SecureCLI: A Python-Based Command-Line Security Toolkit

SecureCLI is a robust, open-source command-line interface (CLI) tool developed in Python, designed to offer a comprehensive suite of cryptography and security functionalities. It provides essential features for developers, security enthusiasts, and system administrators to perform various security-related tasks directly from the terminal. This document outlines SecureCLI's capabilities, usage, and key security considerations, serving as a guide for its effective and responsible deployment.

Built with Python 3.9+, SecureCLI ensures cross-platform compatibility across Linux, macOS, and Windows environments. The project is openly accessible under the MIT License, encouraging community contributions and transparency in its development. For more detailed information and to contribute, please visit the official https://github.com/YANTRA-EKA/securecli

Key Features and Functionality

Caesar Cipher Encryption/Decryption

SecureCLI includes a basic Caesar cipher implementation for educational purposes or simple obfuscation. It allows encryption and decryption of messages using a specified shift value (1-25), supporting uppercase letters, lowercase letters, and numbers.

- **Encryption:** `securecli caesar --encrypt "message" --key 3`
- **Decryption:** `securecli caesar --decrypt "phvvdjh" --key 3`

Base64 Encode/Decode

This feature facilitates the conversion of binary data to a text-based Base64 format and vice-versa, primarily used for transmitting data over mediums that only support text. It leverages Python's efficient built-in base64 library.

- **Encode:** securecli base64 --encode "data"
- **Decode:** securecli base64 --decode “ZGF0YQ=="

SHA-256 Hash Generation

SecureCLI provides the capability to generate SHA-256 hashes of input strings. SHA-256 is a cryptographic hash function widely used for data integrity verification, digital signatures, and more. The tool utilises Python's robust hashlib library.

- **Usage:** securecli sha256 --string "input"
- **Example Output:** e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4

Password Strength Checker

Assessing password strength is crucial for security. SecureCLI's password strength checker evaluates a given password against various criteria, including length, character types (uppercase, lowercase, numbers, special characters), and common patterns. It returns a score from 0 to 100, categorised as Weak (0-30), Medium (31-60), or Strong (61-100).

- **Usage:** securecli check --password "P@$$w0rd123"

Advanced Security Features

Strong Password Generator

To combat weak passwords, SecureCLI includes a strong password generator. This feature creates cryptographically secure passwords by using Python's secrets module, ensuring high-entropy random number generation. Passwords can be customised by length (8-64 characters) and can optionally include special characters and numbers.

```
securecli generate --length 16
```
(generates a 16-character password)

File Integrity Verifier

Ensuring the integrity of files is vital to detect unauthorised modifications. SecureCLI allows users to verify file integrity using SHA-256 hashing. It calculates the hash of a specified file and compares it against an expected hash, reporting whether the file remains untampered.

```
securecli verify --file "path/to/file" --hash "expected_hash"
```

XOR-based Encryption/Decryption

For simple, symmetric encryption, SecureCLI offers XOR-based encryption and decryption. This method performs a bitwise XOR operation between the message and a user-provided key. While easy to implement, it's important to note that XOR encryption is vulnerable to known-plaintext attacks and should be used with extreme caution, primarily for non-sensitive data or as a learning exercise.

```
securecli xor --encrypt "message" --key “secret"
```

Usage Examples and Command Syntax

Caesar Cipher	To view options: `securecli caesar --help` • **Encrypt:** `securecli caesar --encrypt "my secret message" --key 5` • **Decrypt:** `securecli caesar --decrypt "rd xjwkxy rjxxflj" --key 5`
Base64	• **Encode:** `securecli base64 --encode "hello world"` • **Decode:** `securecli base64 --decode "aGVsbG8gd29ybGQ="`
SHA256	• **Generate Hash:** `securecli sha256 --string "test string"`
Password Strength Checker	• **Check Weak:** `securecli check --password "weak"` • **Check Strong:** `securecli check --password "VeryStrongP@$$w0rd"`
Strong Password Generator	• **Generate:** `securecli generate --length 20 --special --number`
File Integrity Verifier	• **Verify:** securecli verify --file "document.txt" --hash "e59..."

Secure Hashing with Python's hashlib

SecureCLI extends its hashing capabilities by demonstrating secure hashing techniques utilising Python's versatile hashlib library. This allows users to generate cryptographic hashes using various algorithms, including SHA-256, SHA-512, and other SHA-3 variants. Cryptographic hashes are one-way functions, meaning it is computationally infeasible to reverse them, and they offer collision resistance, making them ideal for verifying data integrity and digital signatures.**Command Example:** securecli hash --algorithm sha256 --input “data"

Salting Passwords

A critical practice in secure password storage is salting. SecureCLI illustrates this by explaining how to generate a unique, random salt for each password before hashing. This salt is then concatenated with the password, and the combined string is hashed. The resulting hash and the salt are stored, preventing pre-computed rainbow table attacks that exploit common password hashes. This method significantly enhances the security of stored credentials by making each hashed password unique, even if two users have the same original password.

Password Management

SecureCLI integrates with the widely used pass command-line password manager, enabling users to securely store and retrieve their passwords. This integration leverages pass's robust GnuPG (GPG) encryption, ensuring that sensitive credentials are encrypted at rest. For this feature to work, the pass CLI tool must be installed and properly configured on the system.

- **Store Password:** securecli pass --store "account_name" --password "new_password"
- **Retrieve Password:** (Demonstrated through pass's native commands, as SecureCLI acts as an interface)

Important Security Considerations

SecureCLI is a tool designed to assist with security tasks, but it does not guarantee absolute security. Users are solely responsible for understanding the cryptographic principles, limitations, and potential vulnerabilities associated with each feature. Always use the tool responsibly and review its underlying code, especially for custom implementations or sensitive applications.

Security Considerations

When utilising SecureCLI or any security tool, adherence to best practices is paramount to mitigate risks and ensure robust protection. Users must be proactive in their security posture beyond just running commands.

Key Management

Properly storing and handling encryption keys is crucial. Never hardcode keys in scripts, expose them in public repositories, or store them in unsecured locations. Consider using environment variables, dedicated key management systems, or secure vaults.

Input Validation

Always sanitise and validate all user inputs. Failing to do so can lead to various injection attacks, buffer overflows, or unexpected tool behaviour that could compromise security.

Side-Channel Attacks

Be aware of potential side-channel attacks, such as timing attacks or power analysis, which could reveal sensitive information based on the operational characteristics of cryptographic functions. While SecureCLI is designed to be robust, the environment it runs in can introduce vulnerabilities.

Dependencies and Updates

Regularly update SecureCLI and all its underlying Python libraries and system dependencies. Software vulnerabilities are frequently discovered and patched, and keeping components up-to-date is a fundamental security practice.

SecureCLI is a powerful utility, but it is ultimately a tool. Its effectiveness depends on the user's understanding and implementation of broader security principles. It does not replace comprehensive security audits or expert consultation.

Conclusion

SecureCLI offers a valuable collection of Python-based command-line utilities for common cryptography and security tasks, ranging from basic encryption and hashing to advanced password management integration. Its cross-platform compatibility, open-source nature, and focus on practical security features make it an indispensable tool for developers, security enthusiasts, and system administrators alike.

As an open-source project, SecureCLI thrives on community engagement. Contributions, bug reports, feature requests, and general feedback are highly encouraged and welcome. This collaborative approach ensures continuous improvement and adaptation to evolving security landscapes.

Please visit the https://github.com/YANTRA-EKA/securecli to participate.

The SecureCLI development team is committed to continuous enhancement, with future updates planned to introduce new features, improve existing functionalities, and address emerging security challenges. We urge all users to employ SecureCLI responsibly and ethically, understanding both its powerful capabilities and inherent limitations.

SecureCLI is distributed under the MIT License, promoting freedom of use, modification, and distribution. For any inquiries or further information, please refer to the project's official documentation or contact the maintainers via the GitHub platform.