Noiseblind

Background

Introduction

Noiseblind is a noise-based file obfuscation algorithm designed to conceal the recognizable structure and type of digital files by turning them into a designated Noiseblind file format. While it **does** share some similarities with encryption, Noiseblind is an **obfuscation tool** meant to help me learn more about data security, optimization, and general low-level programming.

Noiseblind applies layers of deterministic noise to guide reversible transformations, each affecting the data in different ways. When stacked, these transformations make the file unrecognizable but completely reversible when the correct password is provided.

To make the system accessible, Noiseblind features a lightweight interface for everyday use along with a command-line interface for both scripting and advanced workflows. All transformations require a user-provided password for both obfuscation and restoration, keeping the process simple and secure, while still easy to access. Noiseblind is designed to be quick and efficient, utilizing computer resources effectively without slowing down other applications. ***Noiseblind is not intended as a professional-grade encryption tool***.

Description

Noiseblind is a multi-threaded noise-based obfuscation application that turns user files and passwords into scrambled NBQ files through layered deterministic noise. To achieve this, Noiseblind relies mainly on LibSodium and FastNoise2 for the password hashing and the noise generation respectively. Noiseblind also uses CLI11 for its CLI parsing, Dear ImGui for its GUI, and nlohman/json for handling a config file.

Install

Noiseblind is generally simple to install but as mentioned does have a couple dependencies. As CLI11 and nlohman/json are header only libraries they are not required to install Noiseblind. Similarly through CMake, ImGui and the backend (GLFW and OpenGL) will be automatically installed during compilation, and FastNoise2 is included as a submodule for easy installation. Libsodium is now included as well through shared library files, all dependencies can be found in the external folder.

Usage

There are two main ways to use Noiseblind: CLI and GUI. While both offer the basic Obfuscation and Restoration pipelines, the CLI has more robust functionality with additional options. Launching the GUI is as simple as clicking on the `Noiseblind.exe` or typing `./Noiseblind`, from there the GUI window will launch and all the buttons are clearly labeled.

Note for Linux Users

While the Linux GUI does offer file browser support, it expects either Zenity or Kdialog to be installed.
If neither are installed you'll have to *type* the paths.

CLI

The CLI has more robust functionality and includes the following options:

- Obfuscate
- Restore
- Metadata
- Config

There are a lot of flags but the basics can be summed up as:
`./Noiseblind obfuscate -i input.file -o output.directory -p password`
Try `./Noiseblind --help-all` to see all available options

Optimization Notes

Noiseblind utilizes a custom SIMD engine, **MercurySIMD**, to accelerate obfuscation operations. Mercury is designed specifically for the **AVX2** instruction set architecture. It utilizes the 256 bit YMM caches on the CPU to process 32 bytes per instruction.

- **Hardware Support**: If your CPU supports AVX2 (e.g., AMD Ryzen 3000+, Intel Haswell+), Noiseblind will automatically utilize Mercury for ~9x faster performance.
- **Fallback**: On systems without AVX2, Noiseblind will run in standard scalar mode. For ensuring maximum compatibility on older machines, you can manually disable SIMD by setting #define ENABLE_SIMD false in Layers.hpp before compiling.
- **Compatibility Warning**: Files encrypted using the SIMD engine cannot be restored on systems that lack AVX2 support. However, standard scalar files are universal and can be restored on any system, regardless of hardware acceleration.

HDD vs SSD

Noiseblind feature a dual-mode I/O architecture for SSDs and HDDs. While SSDs uitlize high concurrency asynchronous I/O and a work-stealing method, HDDs engage a Phase-Locked Barrier System to prevent mechanical head thrasing.

- **HDD MODE**: Automatically engages when a rotational drive is detected. It uses a synchronized buffer to ensure large sequential I/O bursts.

- **Performance Tip**: For mechanical drives, a 1 MiB chunk size is recommended to maximize sustained throughput (approaching the ~120 MB/s hardware limit)

Performance

Different Chunk Sizes effect on performance:

Setup	Chunk Size	Runtime	Est. Throughput
Initial (8 Kib)	8,192 B	~49s	~75 MB/s
Optimized (1 Mib)	1,048,576 B	~36s	~100 MB/s
HDD Limit	N/A	N/A	~120 MB/s

Layer impact on processing speed for different file sizes:

Note: Some records were missed for fastest times

File	Size	Layers	Time(Avg. of 6)	Fastest(Recorded)
icon.png	1.02MB	1	116ms	116ms
icon.png	1.02MB	2	132ms	120ms
icon.png	1.02MB	5	136ms	128ms
icon.png	1.02MB	10	151ms	143ms
icon.png	1.02MB	100	498ms	467ms
test.zip	38.5MB	1	452ms	379ms
test.zip	38.5MB	2	416ms	399ms
test.zip	38.5MB	5	505ms	467ms
test.zip	38.5MB	10	593ms	538ms
test.zip	38.5MB	100	2519ms	2268ms
Clips.zip	3.48GB	1	29094ms	26138ms
Clips.zip	3.48GB	2	28447ms	29061ms
Clips.zip	3.48GB	5	35792ms	33966ms
Clips.zip	3.48GB	10	44895ms	37795ms
Clips.zip	3.48GB	100	200363ms	178306ms

Analyzing The Results

Time(ms) Ratio per 10x Increase

Small	Medium	Large
151 / 116 = 1.3	593 / 452 = 1.3	44895 / 29094 = 1.5
498 / 151 = 3.3	2519 / 593 = 4.25	200363 / 44895 = 4.46

Throughput Analysis

Effective Throughput: (100 Layers)

File Size	Computational Throughput	User Throughput
Small:	~204.8 MB/s	~2 MB/s
Meduim:	~1528.4 MB/s	~15.3 MB/s
Large:	~1778.3 MB/s	~17.8 MB/s

Computational throuput is calculated from (file size * layers) / time

Known Issues

- Hardware Detection: In virtual environments (WSL/VMs), the engine may default to HDD(Safe Mode) if the host hardware reports a rotational flag.

Disclaimer

Noiseblind is a performance-based obfuscation engine and is not intended for cryptographic use-cases that require formal proofs.

License