Areeb Hussain | VC ID: S2100162 | UWE ID: 21023603

Component B – Assignment Report

Introduction

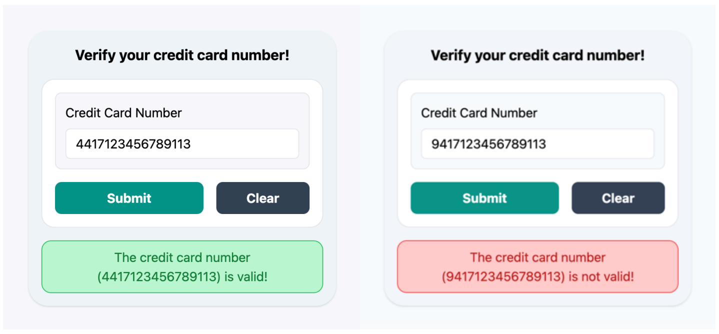
This report explores several programming tasks centered on computer security and cryptography, covering tasks like credit card verification, Hamming code, password cracking and encryption-decryption of messages. Using Python and the Flask framework, the report offers practical implementations and insights into cryptography. It aims to provide an understanding of computer security through tasks and research.

Selection and use of framework

Flask, a Python-based web framework, was chosen for its adaptability and ease of use. It employs MVC design (Data Flair, 2020), enhanced by the templating language Jinja and the styling class library Tailwind. Python libraries like itertools, time, random, and hashlib were used to enrich the app's functionality, making task navigation and interaction more seamless.

Credit Card Number Verification

In this task, 16-digit credit card numbers are validated using Luhn's algorithm in conjunction with input validation. According to Khalid et al. (2013), the Luhn algorithm starts by doubling each alternate digit starting from the next-to-last and going left. The number is reduced by 9 if the result of the doubling is larger than 9. After that, the undoubled digits from the original number are added to the resultant digits. This amount is then divided by 10. If the remainder is zero, it becomes the check digit; if not, 10 minus the remainder is used as the check digit (Khalid et al., 2013). The validation process is initiated with user input on the HTML form, then validated with the 'pattern="[0-9]{16}"' property to verify correct formatting. Upon posting the form, the Flask function passes the value throught the Luhn algorithm to verify the input's validity.


Verify your credit card number!
Credit Card Number
Submit
Clear
Verify your credit card number!
Credit Card Number
Submit
Clear

4417123456789113

The credit card number (4417123456789113) is valid!

9417123456789113

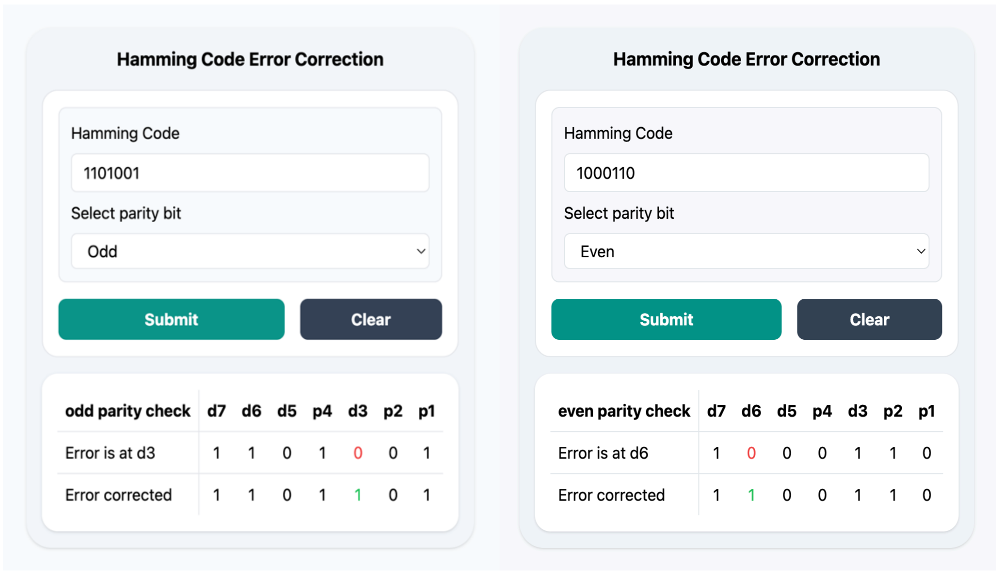
The credit card number (9417123456789113) is not valid!

Figure 1 Credit card number checker app UI

Hamming Code Error Correction

This task is to implement an algorithm utilized in digital communication systems, for error detection and correction in 7 bit Hamming Codes. According to Fauzi et al.s research in 2017 the Hamming code cleverly adds bits to the data sequence to serve both error detection and correction purposes (Fauzi et al., 2017).

A Python-based application was developed using Flask to complete this task. The program features a 'hamming_code' blueprint and two HTTP routes, one for user interface and another for data processing. At its core is the 'error_check()' function, which calculates and compares parity bits. When an error is detected, the function identifies the bit with the error, flips it, and returns a corrected 7-bit string. As shown in Figure 2, this practical approach offers a good web-based solution for the demonstration of the use of Hamming code in error detection and correction.



Hamming Code: 1101001
Select parity bit: Odd

odd parity check	d7	d6	d5	p4	d3	p2	p1
Error is at d3	1	1	0	1	0	0	1
Error corrected	1	1	0	1	1	0	1

Hamming Code: 1000110
Select parity bit: Even

even parity check	d7	d6	d5	p4	d3	p2	p1
Error is at d6	1	0	0	0	1	1	0
Error corrected	1	1	0	0	1	1	0

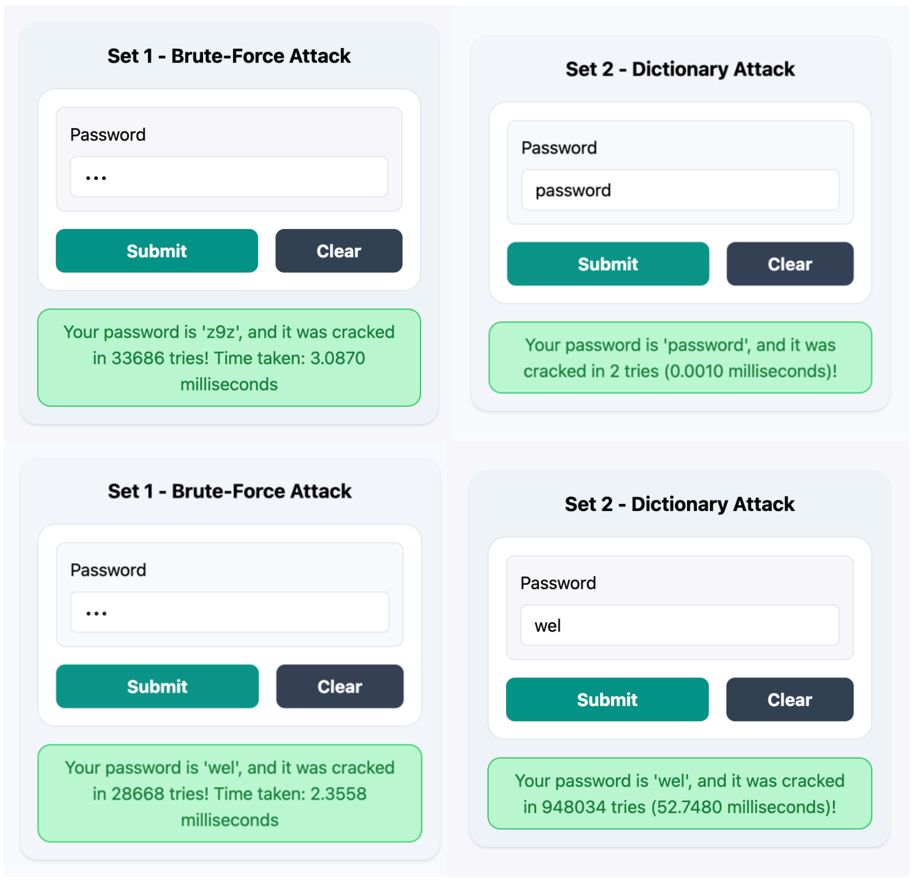
Figure 2 Hamming Code Correction app UI

Brute Force Password Cracking

In Task 3, the focus was on implementing two different types of brute-force attacks for password cracking: Set A and Set B attacks. The Set A attack employed a traditional brute-force method that generated combinations of up to 3 characters, comprised of lowercase letters and numbers. This approach utilized Python's itertools.product for generating permutations and Python's time library for benchmarking. Despite its exponential time complexity, the algorithm proved to be highly efficient for short passwords, completing 46,656 attempts in just 3.8142 milliseconds (Mridha, 2021).

Contrastingly, the Set B attack adopted a dictionary-based method. The algorithm loaded a predefined list of passwords from a text file and then compared the input against this dataset. This algorithm took 53.5181 milliseconds to go through 948,034 entries in the dictionary.

When scaled to 948,034 attempts, Set A would take about 77.06 milliseconds, based on a proportionality factor from its time for 46,656 attempts. The Set A approach was faster but limited by the length and diversity of passwords it could crack. Set B approach, although slower, has the potential to crack more complex passwords, given an a richer dictionary. Figure 3 reveals varying speeds between dictionary and other attack methods.


Set 1 - Brute-Force Attack
Password
Submit
Clear

Your password is 'z9z', and it was cracked in 33686 tries! Time taken: 3.0870 milliseconds

Set 2 - Dictionary Attack
Password
password
Submit
Clear

Your password is 'password', and it was cracked in 2 tries (0.0010 milliseconds)!

Set 1 - Brute-Force Attack
Password
Submit
Clear

Your password is 'wel', and it was cracked in 28668 tries! Time taken: 2.3558 milliseconds

Set 2 - Dictionary Attack
Password
wel
Submit
Clear

Your password is 'wel', and it was cracked in 948034 tries (52.7480 milliseconds)!

Figure 3 Brute force attack app UI

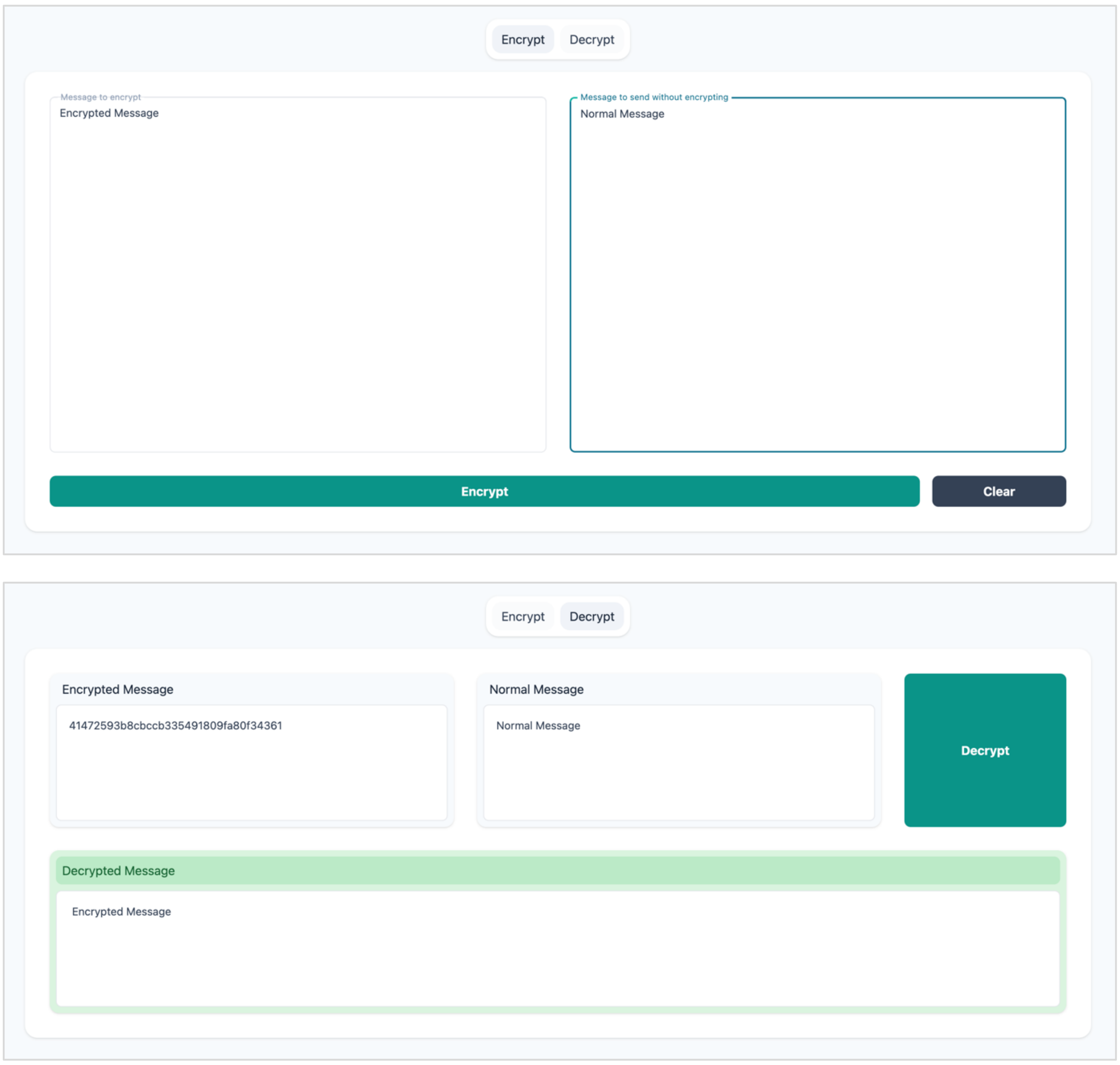
Text Encryption Application Using Stream Cipher and Steganography (30 marks)

The main objective of this task was to create a web application that allows users to encrypt and decrypt text messages using a symmetric key algorithm. This application includes a hashing mechanism for data integrity verification. A thorough evaluation of existing encryption and decryption techniques was carried out before the developing this application.

How can a message be encrypted and decrypted?

> There are three main types of cryptographic algorithms as follows. Secret Key, Public Key, and Hash Functions. Secret Key Cryptography uses one key for both encrypting and decrypting data but is vulnerable to transmission errors. Public Key Cryptography uses two different keys for encryption and decryption, reducing the need for a third party. Hash Functions don't use keys but create a fixed-length hash value for verifying data integrity, often used in password security and file verification (Goshwe, 2013).
>
> According to the paper authored by Goshwe (2013), the RSA (Rivest-Shamir-Adleman) algorithm serves as an effective method for public-key cryptography, which used two distinct keys for encryption and decryption. The algorithm permits the sender to generate a pair of keys, from which, one is public and used for encrypting the message.

The application utilizes Flask to set up routes for text encryption and decryption. Upon accessing the encryption page, a 128-bit key is generated and stored for later decryption. Users input two messages; the first is encrypted using an XOR cipher and a generated key. Both messages are then hashed using SHA-1 for data integrity. This hashed data is sent to the decryption page, where integrity is verified by comparing hashes. If validated, the original message is decrypted using the shared key and displayed. The UI is shown in the figure 4.



Message to encrypt: Encrypted Message

Message to send without encrypting: Normal Message

Encrypted Message: 41472593b8cbccb335491809fa80f34361

Normal Message: Normal Message

Decrypted Message: Encrypted Message

Figure 4 Encryption-decryption app UI

Reference

"Flask vs Django- The Hot Debate of Python Development Section" (2020) Data-flair.training [online]. Available from: https://data-flair.training/blogs/flask-vs-django/ [Accessed 9 September 2023].

Khalid, W., Fazlida, N., Sani, M., Mahmod, R. and Taufik, M. (2013) Enhance luhn algorithm for validation of credit cards numbers [online]. Available from: https://ijcsmc.com/docs/papers/July2013/V2I7201373.pdf [Accessed 6 September 2023].

Fernando, J. (2015) Luhn Algorithm: Uses in identity verification for credit cards Investopedia. June 18, 2015 [online]. Available from: https://www.investopedia.com/terms/l/luhn-algorithm.asp [Accessed 9 September 2023].

Fauzi, A., Nurhayati, & Rahim, R. (2017) Bit Error Detection and Correction with Hamming Code Algorithm. International Journal of Scientific Research in Science, Engineering and Technology, 3(1), pp. 76 [online]. Available from: https://www.researchgate.net/publication/325083029 [Accessed 6 September 2023].

Mridha, P. and Datta, B.K. (2021) An algorithm for analysis the time complexity for iterated local search (ILS) Questjournals.org [online]. Available from: https://www.questjournals.org/jram/papers/v7-i6/I07065254.pdf [Accessed 6 September 2023].

Goshwe., N.Y. (2013) Data encryption and decryption using RSA algorithm in a network environment Ijcsns.org [online]. Available from: http://paper.ijcsns.org/07_book/201307/20130702.pdf [Accessed 7 September 2023].