next, the application will allow users to encrypt files using AES in Cipher Block Chaining mode and e-mail those files along with a unique Initial Vector nonce. Plus, the application will allow a user to decrypt these files received through e-mail. Lastly, the application will encrypt an input file chosen by the user using AES and DES, displaying the duration each took to complete in milliseconds.

ALGORITHMS

Substitution Cipher

Key Generation:

1. Generate a random key or confirm user's custom key contains every alphabet letter precisely one time.
2. Create hashmaps from key to map between plain and cipher characters.

Encryption:

1. Create an empty string for ciphertext.
2. For each character of plain text String:
    a. If the character is in the hashmap: get the cipher character and append it to ciphertext.
    b. Else append plain text character to ciphertext unchanged.

Decryption:

1. Create an empty string for plain text.
2. For each character of ciphertext String:
    a. If the character is in the hashmap: get the plain character and append it to plain text.
    b. Else append ciphertext character to plain text unchanged.

Substitution Breaker

With 2^88 possible keys, brute-forcing a substitution cipher is infeasible, especially on a mobile device. One alternative method of breaking a substitution cipher is analyzing frequency analysis and n-grams for patterns in the ciphertext, comparing the results against English frequencies. However, the research provided a better method to break the cipher automatically without human intervention using hill-climbing. (Lyons 2013).

In hill-climbing, you begin with an arbitrary solution, make a single change, and test if the solution improved. This process continues until no better solution is achieved in that area, at which point you have reached a local maximum. Next, you can start from

another arbitrary solution and begin the process again. This time, your new local maximum could be the same or better than the last. The goal is to find the global maximum, which would be the key to breaking a substitution cipher.

Beginning with a random substitution key, we make an incremental change during each iteration and re-test the decrypted text's fitness using that key. In Practical Cryptography, James Lyons proposes using quadgram statistics as a fitness measure to test how similar the translated text is to English. "War and Peace" by Leo Tolstoy contains 2.5 million quadgrams, and the probability of each quadgram is the count of that quadgram divided by the total number of quadgrams. Next, the log probability is the logarithm of this number, and for any quadgram that appears zero times, the algorithm assigns a floor value. We calculate all the contained quadgrams' probabilities and multiply them together to evaluate plain text. (Lyons 2013).

For the text “HI WORLD”, the quadgrams are {HIWO, IWOR, WORL, ORLD}. Therefore, the total probability is p(HI WORLD) = p(HIWO) * p(IWOR) * p(WORL) * p(ORLD). And p(HIWO), for example, is count(HIWO) / N, where N is the total count of all quadgram counts in the training set. To prevent numerical underflow with small probability double values, the logarithm is taken of each probability. Recall, log(a*b) = log(a) + log(b). Therefore, log(p(HI WORLD)) = log(p(HIWO)) + log(p(IWOR)) + log(p(WORL)) + log(p(ORLD)).

The hill-climbing algorithm is:

1. Create variables to hold the best key and the maximum fitness.
2. Loop for 10,000 rounds unless the same local maximum is found three times:
    a. Generate a random key.
    b. Calculate fitness of decrypted plain text.
    c. Hill-Climb until no further improvements are found:
        i. Swap two characters in the key and re-calculate fitness.
            1. If the fitness improves, save the new key as the best key.
            2. Else revert to the previous key.
    d. If the best fitness found during hill-climbing is better than the maximum fitness: replace maximum fitness and best key.
    e. Else if the fitness matches the maximum fitness three times in a row, break.
3. Decrypt ciphertext using the best key.

Bitmap Encryption using ECB and CBC Mode

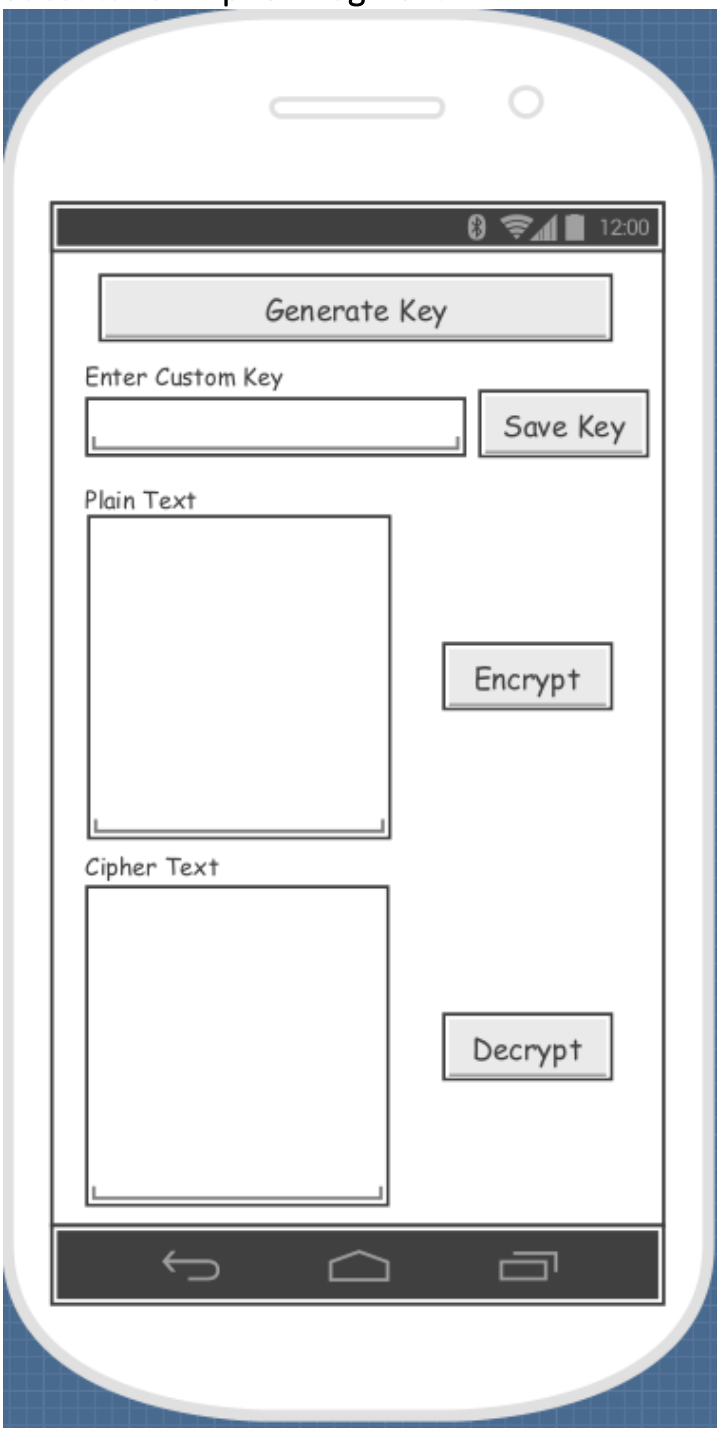
After opening the bitmap file chosen by the user:

1. Store 54-byte header from the original bitmap.
2. Generate a 128-bit AES key.
3. Generate IV.
4. Encrypt the original bitmap in Cipher Block Chaining mode using key and IV.
5. Encrypt the original bitmap in Electronic Code Book mode using the key.
6. Replace the 54-byte header in both encrypted files to make the bitmaps valid for viewing.

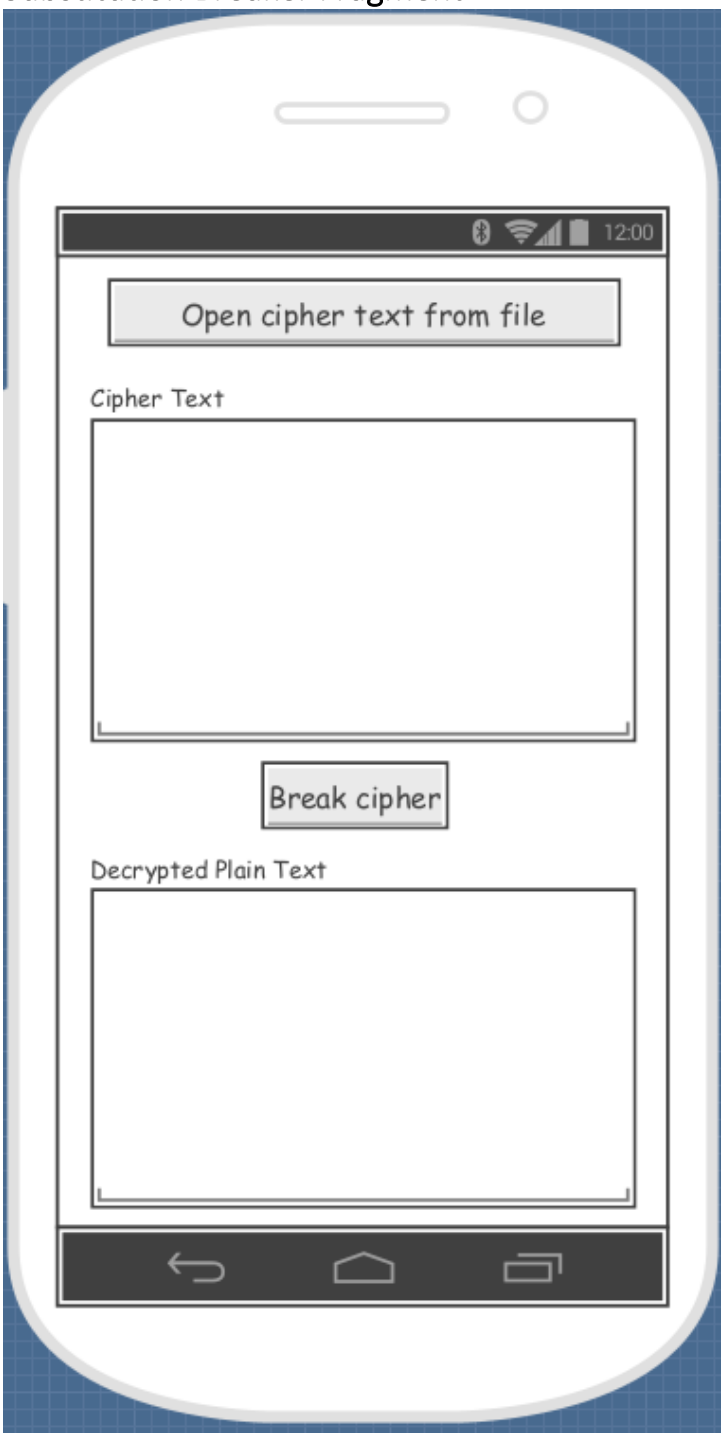
7. Display the original bitmap, ECB encrypted bitmap, and CBC encrypted bitmap for comparison.

USER INTERFACE WIREFRAMES

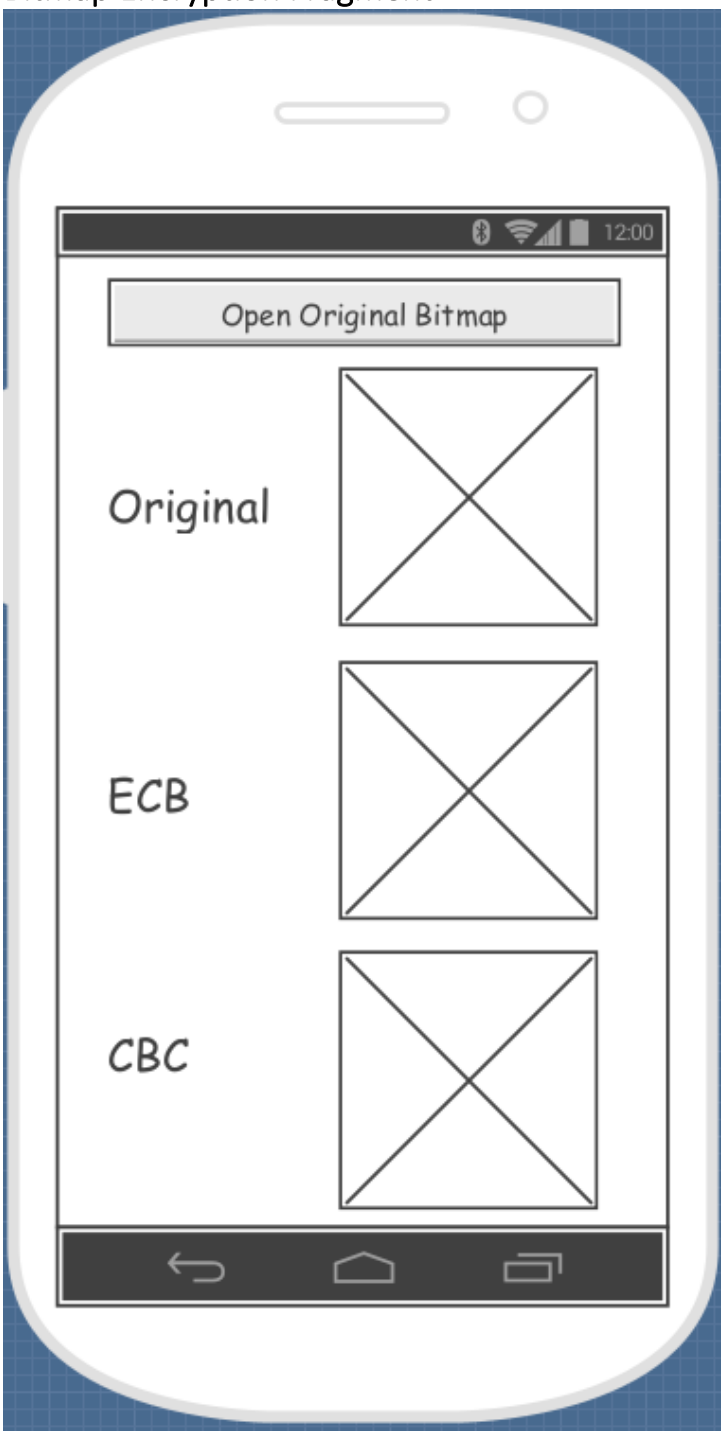
Substitution Cipher Fragment



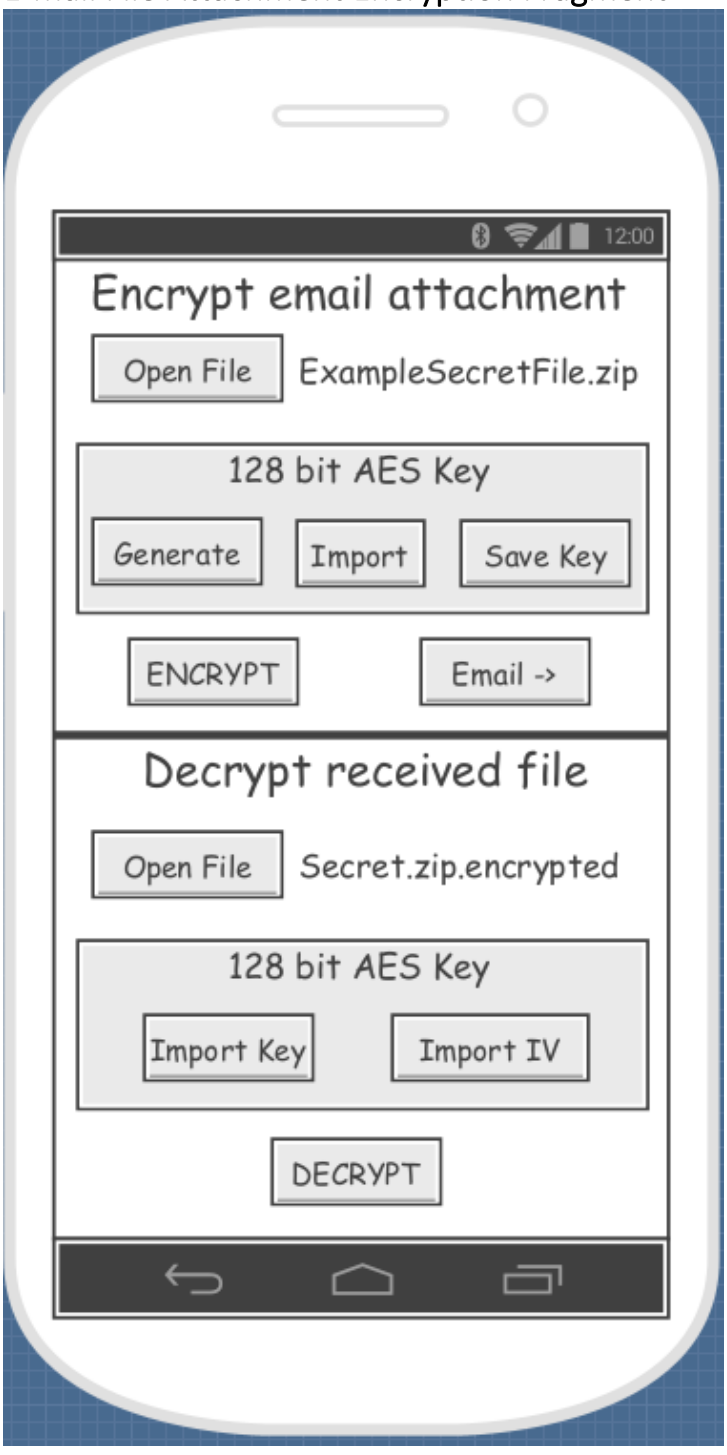
Substitution Breaker Fragment



Bitmap Encryption Fragment



E-mail File Attachment Encryption Fragment



Chapter 2. Implementation

PLATFORM AND DEVELOPMENT

Designing a mobile application leads to the first significant decision: choosing between iOS and Android platforms. Usually, iOS Development is in the Swift or Objective-C languages, and often the Integrated Development Environment (IDE) is Xcode. Unfortunately, there's no practical way to develop iOS applications on a Windows machine. (Apple). One solution is logging into a cloud Mac machine through RDP, but I found it burdensome and inefficient to test this option. Alternatively, Android development is possible across multiple operating systems and is usually in Java or Kotlin within the Android Studio IDE by Intellij. (Griffiths 2021). Unlike iOS, Android and the Android Studio IDE are built from open-source software.

Coupled with these reasons and my experience with Java, I chose Android as my platform of choice. Plus, Android Studio provides Gradle build support, Lint tools, an XML layout editor with drag-and-drop material UI components, an Android emulator, and version control features. I created all UML diagrams and wireframes in Visual Paradigm.

To facilitate implementation, I utilized numerous libraries included with Java, including Security libraries such as Cipher. Next, the GSON library allowed me to efficiently parse the enormous JSON file containing quad-gram statistics. The Glide library, recommended by Google, provided support for retrieving images, decoding them, and displaying them within ImageView components.

Despite being the sole developer of this project, Version Control Systems bring numerous benefits. Modern development is a continuous process, and Version Control Systems help develops track changes, collaborate, store versions properly, trace mistakes, and restore previous working versions. (Metwalli 2020). I utilized Git VCS through my GitHub account, https://github.com/healym3.

PACKAGES

Other than the MainActivity, I enclosed all other classes within packages. The "algorithms" package contains the following classes: AES for block encryption, Breaker for breaking a substitution cipher, and Substitution for performing substitution encryption and decryption. The "data" package contains all classes representing a data entity. Next, the "util" package includes the Utils class, which performs JSON to String conversion, text extraction from files, hiding the Soft Keyboard, and displaying Snackbar notifications. Lastly, the "ui" package contains packages for each of the fragments "substitution" with the SubstitutionFragment and SubstitutionViewModel classes, "breaker" with the BreakerFragment and BreakerViewModel classes, "bitmap" with the BitmapFragment and BitmapViewModel classes, the "files" package with the FileEncryptionFragment and FileEncryptionViewModel classes, and "aesvsdes" with the AesVsDesFragment and AesVsDesViewModel. More information on ViewModels, Fragments and Activities will be given in another section.

MAIN ACTIVITY

As described in the Android Developer Documentation, mobile users often begin applications in various ways. In Java and C++, an application starts with the "main" function. Conversely, Android applications launch from an Activity class instance. The Activity class creates the application's User Interface (UI) window. For example, if you open Google Maps, the main Activity is shown with your location and a search bar to find your desired endpoint. However, clicking an address on a webpage would launch the Google Maps app directly into an Activity to begin navigation to that location. In other words, Google explains "the user journey often begins non-deterministically."

In addition to activities, Fragments provide a User Interface that is modular and reusable. While activities contain global elements such as navigation, a Fragment class represents a UI area, screen, or portion of a screen and is always hosted by an Activity class. Each fragment contains its own layout in this project and dedicates itself to specific project tasks. The first fragment focuses on encrypting text with a substitution cipher. The substitution breaker fragment allows the user to break a cipher without knowing the key. Third, the AES bitmap encryption fragment displays the differences between encrypting in Electronic Code Book mode and Cipher Block Chaining mode. The file encryption fragment allows the user to encrypt and decrypt e-mail attachments using AES in Cipher Block Chaining mode. Lastly, the AES vs. DES fragment measures the performance of both algorithms when encrypting files.

The Main Activity is launched first in the application, and its XML layout contains the NavHost, NavController, and Bottom Navigation Bar. The NavController manages app navigation within the NavHost, displaying the desired fragment to the user. Users can select a tool or task at the bottom of the screen on the navigation bar, and that fragment will be loaded into the navigation host fragment display. Using these navigation components facilitates fragment transactions, handling back actions, transitions, and View Model support (explained further in the section titled "View Models"). (Android).

VIEW MODELS

Without a View Model, the fragment loses all operational data any time the user rotates the screen or switches back and forth between another fragment. With each configuration change, the application recreates the View. For example, after creating a Substitution Key for encrypting plain text, the previously created key will be gone if the user switches to the bitmap encryption fragment and back again. Although certain events such as this are outside the developer's control, ViewModels provide a way to manage UI-related data allowing it to survive. Ultimately, Android explains, "It's easier and more efficient to separate view data ownership from UI controller logic." (Android).

To implement a ViewModel class, the class extends ViewModel. Since ViewModel instances are saved from destruction following configuration changes, any stored data remains accessible upon request when a fragment or activity is recreated. Data members cast as MutableLiveData can be observed for changes within the corresponding Fragment classes. Any changes to the data are automatically reflected within the fragment. This capability is beneficial when a ViewModel makes

asynchronous calls to outside data sources. (Griffiths 2021). Details on ViewModel use will be outlined in the corresponding fragment sections.

LAYOUTS

In Android development, layouts are XML files detailing the user interface components and visual elements built under a hierarchy of Views and ViewGroups. The ViewGroup is the container for Views and other ViewGroups, while a View is drawn onto the screen. (Android). A View object can be a Button, ImageView (displays images), TextView (displays text), or even an EditTextView (creates an editable text view, similar to HTML5 form input). Alternatively, a ViewGroup defines different structures for layouts, such as Constraint Layouts, Linear Layouts (vertical or horizontal), and Frame Layouts. (Griffiths 2021).

For the main Activity class and all Fragment classes, I designed all Layout files within a ConstraintLayout ViewGroup. Using a constraint-based layout allows me to organize specific UI components based on where I want them to show up about each other. Suppose the end-user opens the application on a tablet or phone (rotated horizontally or vertically). The layouts will always make full use of the screen space and appear uniformly.

SUBSTITUTION CIPHER

Several files fit together to implement the substitution cipher encryption and decryption of text. The "fragment_substitution.xml" file defines the layout and structure of the fragment's views. All Fragment control logic resides in the SubstitutionFragment class, and key related dated and methods are boxed together within the SubstitutionKey class. When the user wishes to perform encryption or decryption, the Substitution class is instantiated and passed the current SubstitutionKey.

I ran into an issue after implementing the wireframe design and substitution algorithms. After a user generates a key or inputs a custom key via an EditTextView, all data is lost if the user leaves the fragment by choice or accidentally. No information is saved after a user navigates away and the application pauses the fragment. On resuming, the fragment calls onResume(), but no information is restored.

At this point in my project, I began further research on ViewModels and how to use them for observable data that survives configuration changes and pausing. I created the SubstitutionViewModel class to store the SubstitutionKey object. After confirming that the active key remains, I found another issue. The Substitution object takes in a SubstitutionKey and handles encryption and decryption. If a user starts the encryption process and leaves the fragment or rotates their screen, the class cannot receive the ciphertext. Therefore, I added the Substitution object to the ViewModel as observable, MutableLiveData. Once the encryption or decryption completes, the SubstitutionFragment will observe the change and reflect it to the user.

SUBSTITUTION CIPHER BREAKER

I implemented my algorithm based on the research I'd learned about hill-climbing and quadgram fitness evaluation. I created a separate Java command-line program to build the substitution breaking classes, implement the algorithm, and perform testing.

I converted the quadgram input file (containing statistics on English quadgrams), cipher text, plain text, and key to binary to improve efficiency. In the main breaker loop, the function generates a random key. Then, the primary function sends this key to the hill-climbing function. The hill-climbing procedure begins by decrypting the ciphertext with the starting key and creating a binary plain text array. In a 2D array list, I store the positions of characters associated with that key value for every character of the key. As a result, when the function swaps key letters, I only have to adjust the plain text array at those character positions.

Next, the hill climb tries every combination of switching two key characters at a time. After each switch, the function re-calculates the fitness of the plain text. If the fitness is improved, the key keeps the switched characters. If not, the function undoes the changes to the plain text and reverts the key. In the end, the best fitness and best key are returned to the primary breaker function.

This process continues if a better fitness and key continue to be found, but if the fitness doesn't improve three times, a maximum has been hit. At this point, the Breaker uses the best key found to decrypt the ciphertext and returns a String with the plain text.

URI, CONTENT PROVIDER/RESOLVER, INTENT

A Uniform Resource Identifier (URI) is a "compact sequence of characters identifying an abstract or physical resource." Content providers help the Android application interface between its own data storage and data stored elsewhere, and a ContentResolver resolves a URI to a specific ContentProvider. (Griffiths 2021).

An Intent is a messaging object created to request an action from a separate app component. The Intent objects required by this project involved starting an Activity. Regarding file access, creating an Intent with "ACTION_OPEN_DOCUMENT," opens the phone's file browser activity for the user to select a file, and this Activity returns the Uri for the file. Similarly, creating an Intent with "ACTION_SEND_MULTIPLE," allows the user to share files. The application can specify the sharing method as an e-mail or text message by configuring the Intent. (Android).

USERFILE CLASS

Before implementing bitmap encryption using ECB and CBC mode or e-mail file attachment encryption, I needed a class to hold data related to the opening, encrypting, and decrypting of files selected by the user. When instantiating a UserFile object, the constructor is passed Context, the FILE_TYPE (whether it's an original, unencrypted, or an encrypted file), and the file's Uri.

The Fragment creating the UserFile object uses public methods available to set the Algorithm (AES CBC or AES ECB), generate a key, import a Key or IV from the user-

chosen file, initiate encryption or decryption, and get the encrypted and decrypted files. UserFile interfaces with the AES class from the "algorithm" package to perform the encryption and decryption. Before working with any file chosen by the user, the application needs to use the app's ContentResolver to open an InputStream and copy the stream to a File in the app's "files" directory. UserFile also stores the Paths to the original and encrypted file.

BITMAP - ECB VS CBC MODE OF OPERATION

Upon implementing the BitmapFragment, I realized I needed to create a child of UserFile called UserFileBitmap. The UserFileBitmap class inherits all the features of the UserFile class, but this class adds an additional mode of operation for comparison. When the BitmapFragment creates the UserFile with the bitmap chosen by the user, the fragment also initiates encryption. First, the original file is copied to the application's working "files" directory. Then the file is encrypted in Electronic Code Book mode and Cipher Block Chaining mode.

This portion of the project aims to show the user the vulnerability of using ECB mode. However, the application can not display the encrypted files as a Bitmap object in their current form because the 54-byte headers are invalid. Therefore, after encryption, the UserFileBitmap class re-opens the encrypted files as a FileInputStream and replaces their invalid header bytes with valid header bytes. Finally, the application loads the images into the XML Layout's ImageView components using the Glide Image Loader Library for Android.

Once again, if the user accidentally or purposefully leaves the BitmapFragment screen or rotates their device, the UserFile object and its save data regarding the original and encrypted images' Paths is lost. The information persists by storing this UserFile within the BitmapViewModel class as observable MutableLiveData. Plus, as the encryption and header correction occurs asynchronously, the fragment only needs to observe when the process is complete.

FILE ATTACHMENTS ENCRYPTION / DECRYPTION

In the XML layout for the FileEncryptionFragment, the top half of the UI is dedicated to opening a file, generating or importing a key, saving the key to a file, encrypting the original file, and e-mailing the file along with the IV. In the bottom half of the layout, using a similar set of Buttons and TextViews, the user can open an encrypted file, import the private key and IV, and decrypt the file.

By design, the UserFile class can take in either an original file that the user wants to have encrypted or an encrypted file the user has received and needs to be decrypted. Therefore, in the FileEncryptionViewModel, we need two separate UserFile objects: one for original files and the other for encrypted files. When the user clicks the "Open File" button in the "Encryption" section of the UI, after the user chooses a file, a UserFile object is instantiated with the FILE_TYPE of ORIGINAL. This UserFile object is then saved to the ViewModel. Next, a key can be imported by file or generated securely via public UserFile methods. Similarly, when the user clicks "Open File" in the "Decryption"

section of the layout and chooses a file, a UserFile object is instantiated with the FILE_TYPE ENCRYPTED.

As discussed in the section "URI, CONTENT PROVIDER/RESOLVER, INTENT," an Intent object can be created to start an Activity outside the application. The Intent object is instantiated with Intent to compose an e-mail with multiple file attachments.ACTION_SEND_MULTIPLE and next passing “vnd.android.cursor.dir/email” to the object’s setType() method. The app appends Uri objects for each file (encrypted and IV) to an ArrayList. Next using the Intent object’s putParcelableArrayListExtra() method, the fragment passes Intent.EXTRA_STREAM and the Uri ArrayList. Finally, the startActivity(intent) begins the Activity.

DES VS AES PERFORMANCE

In the last Fragment, I created an XML layout with a Button to open a test file and TextView components to display the duration of encryption processes for AES and DES. I also created a specialized UserFile class called UserFileCompare to manage the control logic for taking in the test file, encrypting with DES and AES, and measuring the performance for each. Once this process is complete, the AesVsDesFragment retrieves the test results and displays them with duration measured in milliseconds and file size measured in kilobytes.

Chapter 3. Results

The most up to date version of the application can be found at:
https://github.com/healym3/EncryptionApp2

After downloading the latest Android Studio, you can import from VCS using this link:
https://github.com/healym3/EncryptionApp2.git

TESTING SUBSTITUTION CIPHER

Various keys, ciphertext, and plain text were used to verify the substitution cipher performs accurately. All tests produced correct results, and two specific test cases are listed here:

First, using a substitution key of "BYSZWVFJEKUMNOCRLQXDGATPIH," the following plain text was entered into the EditText box: "Attack the enemy camp at three am in the morning." The expected ciphertext should be: "BDDBSU DJW WOWNI SBNR BD DJQWW BN EO DJW NCQOEOF." After the 'Encrypt' Button was clicked, the ciphertext result was "BDDBSU DJW WOWNI SBNR BD DJQWW BN EO DJW NCQOEOF." After closing the app and re-opening it, this ciphertext was inputted into the cipher EditText component, and the key was re-entered. Following the click of the 'Decrypt' Button, the plain text result was correctly shown: "ATTACK THE ENEMY CAMP AT THREE AM IN THE MORNING."

Second, using a substitution key of "OAQBKYLCDFVIHJZWTXRSNPGMEU" and plain text "retreat," encryption resulted in the ciphertext "XKSXKOS." Again, this result is verified by decrypting with the same key, resulting in the plain text "RETREAT."

TESTING SUBSTITUTION CIPHER BREAKER

A sufficient amount of cipher text is needed to break substitution ciphers accurately. The first test was using text provided by Dr. Emre Yilmaz and the University of Houston-Downtown:

---

msr crkqimurem xz jxukdmri bjvrejr qec renverriven mrjsextxna vb xer xz msirr qjqcruvj crkqimuremb pvmsve msr jxttrnr xz bjvrejrb qec mrjsextxna qm msr devoribvma xz sxdbmxe cxpemxpe. pr xzzri bmdcremb q devwdr xkkximdevma mx qjsvror qjqcruvj bdjjrbb ve jxukdmri bjvrejr qec renverriven mrjsextxna msixdns ivnxixdb qec ryjvmven jxdibrb, vejtdcven decriniqcdqmr irbrqijs xkkximdevmvrb. pr qir jxuuvmmrc mx miqveven bmdcremb zxi jsqttrenrb ve jxukdmri bjvrejr qec renverriven mrjsextxna, qec ve zvrtcb irwdviven msr gexptrcnr qec bgvttb, xdi jdiivjdtdu kixovcrb zxi bmdcremb pvmsve mryqb qec qixdec msr pxitc. msr crkqimurem xz jxukdmri bjvrejrb qec renverriven mrjsextxna vb msr zqbmrbm nixpven crkqimurem pvmsve msr jxttrnr xz bjvrejrb qec mrjsextxna. qccvmvxeqtta, qb q bmdcrem xz msr jbrm crkqimurem, axd pvtt nqve sqecb xe rykrivrejr qec kiqjmvjqt miqveven.

bmdcremb pvtt fr irwdvirc mx kqimvjvkqmr ve q brurbmri kixhrjm, psvjs pvtt miqve bmdcremb mx pxig xe irqt vecdbmia kixftrub ve q mrqu reovixeurem msqm msra pvtt rejxdemri pxigven ve vecdbmia qzmri niqcdqmvxe. msr brevxi kixhrjm qttxpb bmdcremb mx qkkta msrvi jxdibr uqmrivqtb qec trqieven rykrivrejrb mx brurbmri txen irqt pxitc kixhrjmb.

The substitution breaking process needed only 152 milliseconds in 5 rounds (out of the maximum ten thousand), returned the exact key of 'qfjcrznsvhgtuexkwibmdopyal', and scored a final fitness of 104.3103, indicating a close match to English. The resulting decrypted text:

THE DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING TECHNOLOGY IS ONE OF THREE ACADEMIC DEPARTMENTS WITHIN THE COLLEGE OF SCIENCES AND TECHNOLOGY AT THE UNIVERSITY OF HOUSTON DOWNTOWN. WE OFFER STUDENTS A UNIQUE OPPORTUNITY TO ACHIEVE ACADEMIC SUCCESS IN COMPUTER SCIENCE AND ENGINEERING TECHNOLOGY THROUGH RIGOROUS AND EXCITING COURSES, INCLUDING UNDERGRADUATE RESEARCH OPPORTUNITIES. WE ARE COMMITTED TO TRAINING STUDENTS FOR CHALLENGES IN COMPUTER SCIENCE AND ENGINEERING TECHNOLOGY, AND IN FIELDS REQUIRING THE KNOWLEDGE AND SKILLS, OUR CURRICULUM PROVIDES FOR STUDENTS WITHIN TEXAS AND AROUND THE WORLD. THE DEPARTMENT OF COMPUTER SCIENCES AND ENGINEERING TECHNOLOGY IS THE FASTEST GROWING DEPARTMENT WITHIN THE COLLEGE OF SCIENCES AND TECHNOLOGY. ADDITIONALLY, AS A STUDENT OF THE CSET DEPARTMENT, YOU WILL GAIN HANDS ON EXPERIENCE AND PRACTICAL TRAINING. STUDENTS WILL BE REQUIRED TO PARTICIPATE IN A SEMESTER PROJECT, WHICH WILL TRAIN STUDENTS TO WORK ON REAL INDUSTRY PROBLEMS IN A TEAM ENVIRONMENT THAT THEY WILL ENCOUNTER WORKING IN INDUSTRY AFTER GRADUATION. THE SENIOR PROJECT ALLOWS STUDENTS TO APPLY THEIR COURSE MATERIALS AND LEARNING EXPERIENCES TO SEMESTER LONG REAL WORLD PROJECTS.

The second testing text was pulled from the American Kennel Club's "Fun Facts About Golden Retrievers." The text was encrypted with a substitution key of "ZYXWVUTSRQFEDCBAPONMLKJIHG.'

zn nabomrct wbtn, tbewvc ovmorvkvon cvvw ebmn bu vivoxrnv. msvh'ov z szow-jbofrct yovvw. tbewvc ovmorvkvon zov lnvw ubo vkvohmsrct uobd slcmrct zcw mozxfrct mb ovnxlv zcw nvokrxv wbt jbof. wvnarmv mszm aezhule, nbdvmrdvn tbbuh wvdvzcbo, msvh evzoc plrxfeh zcw xzc dznmvo z dlemrmlwv bu nfreen, rcxelwrct sbj mb avoubod zn nvzoxs zcw ovnxlv wbtn zm msv nrmv bu z wrnznmvo. tbewvc ovmorvkvon dzfv mba-cbmxs msvozah wbtn. tbewvcn zov bumvc vdaebhvw zn msvozah wbtn mb nbbmsv zcw xzed avbaev. tbewvcn nvvd mb szkv ovnvokvn bu vdazmsh zcw lcxbcwrmrbcze ebkv. msvov'n qlnm nbdvmsrct zyblm z tbewvc mszm xzc xzed zcw xbdubom zchbcv, uobd hblct xsrewovc mb vewvoeh azmrvcmn. aznm alaahsbbw, tbewvc ovmorvkvon bumvc nmzh hblct zm svzom. hbl fcbj zee mszm vilyvozcm alaah yvszkrbo? mszm mvcwn mb eznm ebctvo rc tbewvc ovmorvkvon mszc bmsvo yovvwn. msvh dzmlov nebjeh zcw xzc yv aezhule zcw wbjcortsm nreeh rcmb zwlemsbbw.

The substitution breaking processes took 34 milliseconds in 5 rounds, returned the correct key 'zyxwvutsrqfedcbaponmlkjihg,' and scored a final fitness of 98.533, indicating a close match to English. The resulting decrypted text:

AS SPORTING DOGS, GOLDEN RETRIEVERS NEED LOTS OF EXERCISE. THEY'RE A HARD-WORKING BREED. GOLDEN RETRIEVERS ARE USED FOR EVERYTHING FROM HUNTING AND TRACKING TO RESCUE AND SERVICE DOG WORK. DESPITE THAT PLAYFUL, SOMETIMES GOOFY DEMEANOR, THEY LEARN QUICKLY AND CAN MASTER A MULTITUDE OF SKILLS, INCLUDING HOW TO PERFORM AS SEARCH AND RESCUE DOGS AT THE SITE OF A DISASTER. GOLDEN RETRIEVERS MAKE TOP-NOTCH THERAPY DOGS. GOLDENS ARE OFTEN EMPLOYED AS THERAPY DOGS TO SOOTHE AND CALM PEOPLE. GOLDENS SEEM TO HAVE RESERVES OF EMPATHY AND UNCONDITIONAL LOVE. THERE'S JUST SOMETHING ABOUT A GOLDEN THAT CAN CALM AND COMFORT ANYONE, FROM YOUNG CHILDREN TO ELDERLY PATIENTS. PAST PUPPYHOOD, GOLDEN RETRIEVERS OFTEN STAY YOUNG AT HEART. YOU KNOW ALL THAT EXUBERANT PUPPY BEHAVIOR? THAT TENDS TO LAST LONGER IN GOLDEN RETRIEVERS THAN OTHER BREEDS. THEY MATURE SLOWLY AND CAN BE PLAYFUL AND DOWNRIGHT SILLY INTO ADULTHOOD.

TESTING ECB VS CBC MODE OF OPERATION

With any input image selected by the user, the comparison between the image encrypted using Electronic Code Book mode and the image encrypted using Cipher Block Chaining mode is striking. While the ECB image always leaks information, the amount of leakage is more significant with larger input files and files containing many pixels of the same color.

Using Microsoft Paint, I created the original file with three shapes of different colors: an oval, square, and star. After encryption, while the CBC mode left the image appearing to be nothing more than static, the ECB mode image reveals all three shapes.

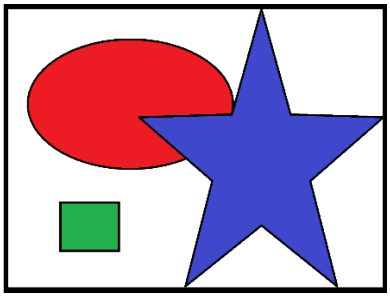


Figure 1, Test #1 - Original Image

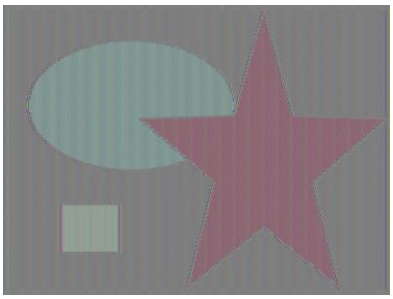


Figure 2, Test #1 - ECB Image

I created a rudimentary paint file containing a mock treasure map for the second test image. It's impossible to look at the CBC image and find clues about what information was held in the original. However, looking at the ECB image, you can make out the basics, including the mountain ranges, the house, and the X marking the treasure's location.

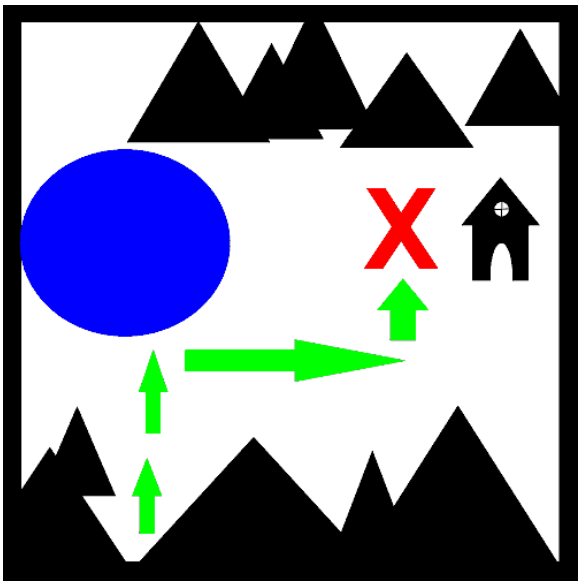


Figure 3, Test #2 – Original Image

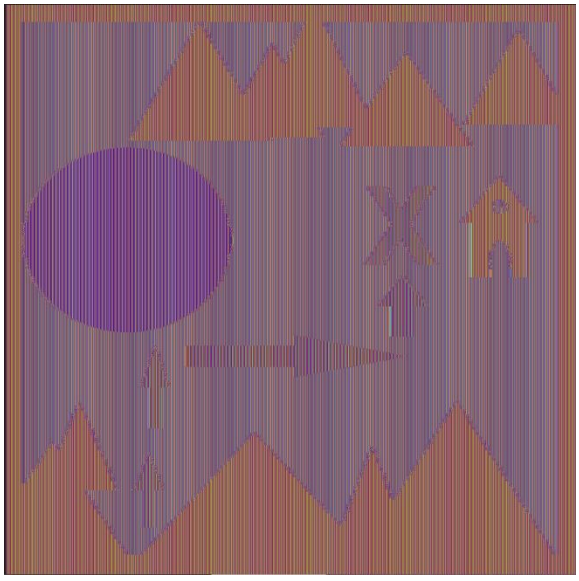


Figure 4, Test #2 - ECB Image

Figure 5, Test #2 – CBC Image

TESTING THE PERFORMANCE OF AES VS DES ENCRYPTION

To gather performance data, I tested files of size (in Megabytes): 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. To create these files, I calculated the file size I needed, added an int into a ByteBuffer called 'buf', rewound the buffer, created the file, set a SeekablyByteChannel to the desired file size, and wrote the buffer 'buf'. (Waldvogel 2019). This file-creating project can be found on my GitHub at: https://github.com/healym3/createTestFilesOfSpecificSize .

The duration of encryption time is measured in Milliseconds. While DES performs worse in every case, the difference grows as the file size increases. This comparison could be explored further with more extensive testing cases.

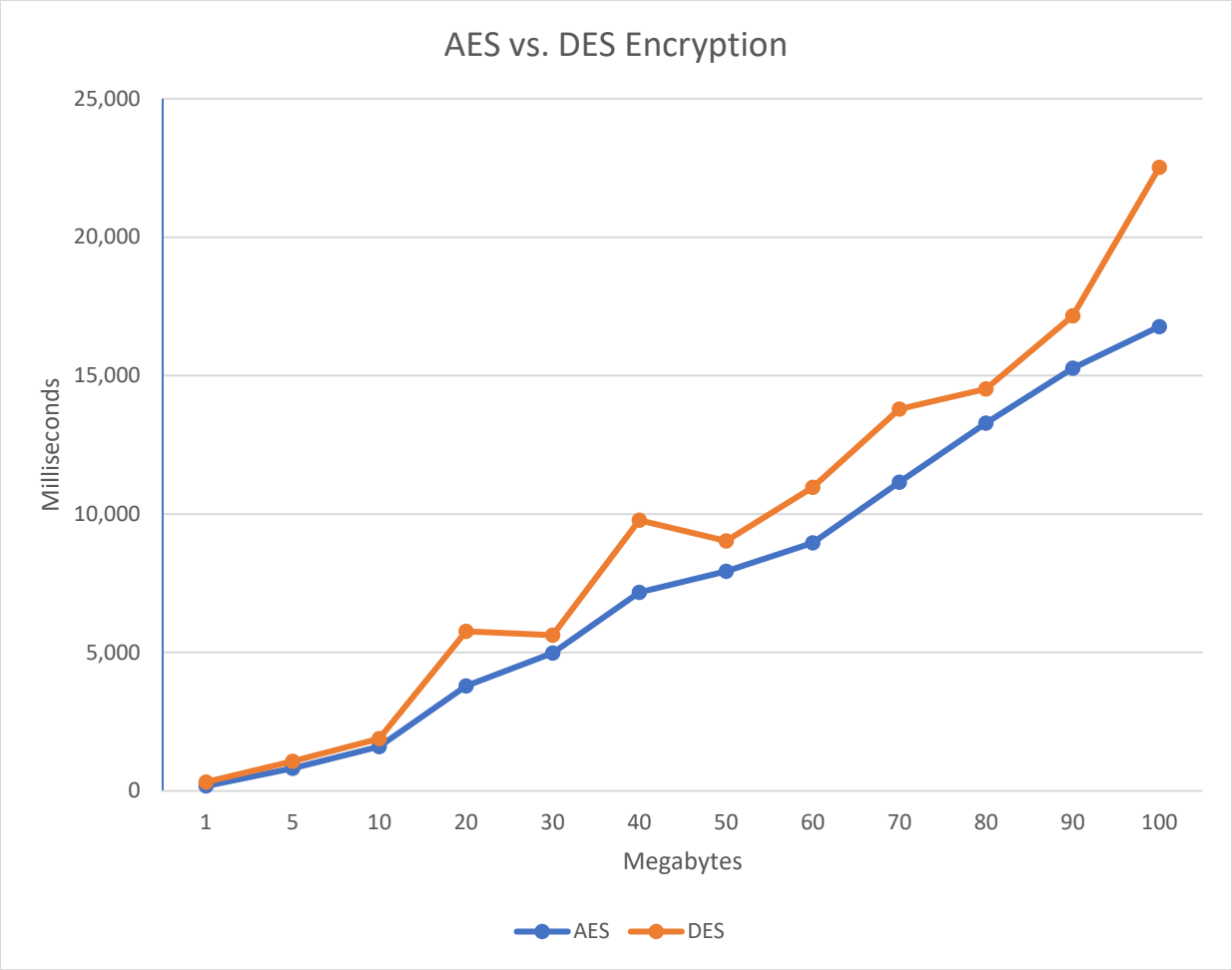


Figure 6, AES Vs DES Encryption Performance Results

References

Waldvogel, Benedikt, and mandev. "Create File with given Size in Java." Stack Overflow, 6 Oct. 2019, https://stackoverflow.com/questions/245251/create-file-with-given-size-in-java.

Al-Sabaawi, Aiman. "Cryptanalysis of Vigenère Cipher: Method Implementation." 2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), 2020, https://doi.org/10.1109/csde50874.2020.9411383.

Alomari, Mohammad Ahmed, et al. "A Study on Encryption Algorithms and Modes for Disk Encryption." 2009 International Conference on Signal Processing Systems, 2009, https://doi.org/10.1109/icsps.2009.118.

Android, Google. "Android Developers." Android Developers, https://developer.android.com/.

Azimi, Mohammadreza, et al. "An Intelligent Key Related Attack Using NSGA II Algorithm." The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012), 2012, https://doi.org/10.1109/aisp.2012.6313737.

Coppersmith, D. "The Data Encryption Standard (DES) and Its Strength against Attacks." IBM Journal of Research and Development, vol. 38, no. 3, 1994, pp. 243–250., https://doi.org/10.1147/rd.383.0243.

Griffiths, Dawn, and David Griffiths. Head First Android Development. O'Reilly, 2021.

"Hill Climbing in AI" Java T Point, 16 Apr. 2019, https://static.javatpoint.com/tutorial/ai/images/hill-climbing-algorithm-in-ai.png.

Inc., Apple. "Apple Developer." Apple Developer, https://developer.apple.com/.

Kuusisto, Finn. "Ancient and Modern Cryptography." XRDS: Crossroads, The ACM Magazine for Students, vol. 21, no. 3, 2015, pp. 57–57., https://doi.org/10.1145/2748053.

"Letter Frequencies in the English Language." Frequency Table, Cornell Math Explorer's Club, 31 July 2015, http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html.

Lyons, James. "Cryptanalysis of the Simple Substitution Cipher." Practical Cryptography, 2013, http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-simple-substitution-cipher/.

Lyons, James. "Quadgram Statistics as a Fitness Measure." Practical Cryptography, 2013, http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/.

Menon, Vipin, and Manuel Rivero. "Developing an IPhone App for a More Safe and Secure Encrypted Conversation/Text/Chat." Journal of Computing Sciences in Colleges, vol. 28, no. 4, Apr. 2013, pp. 75–80.

Metwalli, Sara A. "Version Control 101: Definition and Benefits." Medium, Towards Data Science, 14 Oct. 2020, https://towardsdatascience.com/version-control-101-definition-and-benefits-6fd7ad49e5f1.

Nie, Tingyuan, and Teng Zhang. "A Study of DES and Blowfish Encryption Algorithm." TENCON 2009 - 2009 IEEE Region 10 Conference, 2009, https://doi.org/10.1109/tencon.2009.5396115.

Paar, Christof, et al. Understanding Cryptography a Textbook for Students and Practitioners. Springer Berlin, 2014.

Reisen, Jan. "Golden Retrievers: 10 Fun Facts about the Iconic Scottish Sporting Dogs." American Kennel Club, American Kennel Club, 24 Sept. 2021, https://www.akc.org/expert-advice/dog-breeds/10-facts-about-golden-retrievers/.

Singh, Simon. The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography. Anchor Books, a Division of Random House, Inc, 2000.

Yassein, Muneer Bani, et al. "Comprehensive Study of Symmetric Key and Asymmetric Key Encryption Algorithms." 2017 International Conference on Engineering and Technology (ICET), 2017, https://doi.org/10.1109/icengtechnol.2017.8308215.

Appendix

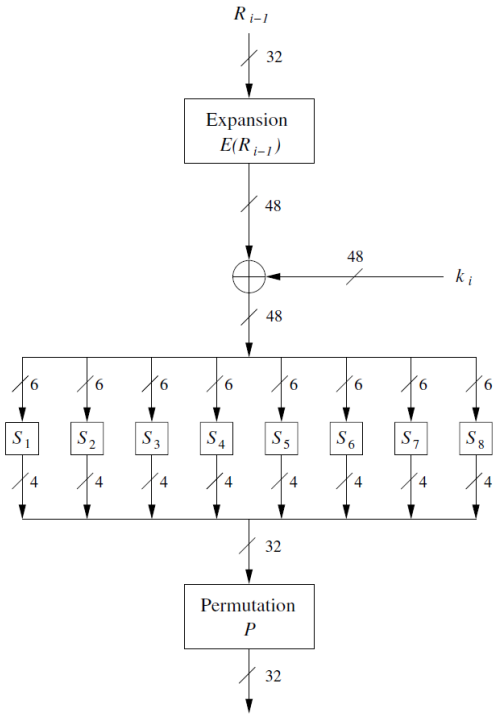


Figure 7, DES f-function (Paar 2014).

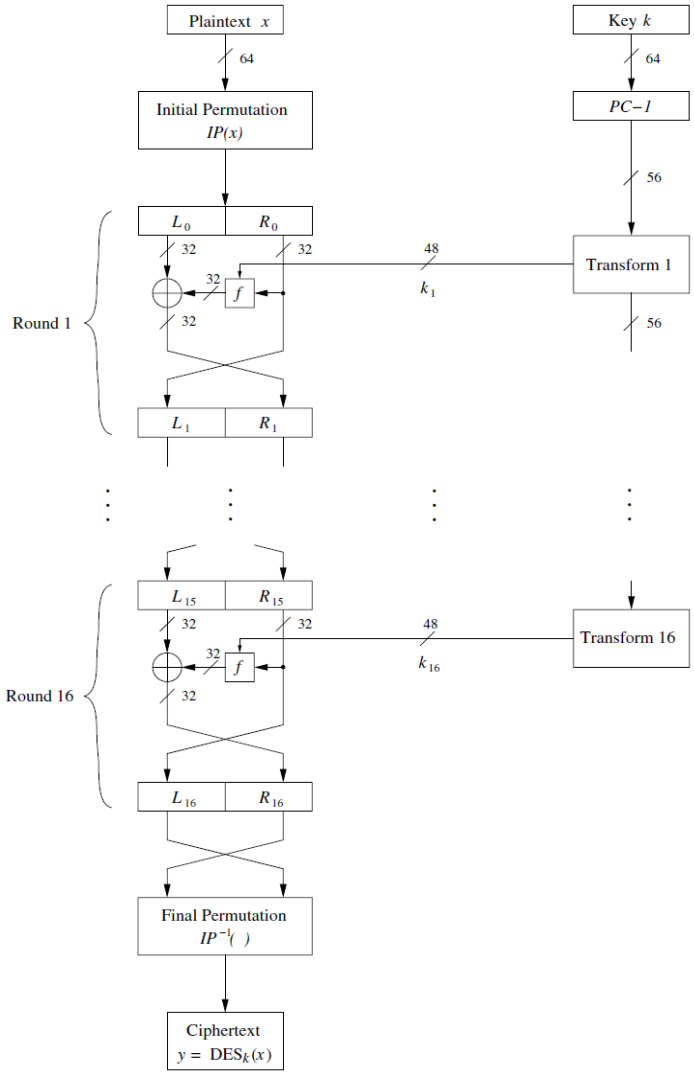


Figure 8, DES Algorithm Overview (Paar 2014).

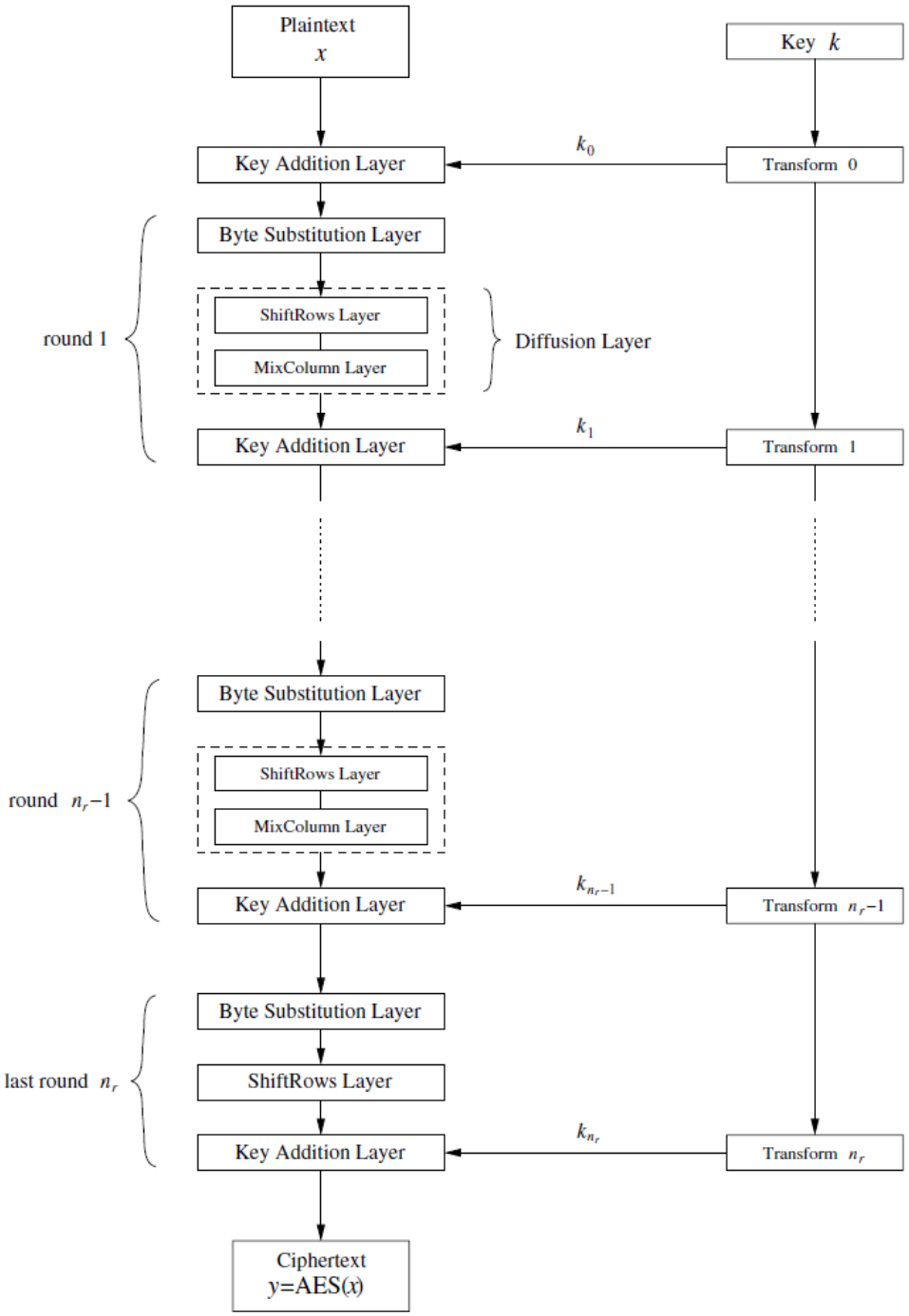


Figure 9, AES Encryption Block Diagram (Paar 2014).

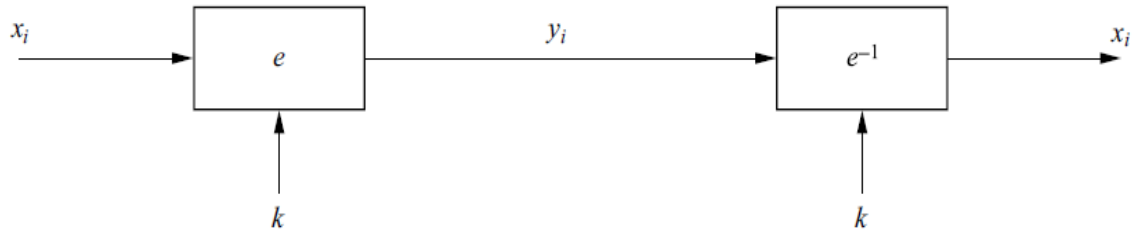


Figure 10, Electronic Code Book ECB Mode of Operation. (Paar 2014).

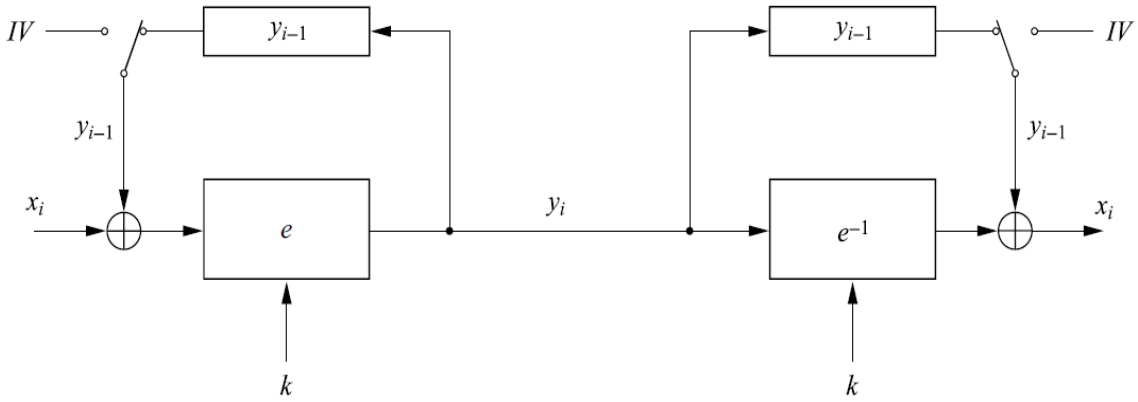


Figure 11, Cipher Block Chaining CBC Mode of Operation. (Paar 2014).

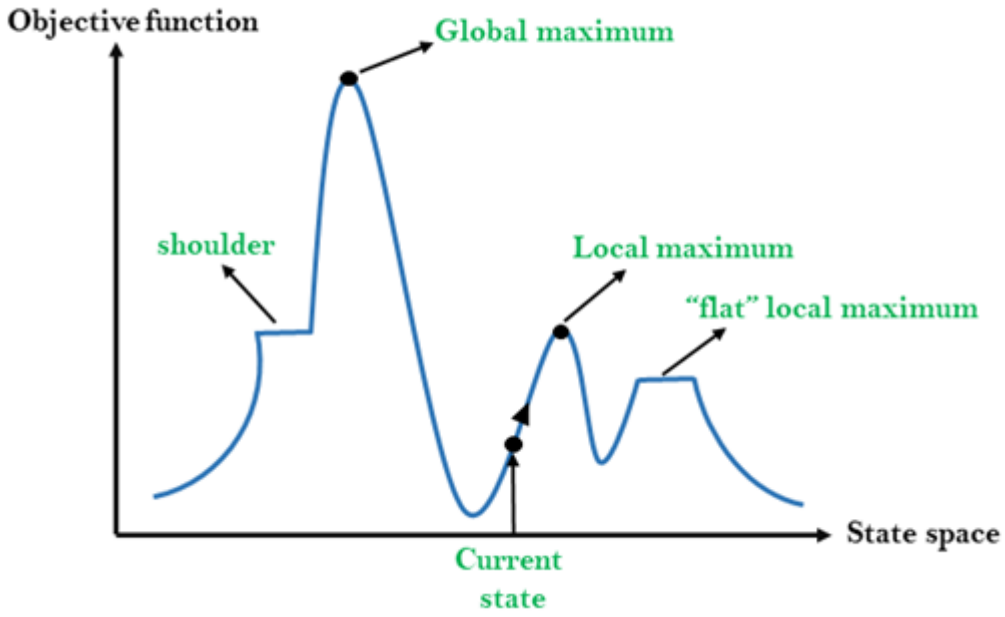


Figure 12, Hill Climbing in AI (Java T Point 2019).

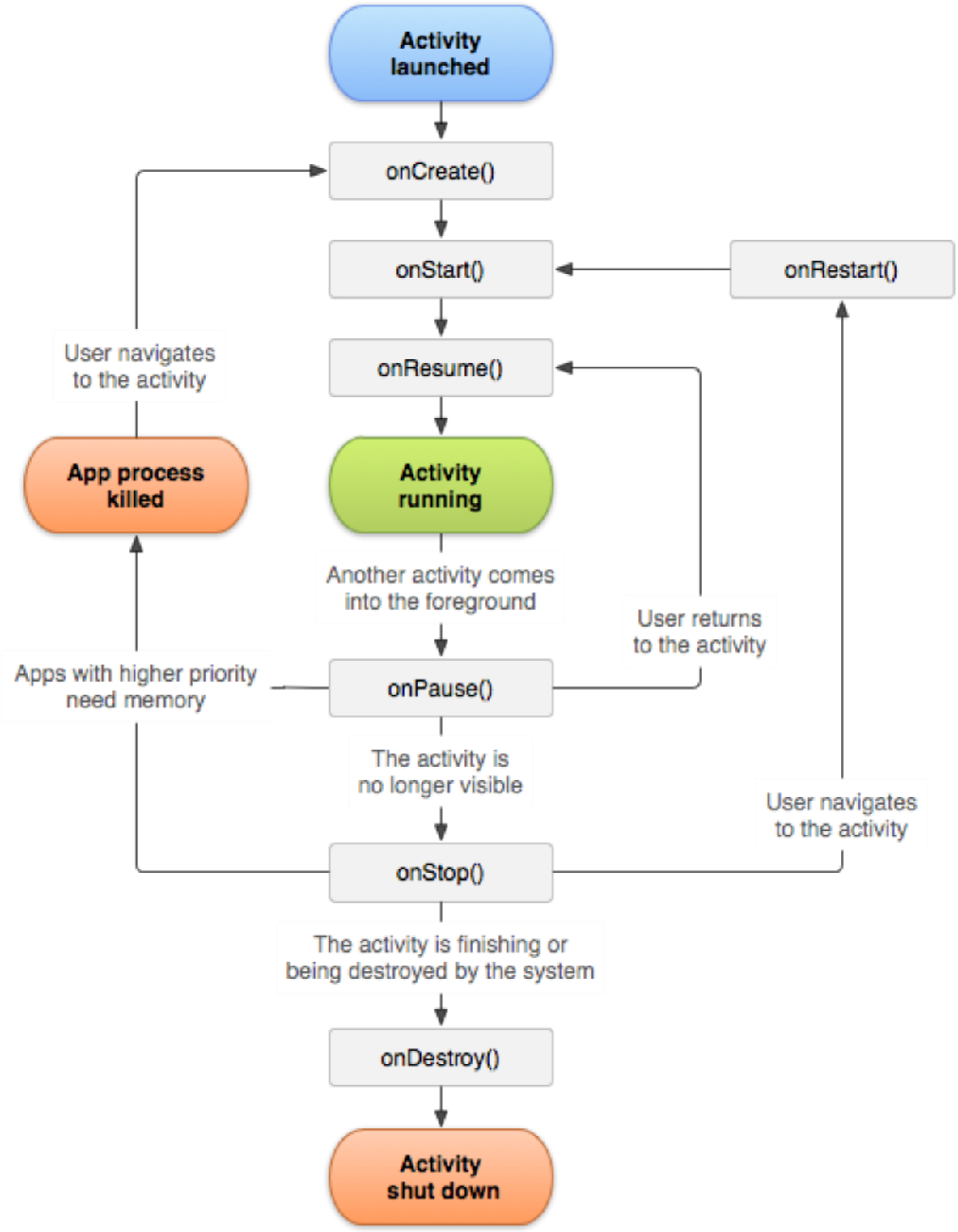


Figure 13, Android Activity Lifecycle. (Android).

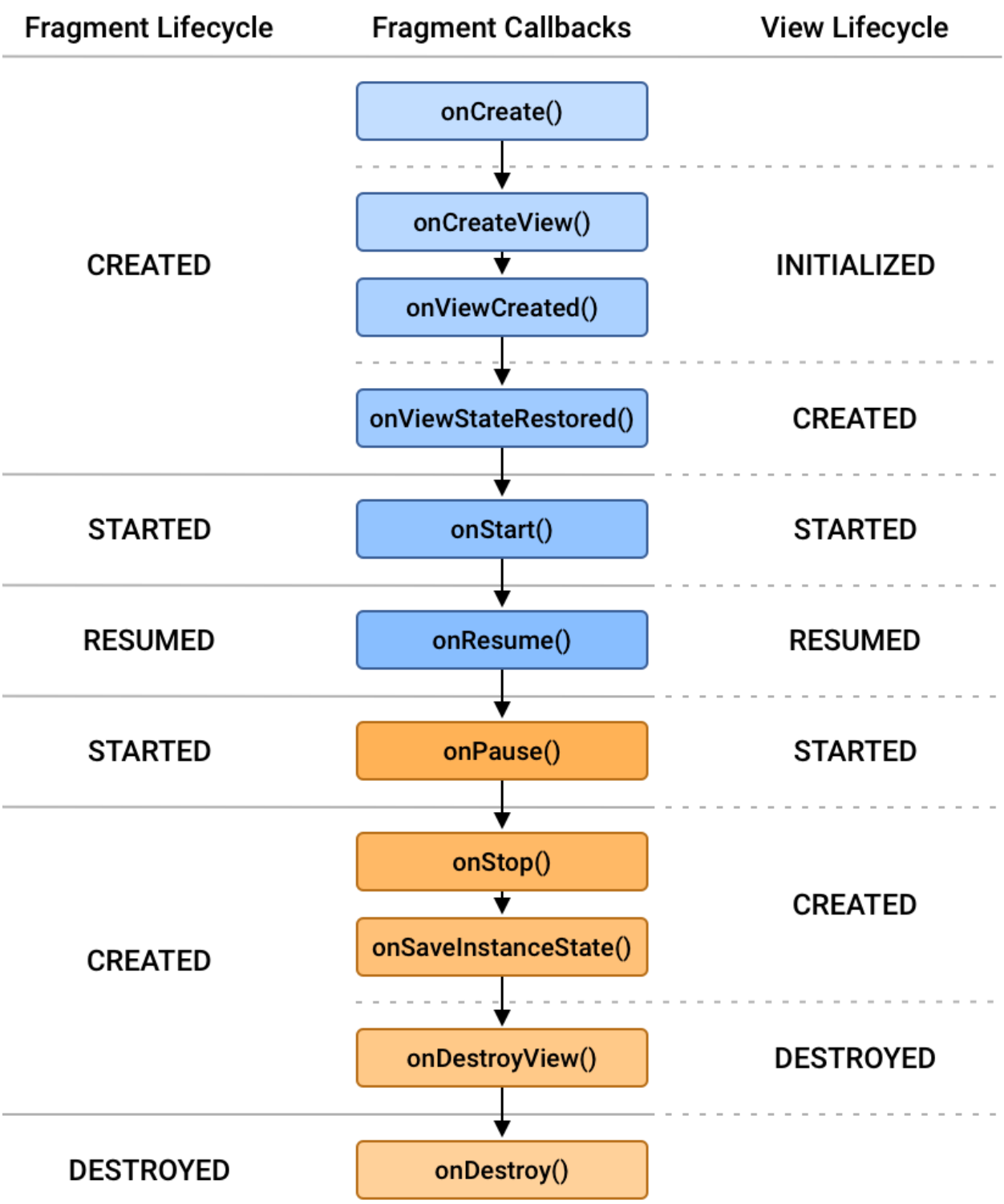


Figure 14, Android Fragment Lifecyle. (Android).