BOUNDARY BY ORAXLABS

Flask signed-cookie sessions

SOURCE SECURITY REVIEW / PUBLIC OPEN-SOURCE CASE STUDY

Project	Reviewed snapshot	Report version
pallets/flask 3.1.3	SecureCookieSessionInterface Commit 22d924701a6a	1.0 · 23 Nov 2026

Executive overview

Flask’s default session backend is a reasonable fit for small, client-visible state when the application controls cookie settings and signing keys. For sensitive session data or a requirement to revoke one active session on demand, keep the authoritative state on the server.

The code verifies a time-limited signature on incoming session cookies, serializes the whole session back to the client when needed, and adds Vary: Cookie when the request accesses session state. The first deployment check is the emitted cookie policy. In the local fixture, the default cookie had HttpOnly and Path=/, with Secure and SameSite omitted.

Review observations

ID	Observation	Priority	Evidence	Where it applies
FS-01	HTTPS cookie attributes and cache policy need deployment checks	P1	Source + local header check	Deployment-specific
FS-02	The client holds signed state, with no per-session revocation lookup in this interface	P1*	Source + upstream docs	Depends on requirements
FS-03	Fallback key retirement and cookie renewal follow separate rules	P2	Source + upstream test	Operational control
FS-04	In-place nested edits may fail to persist	P2	Source + local fixture	Application code

* P1 when session state must remain private to the server or individual sessions must be revocable. P1 is a deployment decision. P2 is an integration check. These are review priorities for adopters.

Verification summary

24 upstream cases passed in the selected local run. Two additional checks confirmed the default emitted headers and the nested edit save behavior. The reviewed paths support configuration and integration recommendations. No new upstream vulnerability is asserted.

Scope and data flow · 2 | Browser and cache · 3 | Payload and revocation · 4
Rotation and nested updates · 5 | Acceptance plan · 6 | Evidence appendix · 7

Scope and execution flow

DEFAULT SESSION BACKEND / FLASK 3.1.3

Scope SecureCookieSessionInterface and its request-context and response integration. Reviewed cookie flags, signing and verification, age checks, change tracking, cookie removal and response variation. The appendix pins the public source commit and lists the runtime versions.

Method Reviewed the tagged source, linked upstream documentation and selected tests. Ran the upstream test selection against a clean detached checkout. Ran two benign local behavioral checks with Flask's in-process test client. An independent reviewer traced the same paths and reran a focused upstream subset.

Excluded Application authorization, custom session interfaces, cryptographic implementation review of itsdangerous, browser cookie enforcement, live traffic, proxies, CDN cache behavior, and third-party integrations.

From incoming cookie to outgoing response

Stage	Observed behavior	Source evidence
1 · Open	The request context opens session state before URL matching. Missing signing configuration results in a NullSession. Missing cookies result in empty normal sessions.	ctx.py:388–404 sessions.py:83–96,303–335
2 · Verify	The configured cookie is loaded by a timed serializer. Flask passes a maximum age derived from the configured session lifetime. BadSignature yields an empty session.	sessions.py:284–335
3 · Use	The request-context session getter marks access. Top-level updates mark the dictionary modified. These flags drive distinct response choices.	ctx.py:367–375 sessions.py:46–80
4 · Save	Flask applies after-request hooks and saves eligible sessions. The interface omits, expires or reissues the cookie according to state and refresh settings.	app.py:1298–1324 sessions.py:337–385

Trust boundaries

The cookie is client-held data. Flask reconstructs application-readable state after the signature and age check. The framework holds the signing keys. Routes decide what to store and which authorization checks to perform. Browsers and intermediaries determine how emitted headers are applied.

Request context: opening and marking access

Default interface: serializer, load and save

FS-01 Cookie delivery and caching

P1 / APPLICATION AND DELIVERY CONFIGURATION

What Flask emits

The framework defaults to HttpOnly on, Secure off, SameSite unset, and no Domain. The configured cookie name, domain, path, and flags are passed through when setting or expiring the cookie. Path falls back to APPLICATION_ROOT when SESSION_COOKIE_PATH is unset. Python None for SameSite produces an omitted attribute in the tested configuration.

Evidence	Observation	Location
Source	Default values are defined on Flask. The session interface resolves cookie settings and passes them to response methods.	app.py:183–197 sessions.py:171–221,337–385
Upstream	Tests assert configured flags on write and deletion, omitted SameSite and explicit Lax or Strict values. A separate test verifies Vary merging on the response.	test_basic.py:312–380,541–616
Local	A normal non-permanent session issued by the default profile emitted Path=/ and HttpOnly. Secure, SameSite, Domain and Partitioned were absent.	Flask test client Werkzeug 3.1.8

What application teams need to decide

For an HTTPS deployment, enable Secure and keep HttpOnly. Choose SameSite explicitly for the app’s flow. Lax is a reasonable starting point for a typical first-party app. Cross-site sign-in or embedding needs its own browser checks. Keep Domain unset unless intentional cross-host sharing is required. Sensitive state-changing routes need the application’s CSRF protection.

Vary and shared caches

Flask adds Cookie to Vary after request-context session access, and on a cookie write or deletion. A separate in-process check observed Vary: Cookie on membership and length reads and no Vary on an untouched route. The save method leaves Cache-Control to the application. Vary identifies a request field affecting reuse and leaves actual cache behavior to the cache policy. For sensitive personalized responses, set route-appropriate Cache-Control and check the headers at the final delivery boundary.

Acceptance Inspect the issued and deletion headers on HTTPS in the app’s own environment. Confirm expected sign-in, logout and redirects in a browser. For sensitive routes, confirm cache headers survive the configured proxy or CDN. Deployment-specific checks are open.

Cookie defaults

Cookie write, delete and Vary behavior

Request-context session access flag

FS-02 Payload and revocation

P1 WHEN REQUIRED / DATA OWNERSHIP AND SESSION DESIGN

What is protected

The interface uses a timed signing serializer over tagged JSON. The signature protects the session's integrity under the configured keys. The payload is client-visible, as Flask's Quickstart states. A user may hold a signed copy of the session state. Treat every field in that payload as data that can be disclosed to the holder.

Two meanings of expiry

Boundary	Observed contract	Evidence
Browser	Permanent sessions receive an Expires date based on the configured lifetime. Non-permanent sessions omit that date.	sessions.py:223–231,372–384 test_basic.py:423–452
Server	open_session passes the configured lifetime to the timed serializer on each incoming cookie load. The check is applied before Flask can read whether this session is permanent.	sessions.py:323–335 app.py:185
Renewal	Modified nonempty sessions are written. Permanent sessions can also be written on ordinary requests when refresh is enabled. A freshly serialized cookie can have a new signature timestamp and expiration.	sessions.py:233–247,369–385 test_basic.py:500–538

What individual revocation requires

When the session is changed to empty, Flask asks the client to remove that cookie. The default load path checks the cookie and signature age. There is no per-session revocation lookup in that path. An application that requires immediate revocation needs a server-side decision, such as authoritative session storage or a validity check tied to the account or session.

Recommendation Inventory the fields written into session. Keep confidential values and authoritative authorization state server-side. Define idle, absolute and logout expectations independently. Choose server-side revocation where the business requirement calls for it.

Acceptance Review the application's allowed field list. With synthetic accounts in the application's test environment, verify the expected behavior after logout, disabling an account and reaching the required session age. The upstream run checks expiry headers and refresh policy. Application revocation and elapsed-time behavior were outside the executed checks.

Flask Quickstart: client-held session data

Timestamped verification and deletion

FS-03 and FS-04

KEY ROLLOUT AND SESSION WRITES

FS-03 Key retirement and reissue follow separate rules

Evidence Flask builds the signer with fallbacks followed by the active key. Fallbacks preserve verification during planned rotation. New writes use the active key. The normal save logic still depends on modification or permanent-session refresh. Nothing in open_session marks the session modified solely because it was read. See sessions.py:233–247,303–335,369–385.

Runtime evidence The upstream fixture signs a normal session with an initial key, observes that it appears empty after an unrecognized key change, and restores continuity by including the original key in the fallback list. This passed in the selected run. It does not measure fleet rollout, verify automatic reissue of unchanged cookies, or establish when fallbacks can be retired.

Action and acceptance Keep a dated key inventory, confirm all workers use the intended active key and fallbacks, and define when older keys leave the verification set. Test expected session continuity during planned rotation and the expected sign-in behavior after retirement. Treat an incident-driven key change as a separate runbook.

Existing planned-rotation test

itsdangerous 2.2 key rotation contract

FS-04 Nested session edits may not persist

Evidence SecureCookieSession attaches a modification callback to the top-level mapping. Mutating a nested object in place does not trigger that callback. A later write for another reason can carry the change, so the behavior depends on the response’s write branch. See sessions.py:57–80,233–247,369–385.

Runtime evidence In an in-process test with a non-permanent session and refresh disabled, editing a nested theme preference alone produced no Set-Cookie and the next request read the prior value. Marking the session modified produced Set-Cookie and the next request read the updated value.

Action and acceptance Use top-level reassignment or set session.modified explicitly in routes that edit nested data. Verify persistence across requests in the app’s actual session and refresh modes. Keep authoritative account state outside the client cookie.

Modification tracking implementation

Existing refresh and write conditions test

Deployment acceptance plan

OWNERS AND OBSERVABLE PASS CONDITIONS

Order	Action and suggested owner	Acceptance evidence
1 · FS-01	Platform + app. Make the production cookie policy explicit. Inspect new and deleted cookies, login redirects, and cache policy at the application edge.	Headers and browser flows above match agreed requirements.
2 · FS-02	App + identity. Inventory fields. Define which decisions require current server state and how individual revocation works.	Sensitive values stay server-side; required revocation and expiration behavior passes staging tests.
3 · FS-03	Platform. Document signing key rollout, overlap, and removal, with a distinct incident process.	Staging verifies continuity while intended and the expected transition after retirement.
4 · FS-04	App. Find routes that mutate nested state and choose a consistent persistence pattern.	A later request reads the intended state in each supported refresh profile.

What to retain

Record the reviewed application revision and the tested environment. For each check, keep expected and observed behavior and the date. Record header attributes without retaining cookie values. Recheck application behavior when a session interface, cookie scope, lifetime, routing or cache rule changes.

Review status

This is an independent public-source review by Boundary by OraXLabs. No Flask modification or application deployment change was evaluated. The local results establish behavior at the reviewed revision and dependency versions. The acceptance plan applies to a specific app once its requirements and deployment are available.

Upstream configuration reference at the reviewed revision

Upstream web security guidance at the reviewed revision

Verification and source index

EVIDENCE REGISTER / 23 NOV 2026

Revision 22d924701a6ae2e4cd01e9a15bbaf3946094af65

Public GitHub release: pallets/flask 3.1.3

Runtime Python 3.13.13 · pytest 9.0.3 · Werkzeug 3.1.8 · itsdangerous 2.2.0. Ran directly against checkout source with PYTHONPATH=src.

Run	What was tested	Limit
24 passed	Selected upstream test nodes across basic, config, request context, interface, testing and helpers. Selection keyword: session or cookie.	0 failed, 0 skipped 200 other cases deselected
14 passed	Independent rerun of the upstream basic-file subset, covering settings, age headers, fallbacks, refresh and Vary.	Subset of 24 116 other cases deselected
2 local checks	Default cookie attributes and nested preference edits, with and without explicitly marking the session modified.	In-process client
3 local checks	Independent membership, length and untouched-route checks for Vary behavior.	In-process client

Re-run the selected upstream tests

```
PYTHONPATH=src PYTEST_DISABLE_PLUGIN_AUTOLOAD=1 python3 -m pytest -q -p no:cacheprovider tests/test_basic.py tests/test_config.py tests/test_reqctx.py tests/test_session_interface.py tests/test_testing.py tests/test_helpers.py -k "session or cookie"
```

Supporting code and test assertions

Session flags, modification tracking and null sessions

Response finalization after request hooks

Cookie behavior, key rotation, and expiry assertions

Response refresh, Vary and cache variation assertions

Evidence strength varies by observation. Local fixtures verify Flask response behavior. Browser, intermediary and deployment decisions remain application-specific.