Building Binaries (Advanced)

Status: Alpha documentation snapshot (2026-03-12).

This guide is for advanced users who want to build Atho binaries from source, including the falcon_cli helper, on Linux, macOS, and Windows. If you want the fastest path, use Docker (it builds falcon_cli inside a container with all dependencies preinstalled).

Cross-OS Binary Layout (required)

Runtime now checks platform-tagged Falcon binary folders first:
- Src/Falcon/Binaries/darwin-arm64/falcon_cli
- Src/Falcon/Binaries/darwin-x86_64/falcon_cli
- Src/Falcon/Binaries/linux-x86_64/falcon_cli
- Src/Falcon/Binaries/linux-arm64/falcon_cli
- Src/Falcon/Binaries/windows-x86_64/falcon_cli.exe

App Binary Build (GUI launcher binary)

You can now generate release binary artifacts with:

Optional modes:

Output goes to:
- releases/binaries/<timestamp>/...

Important:
- Native GUI binaries should be built on their target OS for best compatibility.
- Windows .exe should be built on Windows host (or dedicated CI runner).
- Script always creates a source snapshot fallback package via make_prealpha_package.sh.

You can install the current host build into the correct folder with:

Windows (PowerShell):

Optional explicit source path:

The script prints:

- binary_sha3_384
- version-bound digest
- exact const.py pin values to update (FALCONCLI_PINNED_VERSION_DIGESTS_BY_PLATFORM and FALCONCLI_EXPECTED_HASH_BY_PLATFORM)

Prerequisites by OS
Linux (Debian/Ubuntu)
- build-essential (gcc/g++/make)
- cmake (if you rebuild beyond the provided script)
- libssl-dev (for hashing/crypto)
- Python 3.11+ and pip install -r requirements.txt

macOS
- Xcode Command Line Tools: xcode-select --install
- Homebrew packages (if needed): brew install openssl cmake (adjust PATH/CPPFLAGS/LDFLAGS for OpenSSL if non-system)
- Python 3.11+ and pip install -r requirements.txt

Windows
- Recommended: WSL (Ubuntu) with the Linux prerequisites above.
- Native (more work): install Visual Studio Build Tools (C++ workload) or MSYS2/MinGW-w64; ensure gcc/clang and make are on PATH. Python 3.11+ with pip, and pip install -r requirements.txt.
- PowerShell/WSL tip: use WSL to avoid toolchain headaches; Docker Desktop can build for you (see below).

Building falcon_cli from source
Falcon sources live in Src/Falcon/Falcon/. The Dockerfile builds with gcc by default.

Linux / macOS

Notes:
- The script builds with -O3 -std=c99 -Wall -Wextra -DFALCON_FPNATIVE.
- Set FALCON_USE_AVX2=true on x86_64 with AVX2 support; keep false on ARM/older CPUs.
- After build, ensure falcon_cli is executable: chmod +x falcon_cli.

Windows (WSL recommended)
- **WSL path:** follow Linux steps above.
- **Native:** open MSYS2/MinGW shell or Developer Command Prompt, and compile with your toolchain:
- MinGW example: gcc -O3 -std=c99 -Wall -Wextra -DFALCON_FPNATIVE -o falcon_cli cli.c falcon.c fft.c fpr.c keygen.c rng.c shake.c sign.c vrfy.c common.c codec.c
- Adjust flags for AVX2: add -mavx2 -DFALCON_AVX2 if supported.
- Ensure falcon_cli.exe ends up in Src/Falcon/Binaries/windows-x86_64/ (or set ATHO_FALCONCLI_BIN).

For GUI .exe packaging on Windows host:

This produces:
- dist\AthoAlpha-windows.exe
- a timestamped copy under releases\binaries\...\windows\

Building and running via Docker (simpler, all OS)
If local toolchain setup is painful, let Docker build everything:

The Dockerfile installs build tools and runs the Falcon build as part of the image build. You can then run the nodes via docker compose up ... without touching local compilers.

Common issues and fixes
- **AVX2 on unsupported CPUs:** remove -mavx2 -DFALCON_AVX2 (set FALCON_USE_AVX2=false) to avo
illegal instruction errors.
- **Missing OpenSSL headers (Linux/macOS):** install libssl-dev (Linux) or brew install openss
and set CPPFLAGS/LDFLAGS to point to it.
- **Permissions:** after building, run chmod +x falcon_cli.
- **Path issues (Windows native):** ensure your compiler and make are on PATH; prefer WSL to simplify.
- **Python deps:** always pip install -r requirements.txt in a venv before running Python scripts.

Verifying the build
- Run ./falcon_cli in Src/Falcon/Falcon to check it executes.
- Run .venv/bin/python Src/Falcon/Falcon/install_platform_binary.py to place the binary in the
platform directory and generate pin values.
- Update Constants.FALCONCLI_PINNED_VERSION_DIGESTS_BY_PLATFORM / Constants.FALCONCLI_EXPECTED_HASH_BY_PLATFORM for release pinning (legacy global pin remains fallback-only).

Notes
- These steps are for advanced users; Docker/Compose is the easiest path to get a working falcon_cli and nodes without local toolchain setup.
- Rebuild the image (docker compose build --no-cache) after changing Falcon sources or compile
flags.