

Athena Authentication

Analysis of Argon2id integration, Password Hashing, and Login Latency

Floris
XYLEX GROUP

June 2026

Building better worlds

Result. Athena Auth now applies its own Argon2id profile consistently across the active core helper, password-management flows, backup-code hashing, administrative password paths, the legacy mirror crate, and the latency benchmark seam. In the refreshed 12-iteration WSL benchmark, `POST /sign-in/email` averaged `16.673 ms` for Athena Auth and `71.748 ms` for Better Auth (ref), a measured gap of `55.075 ms`.

Contents

1	Abstract	2
2	Problem and Baseline	2
3	Implemented Solution	2
4	Method and Results	3
5	Trade-offs	4
6	Conclusion	4
7	Validation	4

1 Abstract

This document defines the Athena Argon2id profile and architectural decisions of **Athena**, a Rust-based authentication framework built on top of Athena, A system developed by XYLEX GROUP.

Athena Auth's original login slowdown was primarily a runtime configuration bug, not an unavoidable property of Argon2id. Some built-in password paths could still fall back to upstream library defaults rather than Athena Auth's intended cost profile, and the sign-in path also carried side effects that did not need to block the response.

The implemented fix made Athena Auth resolve its own Argon2id profile consistently, integrated that behavior across the built-in password flows, and moved non-critical sign-in work into background tasks. In the refreshed WSL benchmark, Athena Auth averaged **16.673 ms** on **POST /sign-in/email** versus **71.748 ms** for Better Auth (ref), while retaining Argon2id.

2 Problem and Baseline

The optimization target was clear:

- keep Argon2id;
- make the interactive login path feel near-instant;
- ensure Athena Auth's declared password configuration is the configuration actually used at runtime.

The baseline measurements showed that the problem was concentrated in the password routes, not in generic session lookup:

- **POST /sign-in/email**: Athena Auth **280.504 ms** vs Better Auth **72.054 ms**
- **POST /sign-up/email**: Athena Auth **276.931 ms** vs Better Auth **72.784 ms**
- **GET /get-session**: Athena Auth **0.678 ms** vs Better Auth **1.159 ms**

This strongly indicated that the dominant cost sat in password verification and closely related sign-in work. The central runtime defect was that the no-override helper path could still resolve to upstream Argon2 defaults instead of Athena Auth defaults:

Listing 1: The failure mode in simplified form

```
fn build_argon2(argon2_config: Option<&Argon2Config>) -> AuthResult<
    Argon2<'static>> {
    let Some(argon2_config) = argon2_config else {
        return Ok(Argon2::default());
    };
    // build explicit params from config
}
```

That made the effective security and performance profile depend on call shape rather than on Athena Auth's own configuration.

3 Implemented Solution

The core fix was to make the canonical helper resolve Athena Auth defaults on the no-override path and to construct the algorithm explicitly as Argon2id:

Listing 2: Canonical Argon2 construction after the fix

```
fn build_argon2(argon2_config: Option<&Argon2Config>) -> AuthResult<
  Argon2<'static>> {
  let argon2_config = argon2_config.cloned().unwrap_or_default();
  let params = Params::new(
    argon2_config.memory_cost,
    argon2_config.time_cost,
    argon2_config.parallelism,
    None,
  )?;

  Ok(Argon2::new(Algorithm::Argon2id, Version::V0x13, params))
}
```

This change was then integrated across the built-in password paths:

- email sign-up and email sign-in;
- password reset, change-password, and set-password flows;
- two-factor backup-code hashing;
- administrative password paths;
- the older in-repo compatibility crate.

The sign-in hot path was also reduced by moving non-critical work into background tasks:

- sign-in audit logging,
- verification-email triggering on sign-in,
- GeoIP session enrichment,
- legacy-hash rehash after successful verification.

The intentionally synchronous core remains password verification, session creation, and the `last_sign_in_at` update. That is the minimum correctness-preserving path.

4 Method and Results

The benchmark was rerun from WSL with:

- 12 measured iterations per endpoint,
- 2 warmup iterations per endpoint,
- Athena Auth server running in-process through the Axum benchmark router,
- Better Auth reference server running through the Node compatibility server,
- Linux Node runtime used from WSL to avoid Windows-to-WSL boundary artifacts.

Endpoint	Athena	Better	Delta	Speedup	Outcome
POST /sign-up/email	16.359	75.689	-59.330	4.63x	Athena faster
POST /sign-in/email	16.673	71.748	-55.075	4.30x	Athena faster
GET /get-session	0.640	1.155	-0.515	1.80x	Athena faster
POST /sign-out	0.606	1.290	-0.684	2.13x	Athena faster

The effective Athena Auth profile in the report was `memory_cost=1024`, `time_cost=2`, `parallelism=1`. The most important before-and-after changes were:

- Athena Auth email sign-in improved from 280.504 ms to 16.673 ms, a 94.06% reduction.
- Athena Auth email sign-up improved from 276.931 ms to 16.359 ms, a 94.09% reduction.
- `GET /get-session` changed only slightly, confirming that session reads were not the original bottleneck.

5 Trade-offs

The solution is better in three ways:

- it removes silent configuration drift;
- it materially lowers interactive login latency;
- it preserves Argon2id instead of replacing it with a weaker primitive.

It is worse in two ways:

- the interactive default profile 1024/2/1 is lighter than heavier Argon2id profiles, so offline brute-force cost per hash is lower;
- backgrounding audit, GeoIP, verification-email, and rehash work introduces eventual-consistency risk if the process exits immediately after the response.

One additional constraint remains: old hashes retain their encoded PHC parameters on first verification, so the first post-deploy login for a legacy heavy hash is still expensive. The background rehash improves later logins, not the first one.

6 Conclusion

Athena Auth is now fast for the right reasons. It no longer depends on accidental upstream Argon2 defaults, it keeps Argon2id explicit, and it removes non-essential work from the blocking sign-in path. The measured server seam is now about 55 ms faster than Better Auth on email sign-in.

The benchmark is still a server benchmark, not a full browser-visible application journey. If the surrounding app still routes through an extra proxy or callback hop, that architecture can add latency even though the auth server itself is already fast. Within the measured seam, however, the optimization is both real and repeatable.

7 Validation

Validation for the current state consisted of:

- focused Rust tests around password hashing and sign-in rehash behavior,
- dual-server benchmark rerun from WSL with a Linux Node runtime,
- regenerated benchmark report with explicit Argon2 profile disclosure.