Project Description: Secure File Encryption/Decryption CLI

Goal: Build a simple command-line tool that lets you securely lock (encrypt) and unlock (decrypt) files using a secret key. It keeps file data safe, even during unexpected crashes or power loss.

Prize 42 USD: For correct solutions, it doesn't need to be feature-complete; it just needs to be on the right track and have most of the functionality implemented.

What It Does:

1. Encrypt/Decrypt Files:
    - Protect the contents of a file and its name.
    - Keep the original filename private so the encrypted file doesn't reveal how long the filename is.
2. Crash Protection:
    - This technique uses reliable Write-Ahead Logging (WAL) to ensure that files will not be corrupted if the process stops unexpectedly.
    - Use WAL in an intelligent way for encrypting in place.
    - When encrypting in place, I order not to leave the file in an inconsistent state. If the process is terminated while performing the operation, when you run again, resume the operation. For example, when encrypting, if it fails in the middle, the file is corrupted. To avoid this, when starting again, we continue the encryption.

      Try to resolve this flow when encrypting inline

      - you read 128k

      - you encrypt, and it's 132k

      - you write 129b

      - you overwrite 1b from block2, which is decrypted

      - process dies

      - you run again

      - you cannot recover that byte

      => broken file, unrecoverable

3. Show File Info:

- Without decrypting, it lets users see details about the encrypted file (such as its name, size, and encryption method).

How It Works:

1. Encryption Algorithms:
   - Choose between two secure methods:
     - AES-GCM (a type of AES encryption)
     - ChaCha20-Poly1305 (another widely used encryption method).
       Both are solid options for keeping files safe.
2. CLI Commands:
   - `encrypt`: Lock a file (turn it into an unreadable, encrypted version).
   - `decrypt`: Unlock a file (restore it to its original readable form).
   - `show`: View information about an encrypted file without unlocking it.
3. Command Options:
   - Choose the algorithm: Use—`-alg` (or—`a`) to select an encryption method, such as `AES-GCM-128, AES-GCM-196, AES-GCM-256,` or `ChaCha20Poly1305`.
   - Input and output files:
     - Specify the file to process with `--in` (or `-i`).
     - Optionally, use `--out` (or `-o`) to define where the result should go. If skipped, the input file is replaced directly.
4. Secure Passphrase:
   - The tool will ask you for a passphrase (like a password) when it starts.
   - The passphrase isn't stored anywhere, and it's converted into a strong key to lock/unlock the file.

Bonus Features:

1. Use age tool to encrypt encryption key.
2. Use OpenPGP to encrypt the encryption key.

For these, you'll generate a random key and encrypt it with age or OpenPGP to send to User2 and pass that key as a parameter to CLI. You'll then send the key and encrypted file to User2, who will pass them to the app to decrypt.

How to Submit:

1. Create a public GitHub repository with your project code.

2. Share the repository link via email to radumarias@gmail.com.
3. **Deadline: 25th November 19:00 GMT+2**

For Rust, crates you can use

- https://crates.io/crates/ring
- https://crates.io/crates/chacha20poly1305
- https://crates.io/crates/aes-gcm
- https://crates.io/crates/rand
- https://crates.io/crates/rand_chacha
- https://crates.io/crates/argon2
- https://crates.io/crates/blake3
- https://crates.io/crates/clap
- https://crates.io/crates/thiserror
- https://crates.io/crates/tracing
- https://crates.io/crates/rpassword
- https://crates.io/crates/ctrlc
- https://crates.io/crates/okaywal
- https://crates.io/crates/shush-rs
- https://crates.io/crates/rage