ChaosCrypto WP2 (CLI)

Deterministic ChaosCrypto CLI (WP2 MVP). This README is Windows-first and includes Linux/macOS notes. Quick copy-paste setup is also in `QUICKSTART.md`.

Project Overview

This project provides a CLI-driven system for deterministic encryption based on a shared, reproducible noise field and a chaos-derived keystream. At a high level, it turns a user token, profile parameters, and a coordinate in a noise field into a repeatable keystream that can be used to encrypt and decrypt byte data. The main idea is that encryption results are reproducible and verifiable: the same inputs always yield the same keystream and ciphertext, enabling transparent benchmarking and academic evaluation. The project addresses the need to study chaos-based cryptographic workflows in a controlled, repeatable way without hidden randomness. It is intended to be used entirely through the CLI, where profiles are initialized, keystreams are generated, and data is encrypted or decrypted deterministically, making outputs comparable across runs and systems.

Architecture and Key Components

The architecture is a layered CLI application that orchestrates deterministic key generation and encryption by combining noise-field sampling with a Lorenz-based chaos system. The CLI entry point in `src/chaoscrypto/cli/app.py` parses commands and delegates to the core pipeline, while the noise field is generated by memory modules such as `src/chaoscrypto/core/memory/opensimplex.py` and `src/chaoscrypto/core/memory/perlin.py`. The chaotic system is implemented in `src/chaoscrypto/core/chaos/lorenz.py`, and the resulting keystream is applied via the cryptographic XOR component in `src/chaoscrypto/core/crypto/xor.py`. A shared noise field combined with token-based determinism ensures that the same profile and token yield synchronized, reproducible outputs, which is why this architecture favors transparency and repeatability over opaque randomness.

Configuration and profile persistence are handled in `src/chaoscrypto/io/profiles.py`, while input/output formats for ciphertexts and reports are organized in `src/chaoscrypto/io/formats.py`. The coordination between components is centralized in the pipeline layer (see `src/chaoscrypto/orchestrator/pipeline.py`), which connects CLI actions to noise sampling, seed strategies, chaotic integration, and output serialization. This structure makes the main logic easy to trace from the CLI down to deterministic keystream generation and file I/O, and it supports synchronized benchmarking workflows where identical inputs must always reproduce identical results.

Quickstart (Windows)

Prereqs

- Windows 10/11
- Python 3.10+ (`py -3.11 --version` recommended)
- Git for Windows
- PowerShell (default on Windows)
- Optional: WSL (Ubuntu) as a fallback

1) Clone the repo

PowerShell:

```
git clone https://github.com/th-web-dev/cli_chaoscrypto
cd cli_chaoscrypto
```

Bash (Linux/macOS/WSL):

```
git clone https://github.com/th-web-dev/cli_chaoscrypto
cd cli_chaoscrypto
```

2) Create a virtual environment

PowerShell:

```
py -3.11 -m venv .venv
```

Bash (Linux/macOS/WSL):

```
python3 -m venv .venv
```

3) Install dependencies

PowerShell:

```
.\.venv\Scripts\python.exe -m pip install --upgrade pip
.\.venv\Scripts\python.exe -m pip install -r requirements.txt
.\.venv\Scripts\python.exe -m pip install -e .
```

Bash (Linux/macOS/WSL):

```
.venv/bin/python -m pip install --upgrade pip
.venv/bin/python -m pip install -r requirements.txt
.venv/bin/python -m pip install -e .
```

4) Smoke test the CLI

PowerShell:

```
.\.venv\Scripts\python.exe -m chaoscrypto.cli.app --help
```

Bash (Linux/macOS/WSL):

```
.venv/bin/python -m chaoscrypto.cli.app --help
```

If you see the CLI help, you are ready.

Troubleshooting (Windows)

- Execution policy blocks scripts: use `Set-ExecutionPolicy -Scope`

`Process -ExecutionPolicy Bypass` for the current PowerShell session.
- Python not found: install from https://www.python.org/downloads and re-open PowerShell; use `py -3.11` if multiple versions are installed.
- Long paths: enable Windows long paths (Group Policy/Registry) or run `git config --global core.longpaths true`.
- Pip issues: upgrade pip and wheel in the venv with `python -m pip install --upgrade pip wheel`.

Two ways to run

A) Native Windows (recommended if it works)

Use the Windows venv Python:

```
.\.venv\Scripts\python.exe -m chaoscrypto.cli.app --help
```

Profiles are stored in: - `%USERPROFILE%\.chaoscrypto\wp2\<profile>\profile.json`

B) WSL (Ubuntu) fallback

Open WSL in the repo folder (example path shown):

```
cd /mnt/c/Users/<you>/chaoscrypto-wp2
```

Use the WSL venv Python:

```
.venv/bin/python -m chaoscrypto.cli.app --help
```

Profiles are stored in: - `~/.chaoscrypto/wp2/<profile>/profile.json`

Install / Verify

Install (editable) and verify:

PowerShell:

```
.\.venv\Scripts\python.exe -m pip install -r requirements.txt
.\.venv\Scripts\python.exe -m pip install -e .
.\.venv\Scripts\python.exe -m chaoscrypto.cli.app --help
```

Bash (Linux/macOS/WSL):

```
.venv/bin/python -m pip install -r requirements.txt
.venv/bin/python -m pip install -e .
.venv/bin/python -m chaoscrypto.cli.app --help
```

Minimal demo workflow (5 minutes)

Step 1: init profile

PowerShell:

```
.\.venv\Scripts\python.exe -m chaoscrypto.cli.app init --profile
```

```
alice --token "secret" --memory-type opensimplex --size 128 --scale
0.1
```

Bash (Linux/macOS/WSL):

```
        .venv/bin/python -m chaoscrypto.cli.app init --profile alice --token
"secret" --memory-type opensimplex --size 128 --scale 0.1
```

Expected: - A profile folder is created under your home directory. - `profile.json` is written with memory parameters and token fingerprint (no raw token stored).

Step 2: generate keystream

PowerShell:

```
        .\.venv\Scripts\python.exe -m chaoscrypto.cli.app keystream --
profile alice --token "secret" --coord 12,34 --nbytes 32
```

Bash (Linux/macOS/WSL):

```
        .venv/bin/python -m chaoscrypto.cli.app keystream --profile alice --
token "secret" --coord 12,34 --nbytes 32
```

Expected: - A short keystream preview on stdout and a SHA-256 hash.

Step 3: encrypt and decrypt a small message

PowerShell:

```
        "hello from ChaosCrypto" | Set-Content -Encoding utf8 msg.txt
        .\.venv\Scripts\python.exe -m chaoscrypto.cli.app encrypt --profile
alice --token "secret" --coord 12,34 --in msg.txt --out enc.json
        .\.venv\Scripts\python.exe -m chaoscrypto.cli.app decrypt --profile
alice --token "secret" --in enc.json --out dec.txt
```

Bash (Linux/macOS/WSL):

```
        echo "hello from ChaosCrypto" > msg.txt
        .venv/bin/python -m chaoscrypto.cli.app encrypt --profile alice --
token "secret" --coord 12,34 --in msg.txt --out enc.json
        .venv/bin/python -m chaoscrypto.cli.app decrypt --profile alice --
token "secret" --in enc.json --out dec.txt
```

Expected: - `enc.json` created in the working directory. - `dec.txt` matches `msg.txt`.

Evaluation workflow (BA1 / WP2 pipeline)

Use the provided configs (examples) and write outputs to `out/ba1`.

PowerShell:

```
        .\.venv\Scripts\python.exe -m chaoscrypto.cli.app benchmark --config
examples/ba1_benchmark.yaml --out out/ba1/bench/results.csv --out-
json out/ba1/bench/results.json
        .\.venv\Scripts\python.exe -m chaoscrypto.cli.app analyze --config
examples/ba1_analyze.yaml --out out/ba1/analyze/analysis.csv --out-
json out/ba1/analyze/analysis.json
        .\.venv\Scripts\python.exe -m chaoscrypto.cli.app report --bench-csv
```

```
out/ba1/bench/results.csv --analysis-csv
out/ba1/analyze/analysis.csv --out out/ba1/report/report.md --plots-
dir out/ba1/report/plots --no-timestamp
```

Bash (Linux/macOS/WSL):

```
        .venv/bin/python -m chaoscrypto.cli.app benchmark --config
examples/ba1_benchmark.yaml --out out/ba1/bench/results.csv --out-
json out/ba1/bench/results.json
        .venv/bin/python -m chaoscrypto.cli.app analyze --config
examples/ba1_analyze.yaml --out out/ba1/analyze/analysis.csv --out-
json out/ba1/analyze/analysis.json
        .venv/bin/python -m chaoscrypto.cli.app report --bench-csv
out/ba1/bench/results.csv --analysis-csv
out/ba1/analyze/analysis.csv --out out/ba1/report/report.md --plots-
dir out/ba1/report/plots --no-timestamp
```

Optional scripts: - PowerShell: `scripts/run_ba1.ps1` (run with `powershell -ExecutionPolicy Bypass -File scripts\\run_ba1.ps1`) - Bash: `scripts/run_ba1.sh`

What `out/ba1` contains: - `run_meta.txt`: timestamp, Python version, system info, `pip freeze` - `bench/results.csv` + `bench/results.json` - `analyze/analysis.csv` + `analyze/analysis.json` - `report/report.md` and `report/plots/` (PNG)

Clean (safe warning: deletes the entire folder):

PowerShell:

```
        Remove-Item -Recurse -Force .\out\ba1
```

Bash (Linux/macOS/WSL):

```
        rm -rf out/ba1
```

Config explanation (brief)

YAML keys you will most often adjust:

- `profile`: profile name to use (must exist)
- `coord`: coordinate used to derive the Lorenz seed (x,y)
- `nbytes`: keystream length
- `dt`, `warmup`, `quant_k`: Lorenz integration and sampling
- `size`, `scale`: memory field parameters (must match profile)
- `seed_strategy`: seed selection method
- `memory_type`: `opensimplex` or `perlin`
- `repeats`: benchmark repeats per variant

Security note: - The raw token is never stored; only fingerprints/hashes are recorded in profiles and outputs.

Command cheat sheet

Command	One-liner
init	`python -m chaoscrypto.cli.app init --profile alice --token "secret" --size 128 --scale 0.1`

keystream	`python -m chaoscrypto.cli.app keystream --profile alice --token "secret" --coord 12,34 --nbytes 1024`
encrypt	`python -m chaoscrypto.cli.app encrypt --profile alice --token "secret" --coord 12,34 --in msg.txt --out enc.json`
decrypt	`python -m chaoscrypto.cli.app decrypt --profile alice --token "secret" --in enc.json --out dec.txt`
benchmark	`python -m chaoscrypto.cli.app benchmark --config examples/bench.yaml --out results.csv --out-json results.json`
analyze	`python -m chaoscrypto.cli.app analyze --config examples/analyze.yaml --out analysis.csv --out-json analysis.json`
report	`python -m chaoscrypto.cli.app report --bench-csv results.csv --analysis-csv analysis.csv --out report.md --plots-dir plots`

Notes

- Deterministic: same token + same parameters → identical keystreams and outputs.
- Profiles are stored under your user home directory (never in the repo).
- `--verbose` / `--debug` add logs; they do not change outputs.