BlankOBF EDU Toolkit – Extended User & Developer Documentation

This extended document explains how the toolkit is built, why the code is structured the way it is, and how to use every feature in the GUI (including file/folder browsing, overwrite safety, progress/cancel, and the compiler icon picker). It is written to be educational and beginner-friendly, while still describing the important technical details accurately.

Contents

1. Project goals and safety boundaries

BlankOBF EDU Toolkit is an educational toolkit for:
• Python obfuscation (readability reduction without changing behavior)
• Strong, standard encryption for files and folders
• Packaging Python apps for distribution (BAT/EXE convenience)

It is intentionally designed for safe, legal workflows: the tool encrypts for storage and later manual decryption.
It does not provide mechanisms to automatically execute encrypted content.

2. High-level architecture and folder structure

The project is organized into a few core packages, each responsible for one area:

• blankobf/gui.py
The GUI entry point. Builds tabs, wires buttons, runs long tasks in background threads, and writes logs.

• blankobf/engine.py + transforms/ + layers/
The obfuscation engine. It reads Python code, parses it, applies transformations, and writes the resulting code.

• blankobf/crypto/
Cryptography utilities:
- bundle.py: folder bundling format and layered AEAD encryption
- filecrypt.py: single-file encryption container with metadata and layered AEAD

• run_app.py
Simple launcher that imports the GUI main() and starts the app.

The code is intentionally split into modules so that the GUI stays " thin " : it should call functions in the engine/crypto modules rather than implementing everything inside the UI.

3. GUI design decisions

Why CustomTkinter?
• It ' s still Tkinter underneath (very easy to run), but provides a more modern feel (rounded frames, better widgets).
• It keeps dependencies light compared to PyQt/Kivy.

Why " tabs " ?
• Users can explore features without getting lost in nested windows.
• Each feature stays isolated: bugs or changes in one tab are less likely to break another.

Why background threads?
• Obfuscation and encryption can take time.
• Running heavy work on the main GUI thread freezes the window.
• The GUI starts a daemon thread for work, and uses `after(0, …)` to safely update progress bars and labels.

Why a log box?

• It gives you a “ black box recorder ” : every action writes a line so debugging is easier.
• If a user reports a problem, they can paste the log lines.

4. Obfuscation module

The obfuscation pipeline works like this:
1) Read the input .py as text (UTF-8).
2) Parse and transform it using AST transforms (safe source-to-source transformations).
3) Optionally apply additional “ layers ” when AST-only is turned off.
4) Write the result to the chosen output file.

Main UI inputs:
• Input .py – the source file you want to obfuscate.
• Output .py – where the transformed code should be saved.
• Report .json – optional metadata report for analysis/debugging.

Seed (optional integer):
• If empty → output may differ each run (randomized names etc.).
• If set → reproducible output. Example: seed=12345.

Recursion (integer >= 1):
• How many passes are applied.
• 1 is recommended for stability; 2 can increase “ mixing ” ; 3+ is mostly for experiments.

AST-only vs XOR lab (mutually exclusive):
• AST-only ON → only AST transforms; output stays “ normal Python code ” .
• XOR lab ON → experimental layer applied (slower). It requires AST-only OFF.
The GUI enforces this by disabling the other checkbox automatically.

Progress/cancel:
• The obfuscation engine periodically calls the GUI progress callback.
• Cancel sets a flag (stop event). The pipeline checks that flag and stops early.

5. Encryption module (single file)

The “ Encrypt file ” tab protects any file using strong, standard cryptography.
You can select multiple layers; they apply left → right in order.

Algorithms (layers):
• ChaCha20-Poly1305 (AEAD) – fast and safe on all CPUs
• AES-256-GCM (AEAD) – widely used, hardware accelerated on many systems
• AES-SIV – misuse-resistant mode (if supported by your cryptography package build)

Important: multiple layers are educational and provide defense-in-depth, but they also add complexity.
In practice, one strong AEAD mode is usually sufficient.

Password handling:
• The password is turned into a master key via scrypt (a memory-hard KDF).

• Per-layer subkeys are derived from the master key.

Output format:
• Default output is a container file (commonly .enc). It stores metadata (original filename, salt, layer list).
• You can decrypt later using the correct password.

Overwrite mode (optional):
• If you enable overwrite, the tool will:
1) require a confirmation checkbox,
2) create a .bak backup,
3) write the encrypted container bytes into the original filename.
• The file will NOT be a valid Python file anymore (it ’ s encrypted data). Use decrypt to restore.

6. Decryption module

There are two ways to decrypt:

A) Decrypt (save as)
• Choose an encrypted input file.
• Choose an output file path.
• Enter password.
• Click Decrypt.

B) Decrypt → temp + open
• Choose encrypted input + password.
• Click the temp button.
• The tool creates a temporary folder, decrypts into it using the original filename stored in the header, then opens the folder so you can immediately access the usable file.

This workflow is designed to keep encryption “ safe for storage ” while still being convenient.

7. Folder bundles (.bobf)

When to use bundles:
• You have a project folder with multiple files (scripts, configs, assets).
• You want one encrypted artifact instead of encrypting each file separately.

How it works:
• The tool walks the folder, records relative paths, and packs data into a container.
• The packed bytes are encrypted with the same layered AEAD approach.
• Decrypting restores the full folder structure into the output directory.

What is .bobf?
• A custom container extension used by the toolkit for “ BlankOBF Bundle File ” .
• It ’ s not ZIP; it ’ s a structured encrypted container with integrity protection.

8. Compiler module

The Compiler tab is a convenience tool for distribution.

Create .bat wrapper:
• Copies your selected .py into the output folder (same base name).
• Writes a .bat that launches it using your system Python interpreter.
• Useful for quick “ double-click run ” during development.

Build .exe (PyInstaller):
• Requires: `pip install pyinstaller`
• Uses your chosen entry script and builds a Windows executable into the output folder.
• Optionally adds an icon if you provide a .ico file.

Icon picker:
• Click Browse next to “ Icon (.ico) ” and select your .ico.
• If the path exists, the GUI passes `--icon your.ico` to PyInstaller.
• If you don ’ t select an icon, it builds normally without that flag.

9. Using Browse buttons (file/folder selection)

The Browse buttons exist to reduce typing errors and to ensure correct paths.

Browse (Input .py / Input file):
• Opens a file picker.
• You select the file you want to process.
• The selected path is inserted into the input field.

Browse (Output .py / Output file):
• Opens a “ Save As ” dialog.
• You choose where the result should be written.
• If you leave output empty in some tabs, the GUI may auto-generate a sensible default next to the input file.

Browse (Folder / Output folder):
• Opens a folder picker (directory selection).
• Used for bundle encryption (input folder) and bundle decryption (output folder).
• Used for compiler output folder.

Practical tip:
• If you keep everything in one project directory, it ’ s easiest to place outputs into a subfolder like
`dist/` or `out/` so you don ’ t overwrite source files by accident.

10. Troubleshooting checklist (Windows)

If the GUI does not start:
• Ensure CustomTkinter is installed: `pip install customtkinter`
• Run inside the project folder: `python run_app.py`
• If you installed with `pip install -e .[gui]`, re-check that you are using the same Python interpreter.

If PyInstaller build fails:

• Install it: `pip install pyinstaller`
• Some antivirus software can interfere with builds; try a clean folder.
• Check the log box for stderr output.

If encryption/decryption fails:
• Ensure you typed the exact same password.
• Ensure you selected the correct file type (the decrypt tab expects the container file you created).
• If overwrite was used, restore from the .bak if needed.

If obfuscation breaks a script:
• Try AST-only ON and recursion=1.
• Increase recursion only after confirming output is correct.
• Report generation can help diagnose transformations.

End of document.