

1. Introduction

Coding represents the process of encoding and decoding data. Coding is divided into three categories: **error correction**, **cryptography**, and **data compression**. Data integrity is served by error correction, security is provided by cryptography, and memory space is saved by data compression. This report will cover error correction and encryption, as well as provide several example applications and test results from the practical sheets. The report includes an additional examination of Cryptography's integration with other modules. It demonstrates an example of encryption/decryption process used for chatbots in AI and Ethical use and construction.

2. Error correcting

When sending messages, error-correcting codes are used. Changes in the original message could be caused by external factors or human error. The goal of error-correcting codes is to encode data in such a way that the original message can be recovered even if errors occur. These codes are particularly useful when the message can only be sent once.

The International Standard Book Number is one example (ISBN) of error correction. The ISBN is a 10-digit code in which the first three digits provide information about the book, the next six digits are assigned by the author, and the last digit is used to detect errors.

Example of ISBN: **0-19-964398-3**

The **first** digit stands for the **language** of the book (0 is for English).

The **next two** digits hold information about the **publisher** (19 is for Oxford Uni. Press).

The **last** digit is worked out by: $d_{10} = (d_1 + 2d_2 + 3d_3 + \dots + 9d_9) \bmod 11$.

When the value of 10 is assigned to d_{10} , it is written down as "X".

The ISBN program's checking results indicate whether the given ISBN number are accepted or failed verification, rather than the program's overall success. All test cases were successfully authenticated by my program:

Test Case	Result
99921-58-10-7	Test Pass Successfully
99921-58-10-2	Test Pass Successfully
960-425-059-0	Test Pass Successfully
960-455-059-0	Test Pass Successfully
1-84356-028-3	Test Pass Successfully
0-943396-04-2	Test Pass Successfully
0-9752298-0-X	Test Pass Successfully
5-9752298-0-X	Test Pass Successfully

Pseudo code for ISBN verification app:

- 1) Get input string of length 10.
- 2) Separate input into chars Array.
- 3) $\text{Sum} = (d1 + 2*d2 + 3*d3 + 4*d4 + 5*d5 + 6*d6 + 7*d7 + 8*d8 + 9*d9 + 10*d10)$.
- 4) $\text{Sum} = \text{Sum} \bmod 11$.
- 5) Check if the result equals 0:
 - If yes, code is valid and display a validation message.
 - If no, code is invalid.

A similar example for error correction represents the **Luhn's algorithm**, which is used to generate credit card numbers. Credit cards have 16-digit numbers, with the first six numbers containing information about the issuer, the next nine containing the bank account number, and the final digit the error correction.

1 2 3 4 5 6 7 8

Example of Credit card number: 4552-7204-1234-5693

- The **issuer identifier number** is represented by the **first six digits**.
- The **next nine** characters are the **account number**.
- The last digit represents **error detection** and is generated by the **Luhn's algorithm**.

Luhn's Check

- Numbers at **even positions** are summed: $5 + 2 + 2 + 4 + 2 + 4 + 6 = 25$.
- Numbers at **odd position** are multiplied by 2. If the multiplication result equals or exceeds 10, the multiplied value is subtracted by 9. ($4 \times 2 = 8$; $5 \times 2 = 10$ $10 - 9 = 1$; $7 \times 2 = 14$ $14 - 9 = 5$; $0 \times 2 = 0$; $1 \times 2 = 2$; $3 \times 2 = 6$; $5 \times 2 = 10$ $10 - 9 = 1$; $9 \times 2 = 18$ $18 - 9 = 9$) $= 8 + 1 + 5 + 0 + 2 + 6 + 1 + 9 = 32$
 $25 + 32 = 57$ \Rightarrow the **last digit should be 3, to get a total sum that's perfectly divisible by 10**.
 $57 + 3 = 60 \bmod 10 = 0$ \Rightarrow This is a valid credit card number

The Credit Cards Verification application was tested to see if the verification of the given numbers was successful, not the overall success of the program. All test cases were successfully authenticated by my program.

Test Case	Result
4552-7204-1234-5693	Test Pass Successfully ($60 \% 10 = 0$)
4552-7204-1234-5698	Test Pass Successfully ($65 \% 10 \neq 0$)

8552-7204-1234-5693	Test Pass Successfully (64%10 != 0)
5169-7662-2129-0945	Test Pass Successfully (70%10 = 0)
6169-7662-2129-0945	Test Pass Successfully (71%10 != 0)
5169-7662-2129-9945	Test Pass Successfully (79%10 != 0)

The two examples provided can detect errors, but they can only correct them when they are at the last digit. The Hamming and BCH codes are examples of more efficient error-correcting codes. Their names are derived from the names of their creators. Hamming codes detect and correct a single error, whereas BCH codes can detect and correct multiple errors. Hamming (7, 4) uses binary numbers, while Hamming (10, 8) uses decimal numbers. Hamming (8, 4) is extended to detect double and single errors but can only correct one error. BCH (6, 10) can detect multiple errors in the code but can only fix up to two.

The algorithms of the Hamming and BCH codes are very similar. Both of their encodings are accomplished by generating parity check digits from given formulas. Some BCH codes are unusable under certain conditions. The decoding process consists of generating syndrome values that indicate whether or not an error has occurred. There are no errors in either example when all of the syndromes equal "0." The difference between them is that Hamming codes only use the values of the syndromes to detect and correct errors. BCH codes, on the other hand, generate new values (P, Q, R) from the syndromes and use these values to determine whether they are dealing with one, two, or three errors. Their error correction is computationally heavier because more mathematical operations are performed, but they are also more efficient because of their larger scale.

Test Cases for BCH Encoding Program:

Test Case	Result
000001	0000017671
000002	0000023132
000010	0000101974
000011	0000118435
000003	Unusable number
000009	Unusable number

The BCH (6, 10) program successfully recognizes one, two, and more errors, according to testing. It can also correct up to two errors.

Input	Output	Results
3745195876	3745195876	No errors
3945195876	3745195876	Single error
3745995876	3745195876	Single error
3715195076	3745195876	Double error
0743195876	3745195876	Double error
3745195840	3745195876	Double error
2745795878	-	More than 2 errors
8745105876	3745195876	Double error
1145195876	3745195876	Double error
3745191976	3745195876	Double error
3745190872	3745195876	Double error
3745102876	3745195876	Double error
3742102896	-	More than 2 errors
1115195876	-	More than 2 errors
3121195876	-	More than 2 errors

Extra testing on test cases that contain more than two errors:

Test Case	Test Result
3121195876	Pass
1135694766	Pass
0888888074	Pass
5614216009	Pass
9990909923	Pass
1836703776	Pass
9885980731	Pass

3. Cryptography

Data should not only be correct but also secure during transmission. That is where cryptography and encryption algorithms come in handy. SHA-1, MD5, RSA, and SHA-256 are some of the more popular ones. All of the applications demonstrated in this report use SHA-1 encryption. Encryption is accomplished by hashing a given plain text until it is no longer recognizable or meaningful. The hash must be deciphered in order to recover the original plain text.

The first two applications for demonstration use *Brute Force* deciphering or “cracking passwords”. The goal of program A is to decipher passwords of up to six bits in length that can contain both **numbers (0-9)** and **lowercase letters (a-z)**. The program generates random strings

of varying lengths within the specified range, hashes each string, and compares it to the list of targets. Program B's goal is to decipher three valid BCH codes. It generates random six-digit strings and checks their parity. Checks to see if the result is a valid BCH code, and if so, compares its hash to the list of targets.

The possibility of randomly generating a specific password may appear unlikely, but both programs achieved their objectives in a reasonable amount of time. Within a 4-hour run, Program A had cracked all twelve passwords. For about three seconds, Program B breaks all passwords. The disadvantage is that different iterations of the algorithm may repeat some of the random strings. When generating consecutive strings (e.g., 000, 001), the position of the target affects the execution time – "111" takes longer than "000".

Pseudo Code for program B:

- 1) Generate a random 6-bit string of digits.
- 2) Encode the string, generating the four parity digit.
- 3) Check if the generated BCH code is valid.
 - If yes, hash it and go to step 4)
 - If no, go back to step 1)
- 4) Compare the produced hash with the targets list.

Creating a Dictionary is another method for deciphering hashes. The dictionary saves passwords and hashes but is limited to a set of passwords. It is not a very convenient method because it consumes a lot of hard drive space. The Rainbow Tables method improves the use of hard drive space. Rainbow tables save space by constructing password chains and only storing the first and last strings.

Pass 1	Pass 2	Pass 3	Pass 4	Pass 5	Pass 6	Pass 7
--------	--------	--------	--------	--------	--------	--------

Chains are primarily built using two types of functions: hashing and reduction. The hashing function is simple: it generates hashes of a given plain text. The reduction functions use a set of mathematical operations to generate new plain text from hashes. The name "Rainbow" is derived from the various reduction functions used throughout the table, which resemble a rainbow.

Start	->	Reduction1	->	Reduction2	->	Reduction3	->	Reduction4	->	End
Plain text	Hash	Plain text	Hash	Plain text	Hash	Plain text	Hash	Plain text	Hash	Plain text

The last task involves writing a text encryption application using a stream Cipher and steganographic techniques. This app combines cryptography and steganography. It conceals an encrypted message within a regular message using a pseudo random number generator for the stream cipher. Assuming the first message is "How are you today? I had a very busy day! I

travelled 400 miles returning to London. It was windy and rainy. The traffic was bad too. I managed to finish my job, ref No 3789. But I am really tired. If possible, can we cancel tonight's meeting?" and the second one "*meet@9*", after encrypting the second message with the stream cipher, we will obtain:

	0x6d6565744039	(meet@9)
\oplus	0xa73e80e2b563	(a random key stream)
	0xca5be596f55a	(encrypted message)

0xca5be596f55a in binary is 1100 1010 0101 1011 1110 0101 1001 0110 1111 0101 0101 1010.

Following encryption, we will use a steganographic technique to "hide" the encrypted message in the first message and obtain the cipher text. In the reverse order, we must obtain the initial message, message1, and the encrypted message, message2.

Pseudo Code for Hiding text:

- 1) Add fullstop at the end of original message
- 2) Go through the original message
 - a. if character from position p is "1", then we insert at a random position the character " " (white space).
 - b. if character from position p is "0", then we insert at a random position the character ".".
- 3) Repeat Step 2) until the end of the string is reached.

After the encrypted message has been hidden, another message containing the encrypted message will be displayed:

A possible cipher-text:

How. are y ou today? I had a very busy day! I travel.led 400 miles
 return i n.g to London.. It was windy and r a.in y. The t.raffic. .was bad
 too .I m.a.naged to finish my j ob., ref. .No 37.89 .. But .I am re ally tire d.
 I.f possible., can we. can cel tonig ht's me.eting?.

4. Combination of Cryptography and Error correction

Error correction and cryptography frequently collide. Maiorana and Ercole's Biometric Authentication System is one example (2007). The method is based on user-adaptive error-correcting codes that secure the stored templates and cancel them if necessary, such as when a malware attack is detected. Error correction ensures the integrity of the stored templates. The use of distributed cryptography provides hierarchical key management services while also

improving system fault tolerance. The system's testing results in success, primarily for signature and iris identification.

5. Combining Cryptography with different modules

Most aspects of computing and computer science can be combined with cryptography. Encryption provides the security that many applications require. Applications that use personal data must encrypt it in their database in order to be ethical. Decryption contributes to application maintenance by allowing developers to decipher and analyze gathered data.

The following example combines Cryptography, Artificial Intelligence, and Ethics modules. The use of AI chatbots has grown significantly in recent years. They improve communication between app developers and customers. To be ethical, chatbots must never store customer conversations in their database. In that case, how can developers tell if the chatbot is performing well? Encryption is a simple solution to this problem. With encryption, the chatbot can store customer conversations while remaining ethical, as long as the conversations are encrypted before storage.

Algorithm for encryption of conversations:

Crypto → **c + r + y + p + t + o** → **sha1(c) + sha1(r) + sha1(y) + sha1(p) + sha1(t) + sha1(o)**

Every word in the conversation is broken down into chars. Each character is encrypted separately, and the total length of the word is saved as a passive key. These keys are only useful for reassembling the text after deciphering and have no bearing on security. The decision to hash individual characters was motivated by the desire for faster brute force decryption.

Hash 042dc4512fa3d391c5170cf3aa61e6a638f84342




Regardless of their plain text, all hashes generated by the same encryption algorithms have the same length. The length of SHA-1 is 40 bits. Anyone with this knowledge could decipher and separate the big hash into individual hashes. Because the hashed password space is 1 in this case, that would be especially simple (individual chars). It is designed to be quick. To address this issue, hash bits are shifted to specific key positions before being integrated into the big hash.

This is what a big hash looks like:

7cf184f475c67a68d5828329ecb19349720b0cae58e6b3a4ba14a112e090df7fc6029add0f3555cc0
 7c342be856e56f00e7f4347842e2e21b774e61d07c342be856e56f00e7f4347842e2e21b774e61d7a
 81af3e7d591a83c713f8761ea1efe93dcf36150ab8318a1bcaf639e678dd3d02e2b5c343ed4111ca7
 3ab6...

This application's decryption algorithm extracts the big hash from the database and breaks it down into individual hashes of 40 bits each. The gathered hashes were previously shuffled using a specific key pattern. Using the same pattern, each bit returns to its original location.

042dc442512fa3d39143c5170cf3a8f8a61e6a63 → Shuffled hash
 042dc4512fa3d391c5170cf3aa61e6a638f84342 → Original hash



The concept of hashing individual characters stems from the need for fast decryption of large texts (such as chatbot conversations). Brute forcing 1bit strings is an extremely fast and simple deciphering implementation, even on large text files. The algorithm decrypts letters while keeping their order. Once all of the letters have been converted to plain text, words are formed using the passive key values - the length of each word in the original text. The system recognizes all punctuation marks and will not be broken if unknown symbols are entered with the text. It also successfully restores all capital letters and punctuation, preserving the integrity of the data.

6. Conclusion

Cryptography and error correction applications are already in use in our daily lives. Error correction ensures the integrity of our data, while cryptography protects it. The convergence of their applications is extremely beneficial, particularly in automated identification using biometrics. This module's knowledge was successfully applied to a large-scale project, combining aspects from two other modules as well.

(Total words count: 1918)

7. References

- Maiorana, E. and Ercole, C., (2007). Secure biometric authentication system architecture using error correcting codes and distributed cryptography. *Proceedings of Gruppo nazionale Telecomunicazioni e Teoria dell'Informazione*, pp.1-12.
- Jain, A.K., (2004) *An introduction to biometric recognition*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, No. 1