ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΣΦΑΛΕΙΑ ΔΙΚΤΥΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Project

Γιώργος Μαργαρίτης

Contents

Cryptopals in rust

George Margaritis

June 2024

Abstract

This project focuses on solving the Cryptopals' challenges, a set of programming challenges that cover various fields of cryptography and cyber security. These challenges are designed to provide hands-on experience in understanding cryptographic algorithms and breaking cryptographic systems.

1 Introduction

For this project, I used the Rust programming language to solve these problems, focusing primarily on the Electronic Codebook (ECB) and Cipher Block Chaining (CBC) encryption and how to break them.

2 Set 1 - Basics

2.1 Challenges 1-3

In the first 3 challenges, I had to implement basic function such as convert base64 encoding to hex, fixed xor between two buffers of the same size and finally a single byte-xor where the first buffer is being XOR'ed against one byte only.

2.2 Challenge 4

In challenge 4, I was given a text file that contains 60-character strings and I had to find which one has been encrypted by single-character XOR. To solve this challenge, for each string I used brute force checking all bytes possible (0-256) to decrypt it and then pass it into a score function that counts how many ASCII characters it contains. The string with the highest score was the solution to the set.

2.3 Challenge 5 - Implementing repeating-key XOR

In challenge 5, I implemented a repeating-key XOR. With this method, the message and the key can be of different lengths, and the key is repeated to match the length of the message.

2.4 Challenge 6 - Break repeating-key XOR

In challenge 6, I was given a file containing a message that had been base64'd, encrypted with repeating-key XOR and the goal was to decrypt it. First, I had to guess the KEYSIZE, brute forcing values from 2 to 40. For each KEYSIZE, I took the first KEYSIZE worth of bytes and the second KEYSIZE worth of bytes and found the hamming distance between them. The KEYSIZE that produced the smallest distance is most likely the key size. Knowing the keysize, I broke the ciphertext into blocks of keysize length, I transposed the blocks and then I used the method of challenge 4 to find each byte of the key.

2.5 Challenge 7 - AES in ECB mode

In challenge 7, I was given a file containing a base64'd text that has been encrypted via AES-128 in ECB mode with the given key and I had to decrypt it.

2.6 Challenge 8 - Detect AES in ECB mode

In the last challenge of the first set, I was give a file containing hex-encoded ciphertexts and I had to find which one has been encrypted with ECB. To do this, I had to split each ciphertext to chunks of size 16. If a ciphertext had 2 identical chunks then this is the solution. In AES-128 ECB, each message is splitted into blocks of size 16 and then each block is XOR'ed against the key. As a result, if 2 blocks are identical they will produce the same XOR result. This is the weakness of ECB, the results are predictable.

3 Set 2 - Block Crypto

3.1 Challenge 9 - Implement PKCS7 padding

In challenge 9, I had to implement PKCS7 padding. In this padding method, the number of missing bytes until the given length is appended to the block. For example, if the block is "YELLOW SUBMARINE" and the given length is 20 then the block will become "YELLOW SUBMARINE4444" since it misses $20 - 16 = 4$ bytes

3.2 Challenge 10 - Implement CBC mode

In challenge 10, I had to implement CBC encryption.CBC enhances security by chaining the ciphertext of one block with the plaintext of the next block before encryption. This chaining ensures that identical plaintext blocks encrypt to different ciphertext blocks, unlike ECB.

3.3 Challenge 11 - An ECB/CBC detection oracle

A simple oracle that uses a random key, prepends and appends random bytes to a given text and randomly chooses either ECB or CBC. Then another function should detect if the oracle used ECB or CBC (same method as challenge 8).

3.4 Challenge 12 - Byte-at-a-time ECB decryption

In challenge 12, I had to create a function (f1) that appends to my-message an unknown message and then uses an unknown but consistent key to encrypt it in ECB mode. The goal of this challenge is to decrypt the unknown message. After discovering the block size of the cipher, I crafted a block that is exactly 1 byte short (for instance if block size is 8 $\Rightarrow$ "XXXXXXX") and then I encrypted it using f1.After, for each byte (0-256) I put it in the last place of the block ("XXXXXXXA","XXXXXXXB","XXXXXXXC", ...) and compared the encrypted message with the one produced by the 1 byte short block.If the 2 encryptions are identical, then the byte that produced this, is the first byte of the key. Finally, I followed the same process for the rest of the key's bytes.

4 Summary

4.1 ECB

In ECB encryption, the message is splitted into blocks of the same size usually 16,24 or 32. Each block is then XOR'ed against the key. The weakness of this method is that 2 identical blocks will always produce the same encrypted blocks, thus making this method predictable. Finally, ECB is vulnerable to byte-at-a-time attacks where the key can easily be discovered.

4.2 CBC

In CBC encryption, the message is splitted into blocks of the same size usually 16,24 or 32. Each block is first XOR'ed against the previous encrypted block (or the Initialization Vector) and then XOR'ed with the key. With this method 2 identical blocks will NOT produce the same result.However, this method is still vulnerable to padding attacks and bit-flipping attacks.