CMPG215: Encryption project

Group 26 :

Wiid de Wet	43700292
Danika le Roux	41049764
Llewlyn Kruger	41186427
Jeanluc Begue	40779173
Connor Langley	41223268

Decryption method:

DES (Data encryption standard) - JAVA

Why we chose DES:

Since its creation in the 1970s, the Data Encryption Standard (DES), a symmetric-key encryption technique, has been widely utilized for safe data transport and storage. DES may be preferred to other encryption techniques for a number of reasons:

1. **Security:** The DES encryption algorithm has withstood the test of time and has received extensive research. While various flaws have been found over time, they have generally been addressed using longer key lengths and more secure implementations.
2. **Compatibility:** DES is extensively supported by a large range of hardware and software products, making it an effective option for system compatibility.
3. **Efficiency:** DES is a rather speedy method of encryption that can swiftly encrypt and decrypt data, making it appropriate for applications that demand high performance.
4. **Regulatory compliance:** Government regulations demand the use of particular encryption techniques, like DES, in several sectors, including finance and healthcare. There might not be a choice but to utilize DES in certain circumstances.

Advantages of DES	**Disadvantages of DES**
Des has been around for a very long (since 1977) and no real weakness has been discovered, brute force is still the best option to break through it.	DES was not designed to encrypt applications therefore it runs very slowly when doing so.
Users can use a similar system and work individually.	Due to improvements in Computation speeds, DES can be easily brute-forced.
Because DES is the United States standard it is revisited every 5 years and was updated if necessary until 2005.	DES is susceptible to Cryptanalysis, which is deciphering coded messages without the encryption key.
DES allows the option to save an encrypted file that can only be accessed using a password.	Because DES is symmetric it only generates one private key so if that key is lost we cannot receive the data on the other end.

<u>Own explanation of the algorithm (generic):</u>

1) A 56-bit key is generated from a password or passphrase.

2) The 64-bit block of data to be encrypted is divided into two 32-bit halves.

3) The data goes through 16 rounds of encryption. Each round involves taking the right half of the data, expanding it to 48 bits, and XORing it with a 48-bit subkey derived from the primary key.

4) The result of the XOR (logical operator Exclusively OR) operation is passed through a substitution process using fixed tables called S-boxes.

5) The output of the substitution process is permuted using a fixed table called the P-box.

6) The permuted output is XORed with the left half of the data.

7) The left and right halves of the data are swapped, and the process is repeated for the next round.

8) After 16 rounds, the final 64-bit block is permuted to produce the encrypted data.

The decryption process is essentially the same as the encryption process but with the subkeys used in reverse order. The main idea is that the data is shuffled and scrambled in a way that is difficult for someone without the key to decipher.

Our programme:

main():
The password variable is set to the string "password". This password will be used to create the encryption key for AES.
The file variable is set to a File object representing the file to be encrypted. This file will be read in and encrypted.

The getSHA() method:
 is called to create a hash of the password. This hash will be used as the encryption key for AES.
A new SecretKeySpec object is created with the hash as the key and "AES" as the algorithm name. This object will be used to initialize the Cipher object for encryption.
An instance of the Cipher class is obtained for AES encryption using the Cipher.getInstance() method.
The Cipher object is initialized for encryption with the SecretKeySpec object and the encryption key.

The encryptFile() method:
is called with the Cipher object and File object as arguments.
encryptFile(Cipher cipher, File file):

The input variable is set to an array of bytes containing the contents of the specified File object. This is done using the Files.readAllBytes() method.

The doFinal() method:

 of the Cipher object is called with the input array as input to encrypt the bytes.
The encrypted bytes are stored in the output variable.
A new FileOutputStream object is created with the file path of the original file plus the extension ".enc". This is the file where the encrypted bytes will be written.
The encrypted bytes are written to the output file using the write() method of the FileOutputStream object.

The original file is deleted using the delete() method of the File object. If the deletion is successful, a message is printed to the console.
The output stream is closed using the close() method.
getSHA(String input):

A MessageDigest object is obtained for the SHA-256 algorithm using the MessageDigest.getInstance() method.
The SHA-256 hash of the input string is computed using the digest() method of the MessageDigest object.
The resulting hash is returned as an array of bytes.

Comparison of Own Algorithm with Generic:

cannot be used to send and unencrypt files on another machine if it is not the original program. Can be used over multiple machines to decrypt and encrypt files.

Both programs use a common encryption method called DES. However most expert-level and generic programmes have additional security features in the form of threat detection and other risk controllers where ours does not.

Other more Expert level programmes would most likely include more error handling and input validation in order to make the program more robust and secure.

More common Encryption algorithms would have greater detail in the user interface with more available functions and better design whereas ours is very simple and has limited functionality.

More expert-level systems are also very likely to have more secure storage for encrypted files and will include features such as hashing algorithms and key management systems in place where ours does not

References:

1. *Java Code for DES - Javatpoint*. 2021. *www.javatpoint.com*. https://www.javatpoint.com/java-code-for-des Date of access: 15 May 2023.
2. *The DES Algorithm Illustrated*. 2023. *Tu-berlin.de*. https://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm Date of access: 15 May 2023.
3. Jhonny098. 2022. *Encryption with DES in Java*. *Stack Overflow*. https://stackoverflow.com/questions/72351900/encryption-with-des-in-java Date of access: 15 May 2023.
4. S-Logix. 2022. *Source code for encrypt and decrypt the using DES in Java | S-Logix*. *Slogix.in*. https://slogix.in/source-code/java-samples/how-to-encrypt-and-decrypt-data-using-des-in-java/ Date of access: 15 May 2023.