FC Venter 42468000
Carla Jansen 41799305
Johan Coetzee 40887340
Kayla Botha 34250174
Wimpie Botha 41153324
Group 42

FILE ENCRYPTION AND DECRYPTION TOOL DOCUMENTATION

Contents

Introduction

This Python-based tool provides an interface for file encryption and decryption using two different methods: Advanced Encryption Standard (AES) and a custom method that uses a combination of XOR cipher and Caesar cipher (Stallings, 2016; Boneh, 2015). The user interface is implemented using the Tkinter library.

Custom Encryption Algorithm

The custom encryption algorithm is a combination of two simple, classical encryption techniques: XOR cipher and the Caesar cipher.

XOR Cipher

The XOR cipher is a simple encryption algorithm used for encrypting plaintext into ciphertext and vice versa. It operates under the principles of the XOR logic gate where each bit of the plaintext is XORed with a bit from the key.

Despite its simplicity, the XOR cipher holds a crucial property: the operation is reversible. This means if we XOR the plaintext with a key, we get the ciphertext, and if we XOR the ciphertext with the same key, we get back the plaintext. This property makes the XOR cipher a valuable tool for encryption and decryption.

However, it is worth noting that the XOR cipher on its own is not very secure. If the key is shorter than the plaintext and is repeated, patterns can start to emerge, making it susceptible to frequency analysis attacks. Also, if the key is known or can be guessed, the encryption can be easily broken.

Caesar Cipher

The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each byte of the plaintext is ‘shifted’ a certain number of places down or up the alphabet. In the context of this script, each byte in the input is shifted along some number of places; the amount of shift depends on the byte from the key.

While the Caesar cipher was considered secure in its time, it is now extremely easy to break due to the limited number of possible keys, and it doesn’t hold up to frequency analysis.

Custom Algorithm Composition

The custom algorithm combines the XOR and Caesar ciphers. When encrypting, it applies the XOR cipher first, followed by the Caesar cipher. For decryption, it reverses the order: first the Caesar cipher (in reverse), then the XOR cipher. (Schneier, 1996).

This combination provides a slightly more secure method of encryption than using either the XOR or Caesar ciphers alone. However, it’s still far less secure than modern encryption methods due to the susceptibility of both ciphers to frequency analysis and known-plaintext attacks.

AES Encryption

AES, or Advanced Encryption Standard, is a symmetric encryption algorithm widely recognized as the most secure encryption standard and used globally to protect sensitive data.

AES is a symmetric key encryption method that is widely used and considered to be very secure. It has a block size of 128 bits and key sizes of 128, 192, or 256 bits (Daemen & Rijmen, 1999). AES operates on bytes and uses substitution-permutation networks, where each byte is replaced with another according to a lookup table (NIST, 2001). It is fast in both software and hardware, is relatively easy to implement, and requires little memory (Paar & Pelzl, 2010). This complexity makes AES resistant to all known practical attacks when used properly.

In this script, if AES is selected, the AES encryption from the imported AES class is applied to encrypt and decrypt files. The implementation assumes that the AES class is defined in a separate file named "AES.py" in the same directory.

Key Hashing

Regardless of the selected encryption method, the provided password is first hashed using the SHA-256 algorithm. SHA-256 belongs to the SHA-2 family of cryptographic hash functions and generates an almost-unique 256-bit (32-byte) signature for a text.

The hashed key is then truncated to the first 16 bytes (128 bits) to make it compatible with the key size requirements of the AES standard and the custom encryption. This provides an additional layer of security as the key used for encryption and decryption is not the plaintext password entered by the user, but a hashed version (Stallings, 2016).

File Handling

Upon successful encryption, the original file is replaced with its encrypted version, bearing the extension '.enc'. Similarly, after successful decryption, the '.enc' file is replaced with the decrypted file. This means the tool does not keep a copy of the original file after encryption or the encrypted file after decryption.

User Interface

The user interface for this tool is straightforward. It consists of fields for the user to input the file path and the password. There are also radio buttons for the user to select whether they want to encrypt or decrypt, and to choose between the custom encryption algorithm and AES.

There's a 'Browse' button to open a file dialog for easier file selection, and a 'Process' button to start the encryption or decryption operation.

After the operation completes, the tool displays a success message.

Comparison to Other Encryption Algorithms

While the custom encryption algorithm is simpler and easier to understand, it’s not as secure as modern encryption methods. The XOR and Caesar ciphers, while useful for learning and understanding basic cryptography concepts, are not robust against most types of cryptographic attacks.

On the other hand, AES is a widely accepted standard for encryption and is used globally for all types of sensitive data. It’s a symmetric block cipher that operates on a fixed block size (128 bits in this script) and provides excellent security. Its key size can be 128, 192, or 256 bits, and it’s resistant to all known practical attacks when the key is not known.

Comparing the two methods, AES is generally more secure due to its wide acceptance, rigorous testing, and use of more complex operations (Daemen & Rijmen, 1999). The custom method, while less secure than AES, provides a simple and easy-to-understand example of how encryption can work and may be sufficient for less sensitive data (Schneier, 1996).

It’s important to note that the use of SHA-256 for hashing the key adds an extra layer of security to both methods. By ensuring that the encryption and decryption keys are not simple text passwords, this reduces the possibility of brute-force attacks.

In conclusion, for actual secure data protection, you should use AES or another advanced encryption standard. The custom encryption method could be used for learning purposes or for situations where high security is not a critical concern.

References

Daemen, J., & Rijmen, V. 1999. AES Proposal: Rijndael. National Institute of Standards and Technology (NIST). http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

Paar, C., & Pelzl, J. 2010. Understanding Cryptography: A Textbook for Students and Practitioners. Springer-Verlag. ISBN: 978-3-642-04100-6.

Boneh, D. 2015. Cryptography I. Coursera. Stanford University. https://www.coursera.org/learn/crypto

National Institute of Standards and Technology (NIST). 2001. Announcing the ADVANCED ENCRYPTION STANDARD (AES). NIST FIPS PUB 197. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

Schneier, B. 1996. Applied Cryptography: Protocols, Algorithms and Source Code in C. Wiley. ISBN: 978-0471117094.

Khan Academy. 2021. Journey into cryptography. Khan Academy. https://www.khanacademy.org/computing/computer-science/cryptography

Stallings, W. 2016. Cryptography and Network Security: Principles and Practice. 7th ed. Pearson. ISBN: 978-0134444284.

Elminaam, D.A., Abdul-Kader, H.M., & Hadhoud, M.M. 2008. Evaluating The Performance of Symmetric Encryption Algorithms. International Journal of Network Security, 10(3): 213–219. http://ijns.jalaxy.com.tw/contents/ijns-v10-n3/ijns-2009-v10-n3-p213-219.pdf