RFC 0002 — USG Entitlement Token Profile

A Standards-Track Specification for Tokenized Sports Access

Document Series: USG-RFC
Series Number: 0002
DOI: 10.5281/zenodo.17781620
Updates: RFC 0001
Relation: isSupplementTo 10.5281/zenodo.17565794
Author: Scott Jellen (Independent Researcher)
Date: December 2025

Status: Draft for Review and Implementation Feedback

Status of This Memo

This document defines the **USG Entitlement Token Profile**, a standards-track specification for token-based authorization within the Universal Sports Graph (USG) ecosystem.
It updates RFC 0001 by normatively defining token structure, required claims, optional claims, validation rules, security properties, privacy requirements, and interoperability expectations.

This document is intended for public review and pilot implementation.
Distribution is unlimited.

---

1. Introduction

USG Entitlement Tokens provide portable, cryptographically verifiable authorization to access live and on-demand sports events across distributed platforms.
Tokens enable interoperability between leagues, clearinghouses, and distributors without requiring shared identity systems or centralized authentication flows.

This specification defines:

- token structure
- required and optional claims
- temporal semantics
- signature and verification rules
- replay protection
- territory and rights enforcement
- error codes
- security and privacy considerations

RFC 0002 forms the normative foundation for USG access-control operations.

---

2. Terminology

- **USG**: Universal Sports Graph
- **Issuer**: Entity authorized to generate entitlement tokens (e.g., clearinghouse)
- **Verifier**: Platform or service validating tokens (e.g., streaming provider)
- **Subject (`sub`)**: Pseudonymous identifier representing the end user
- **Entitlement Token**: JWT-based authorization artifact defined by this RFC
- **Registry**: Rights Registry defined in RFC 0001
- **Replay Cache**: Data structure used to detect reused `jti` values

All normative language uses RFC 2119 keywords.

---

3. Token Overview

USG tokens authorize:

- access to specific events (`event_id`)
- within specific territories (`territory`)
- within defined temporal windows (`window`)
- with defined permissions (`scope`)

Tokens MUST be signed JWTs and MUST be treated as opaque by clients.

---

4. Token Structure and Format

USG Entitlement Tokens MUST use the JSON Web Token (JWT) structure defined in RFC 7519.

```
base64url(header) + "." + base64url(payload) + "." + base64url(signature)
```

4.1 Header

```
{
  "alg": "ES256",
  "kid": "usg-key-2025-01",
  "typ": "JWT",
  "usg_ver": "1.0"
}
```

4.1.1 Header Requirements

- `alg`:
  - `ES256` MUST be used for all production multi-party deployments.
  - `RS256` MAY be used where required by organizational policy.
  - `HS256` MUST NOT be used in production deployments involving multiple organizations and SHOULD be limited to sandbox-only environments.

- `kid`: MUST identify a trusted Key Registry entry.
- `typ`: MUST be `"JWT"`.
- `usg_ver`: MUST identify the token profile version (e.g., `"1.0"`).

4.2 Payload

The payload contains all required and optional claims defined in Sections 5 and 6.

4.3 Signature

The signature MUST be computed over the header and payload using the algorithm specified in `alg`.

Issuers MUST:

- store private keys in secure hardware (HSM, TEE, or equivalent);
- treat keys as non-exportable once provisioned;
- log and audit signing operations.

Verifiers MUST reject any token with:

- an invalid signature,
- an untrusted signing key,
- or an unrecognized issuer.

4.4 Token Opaqueness

Clients MUST treat tokens as opaque strings.
Only verifiers MAY parse payloads.
Clients MUST NOT interpret or rely on claim semantics.

4.5 Size Constraints

Tokens SHOULD be less than 2 KB.
Issuers SHOULD avoid unnecessary or oversized optional claims.

4.6 Deterministic Encoding

Issuers SHOULD serialize JSON deterministically (stable key ordering and canonical formatting).
Verifiers MUST NOT reject tokens solely due to key order or irrelevant whitespace differences.

---

5. Required Claims

A valid USG token MUST contain the following claims:

```
{
  "iss": "usg-clearinghouse",
  "sub": "wallet_8234",
  "aud": ["youtubetv"],
  "jti": "c92d21a1-85bd-4b7d-bf20-52ac5ef97293",
```

```
  "iat": 1733359200,
  "nbf": 1733359200,
  "exp": 1733369999,

  "event_id": "nba_2025_gsw_lal_0410",
  "scope": ["view"],
  "territory": ["US", "CA"],

  "window": {
    "start": "2025-04-10T19:00Z",
    "end":   "2025-04-10T23:59Z"
  }
}
```

5.1 Required Claim Semantics

- `iss`: MUST identify a trusted issuer (e.g., clearinghouse or delegated authority).
- `sub`: MUST be a pseudonymous subject identifier and MUST NOT contain PII.
- `aud`: MUST be an array of verifier identifiers; the verifying platform MUST appear in this list.
- `jti`: MUST be globally unique per token and is used for replay protection.
- `iat`, `nbf`, `exp`: MUST define a valid temporal window in Unix time (seconds since epoch).
- `event_id`: MUST correspond to a valid event in the Rights Registry (RFC 0001).
- `scope`: MUST include `"view"` for primary playback authorization.
- `territory`: MUST list allowed regions (e.g., ISO 3166-1 alpha-2 country codes).
- `window.start`, `window.end`: MUST define the canonical access window for the event and MUST align with the Rights Registry.

`exp` MUST NOT exceed `window.end`.

---

6. Optional Claims

Optional claims MAY appear in a USG token to support extended scenarios, device policies, promotions, or future extensions.

Optional claims MUST NOT weaken or override the semantics of required claims.

6.1 General Rules for Optional Claims

- Optional claims MAY be omitted without affecting token validity.
- Optional claims MUST NOT contradict `scope`, `territory`, `event_id`, or `window`.
- Optional claims MUST NOT introduce personally identifiable information (PII).
- Optional claims MAY be used for commercial logic, device limits, or UX hints, but MUST NOT alter the core authorization contract defined by required claims.
- Optional claims SHOULD be compact so that the overall token size remains below 2 KB wherever possible.

6.2 Standard Optional Claims

6.2.1 `device_cap`

Indicates the maximum number of concurrent devices allowed for this entitlement.

```
"device_cap": 2
```

Verifiers MAY enforce device caps according to local policy and user agreements.

6.2.2 `geo_override`

Indicates conditions under which territorial enforcement may differ from standard policies (e.g., venue-based access).

```
"geo_override": "stadium_exempt"
```

Verifiers MUST apply overrides conservatively and only where explicitly supported by consortium or local policy.

6.2.3 `promo_code`

Represents a promotional or discount code used during entitlement purchase.

```
"promo_code": "MARCHMADNESS25"
```

This claim MUST NOT affect authorization semantics and MAY be used only for analytics, billing reconciliation, or UX rendering.

6.2.4 `settlement_hint`

Provides an optional link between the entitlement and a settlement transaction.

```
"settlement_hint": "txn_5567"
```

This claim MUST NOT be interpreted as authoritative. Canonical transaction binding occurs in the Clearinghouse and related specifications (e.g., RFC 0003).

6.3 Future Optional Claims

Future USG specifications MAY define new optional claims.

- New optional claims SHOULD be tied to a future `usg_ver` profile version.
- Verifiers MUST ignore unknown optional claims.
- Optional claims MUST NOT introduce breaking changes for legacy verifiers.

6.4 Deprecated or Prohibited Claims

The following are NOT permitted:

- Optional claims that contain PII (e.g., email address, IP address, device fingerprint).
- Optional claims that conflict with rights defined in the Rights Registry.
- Optional claims that extend `window.end` or expand `territory` beyond Registry-approved bounds.

---

7. Signing and Verification Rules

This section defines normative requirements for issuing and validating USG Entitlement Tokens.

7.1 Signing Requirements (Issuers)

Issuers MUST:

1. **Algorithm Selection**
    - Use `ES256` for all production multi-party deployments (RECOMMENDED).
    - `RS256` MAY be used where required by organizational or regulatory policy.
    - `HS256` MUST NOT be used in multi-party production environments and SHOULD be restricted to sandbox or strictly internal testing.
2. **Private Key Protection**
    - Store private keys in secure hardware (HSM, TEE, or equivalent).
    - Treat keys as non-exportable once provisioned.
    - Log and audit key usage and signing operations regularly.
3. **Key Rotation**
    - Rotate signing keys at least every 90 days.
    - Use unique `kid` values per key version.
    - Retain verification capability for retired keys for at least a short grace period (e.g., 7 days) to cover in-flight tokens.
4. **Canonical Claim Binding**
    - Ensure `exp` is less than or equal to `window.end`.
    - Ensure `territory` is compatible with the Rights Registry entry for the event.
    - Ensure `scope` includes `"view"` for viewer entitlements.

7.2 Verification Requirements (Verifiers)

Verifiers MUST:

1. **Signature Validation**
    - Validate the token signature using the algorithm in `alg` and the public key referenced by `kid`.
    - Reject tokens if the key is not obtained from a trusted Key Registry.
2. **Temporal Validation**

- Reject tokens where `exp < now` (`error_expired`).
- Reject tokens where `nbf > now` (`error_not_yet_valid`).
- Reject tokens where `iat > exp` (`error_invalid_claim`).

3. **Audience Enforcement**
   - Ensure the verifier's identifier appears in `aud`.
   - Reject tokens where the verifier identifier is not included (`error_invalid_audience`).
4. **Event Validity**
   - Confirm that `event_id` corresponds to a valid entry in the Rights Registry.
   - Reject tokens whose events are not found or have been withdrawn (`error_event_invalid`).
5. **Scope Enforcement**
   - Ensure `scope` includes `"view"` for primary playback authorization.
   - Reject tokens that omit `"view"` (`error_scope_denied`).
6. **Territory Enforcement**
   - Determine the user's effective region using privacy-compatible methods.
   - Ensure the effective region is contained within `territory`.
   - Reject tokens where the region is not authorized (`error_region_block`).
7. **Replay Protection**
   - Implement replay protection using `jti` as defined in Section 8.
   - Maintain a replay cache of seen `jti` values.
   - Reject tokens where `jti` has already been observed (`error_replay_detected`).
8. **Structural and Type Validation**
   - Reject tokens with malformed JSON.
   - Reject tokens that use invalid types (e.g., non-array `scope` or `territory`).
   - Reject tokens with unparseable timestamps (`error_token_malformed` or `error_invalid_claim`).

7.3 Trust Model and Key Distribution

- Verifiers MUST obtain issuer public keys from a trusted Key Registry.
- The Key Registry MUST be distributed via mutually authenticated TLS, pinned certificates, or signed out-of-band documents.
- Verifiers MUST reject keys not originating from an approved list of issuers.

7.4 Failure Behavior

Verifiers MUST fail closed:

- Tokens that cannot be validated, parsed, or trusted MUST be rejected.
- Tokens missing required claims MUST be rejected.
- Unknown optional claims MUST be ignored and MUST NOT cause rejection on their own.

Error codes returned to callers MUST align with Section 10.

7.5 Multi-Issuer Environments

In multi-issuer ecosystems:

- Each issuer MUST publish its own Key Registry entry.
- All issuers participating in a shared deployment MUST align on a common `usg_ver` token profile.
- Verifiers MUST map `iss` values to a configured list of approved issuers and reject tokens from unknown issuers.

7.6 Token Freshness (Recommended)

Verifiers SHOULD:

- Reject tokens with excessively old `iat` values, even if `nbf` and `exp` are technically valid.
- Prefer short-lived tokens where possible to reduce replay and theft windows.

---

8. Replay Protection

Replay protection ensures that a valid entitlement token cannot be reused outside its intended context. USG relies on the `jti` claim as the canonical replay identifier.

8.1 Requirements for Issuers

Issuers MUST:

1. Generate globally unique `jti` values using cryptographically strong randomness or UUIDv4.
2. Maintain a short-term record of issued `jti` values for at least the token's lifetime.
3. Reject issuance requests that would result in duplicate `jti`.
4. Bind each `jti` to the specific issued claim set (event, scope, territory).

8.2 Requirements for Verifiers

Verifiers MUST:

1. Maintain a replay cache of previously seen `jti` values.
2. Store `jti` in hashed form where possible (e.g., SHA-256).
3. Reject tokens whose `jti` already exists in the cache (`error_replay_detected`).
4. Expire replay entries no later than `exp`.
5. Treat replay attempts as security events and log accordingly.

8.3 Multi-Device Considerations

If users are permitted to access entitlements across multiple devices:

- Each token MUST still have a unique `jti`.
- Multi-device logic MUST be handled via `device_cap` or platform policy.
- Reuse of the same token across devices MUST still trigger replay detection.

8.4 Clearinghouse Interaction

The Clearinghouse MAY:

- audit `jti` patterns to detect systemic abuse;
- correlate `jti` entries with settlement records;
- detect anomalous issuance behavior by platforms.

The Clearinghouse is **not** responsible for final replay enforcement — this occurs at verifiers.

8.5 Token Forwarding and Theft

Replay protection mitigates token theft and forwarding:

- Extracted tokens reuse the same `jti`.
- Any replay of the same token MUST be rejected.
- Replay detection MUST NOT rely on IP address, user agent, or device fingerprint.

8.6 Recommended Hardening

Verifiers SHOULD:

- reject tokens with unusually old `iat` values;
- limit maximum acceptable clock skew;
- monitor high-frequency `jti` patterns for abuse.

---

9. Territory and Rights Enforcement

This section defines how verifiers must enforce territorial and temporal limitations associated with a given event.

9.1 Territory Enforcement Requirements

Verifiers MUST:

1. Determine the request's effective region using privacy-compliant techniques.
2. Ensure the effective region appears within the token's `territory` list.
3. Reject tokens when the region is not authorized (`error_region_block`).

9.2 Relation to the Rights Registry

- Token `territory` MUST match or be a subset of the territory list defined in the Rights Registry (RFC 0001).
- Issuers MUST NOT generate entitlements for regions not permitted by the Registry.
- Verifiers MUST reject tokens that contradict Registry-defined territories.

9.3 Access Window Enforcement

Verifiers MUST enforce the event access window:

- Access MUST occur between `window.start` and `window.end`.
- `exp` MUST NOT exceed `window.end`.
- Attempts outside the canonical window MUST be rejected (`error_window_closed`).

9.4 Scope Enforcement

- Tokens MUST include `"view"` in `scope` for playback authorization.
- Additional scopes (e.g., `"replay"`, `"analytics"`) MAY be used for extended features.
- Unknown scopes MUST be ignored.

9.5 Geo Overrides

Tokens MAY include a geo-override claim:

```
"geo_override": "stadium_exempt"
```

Verifiers MAY honor overrides **only** when:

- the override is permitted by consortium rules;
- the override does not contradict the Rights Registry;
- the override does not extend the temporal window;
- the override does not expand territories beyond Registry limitations.

9.6 Multi-Region Access

- Tokens MAY authorize multiple territories.
- Verifiers MUST treat each region code independently.
- Multi-territory tokens MUST still reflect Registry constraints.

9.7 Enforcement Failures

Verifiers MUST reject tokens when:

- the region is not within `territory`;
- territory codes are malformed;
- the region contradicts the Registry;
- the access time violates the canonical window.

Error codes MUST follow Section 10.

9.8 Privacy Constraints

Territorial enforcement MUST use privacy-preserving region inference:

- coarse geolocation;
- account region;
- platform-level billing region.

Precise GPS or device fingerprinting MUST NOT be required or used.

---

10. Error Codes and Failure Modes

This section defines canonical error codes for all token validation failures.

Verifiers MUST return the earliest fatal error encountered and MUST NOT attempt partial authorization.

Error Code	Description
`error_invalid_signature`	Token signature invalid or unverifiable
`error_untrusted_issuer`	`iss` not recognized or key not from trusted source
`error_token_malformed`	Token structure invalid or not a valid JWT
`error_missing_claim`	Required claim missing or null
`error_invalid_claim`	Claim violates type, semantics, or Registry rules
`error_expired`	Token expired (`exp < now`)
`error_not_yet_valid`	Token not valid yet (`nbf > now`)
`error_invalid_audience`	Verifier not included in `aud`
`error_region_block`	User region not in `territory`
`error_scope_denied`	`"view"` missing from `scope`
`error_event_invalid`	`event_id` not found or withdrawn
`error_window_closed`	Access occurs outside `window`
`error_replay_detected`	`jti` already used
`error_key_expired`	Signing key no longer valid or revoked
`error_internal`	Non-token internal error

Verifiers SHOULD implement error prioritization in the following order:

1. malformed / structural
2. signature / trust
3. missing claims
4. invalid claims
5. temporal errors
6. audience
7. registry violations
8. territory
9. scope
10. replay

---

11. Security Considerations

Security is central to the USG Entitlement Token design.
This section summarizes the mandatory controls for issuers, verifiers, and the consortium.

11.1 Integrity and Authenticity

USG tokens MUST:

- use asymmetric cryptography for all multi-party deployments;
- use secure algorithms (`ES256` RECOMMENDED, `RS256` permitted);
- include a `kid` header pointing to a trusted Key Registry entry;
- be signed using private keys stored in secure hardware (HSM/TEE).

Verifiers MUST:

- validate signatures before examining any claim;
- reject tokens signed with unknown or revoked keys;
- validate that `alg` matches the key type.

11.2 Key Lifecycle Management

Issuers MUST:

- rotate signing keys every 90 days or less;
- use unique `kid` values per rotation;
- remove retired keys from issuance while keeping them verifiable for in-flight tokens;
- revoke keys immediately if compromise is suspected.

Key compromise MUST be treated as a severe security incident.

11.3 Replay Protection as a Security Control

Replay attacks enable unauthorized reuse of entitlements.

To mitigate:

- `jti` MUST be unique per token;
- verifiers MUST maintain replay caches;
- replay attempts MUST be treated as security events.

Replay protection MUST NOT depend on device fingerprinting or IP address.

11.4 Token Theft and Extraction

Verifiers MUST assume that:

- tokens may be intercepted, copied, or exfiltrated;
- tokens may be presented from devices or networks unrelated to issuance.

Thus:

- all authorization MUST rely on token cryptography and claims;
- clients MUST treat tokens as secrets and avoid exposing them to third-party scripts.

11.5 Temporal Enforcement

Incorrect or loose time handling enables:

- early access (`nbf` mis-enforced);
- late access (`exp` ignored);

- extended window exploitation.

Verifiers MUST:

- enforce `nbf`, `iat`, and `exp` strictly;
- apply reasonable clock-skew allowances (e.g., ±30 seconds);
- reject tokens that violate window alignment with `window.end`.

11.6 Logging and Audit

Logs MUST NOT contain:

- full unredacted tokens;
- private keys or key material;
- PII.

Safe logging MUST include:

- hashed `jti`;
- issuer ID;
- event ID;
- error codes.

Audit systems SHOULD correlate validation anomalies for detection.

---

12. Privacy Considerations

USG is designed to minimize personal data usage and comply with global privacy frameworks (GDPR, CCPA, etc.).

12.1 Pseudonymity of `sub`

- `sub` MUST be pseudonymous.
- `sub` MUST NOT contain PII (e.g., email, phone, real name, billing address).
- Verifiers MUST NOT use `sub` to track users across platforms.

12.2 No PII in Tokens

Tokens MUST NOT include:

- names;
- email addresses;
- device fingerprints;
- precise geolocation;
- IP addresses;
- persistent identifiers linking across domains.

12.3 Minimal Regional Inference

Territory enforcement MUST use:

- coarse geolocation;
- account region;
- billing region;
- platform metadata.

It MUST NOT use:

- GPS data;
- Wi-Fi triangulation;
- persistent device identifiers;
- hardware serial numbers.

12.4 Replay Cache Privacy

Replay caches:

- MUST store only hashed representations of `jti`;
- MUST NOT store user identifiers;
- MUST expire entries promptly upon `exp`.

12.5 Data Retention

Operators SHOULD:

- minimize data retention;
- retain authorization logs only for fraud detection or regulatory reasons;
- avoid retaining non-essential validation data.

12.6 Clearinghouse Privacy

Clearinghouse operations MUST NOT expose user identity.

- Settlement records MUST be pseudonymous.
- Audit datasets MUST be anonymized.

---

13. IANA and Namespace Considerations

USG uses a provisional namespace for identifiers, schemas, and artifacts.

13.1 Provisional Namespace Definition

```
urn:usg:{category}:{identifier}
```

13.2 Examples

- `urn:usg:schema:entitlement-token:1.0`
- `urn:usg:token:profile:1.0`
- `urn:usg:event:nba_2025_gsw_lal_0410`

- `urn:usg:key:issuer:2025-01`

These URNs are intended for internal USG ecosystem interoperability and reference.
No global IANA registration is required at this time.

13.3 Future Standardization Path

Future versions MAY:

- apply for registration with IANA under URN namespaces;
- align with OECD Digital Public Infrastructure frameworks;
- be submitted to IETF for community review.

13.4 Change Control

The USG Consortium (or equivalent governing body) SHOULD:

- maintain public documentation of namespace usage;
- publish revision histories when `usg_ver` versions change;
- ensure backward compatibility for at least one version cycle.

Appendix A — Sample Tokens and Validation Walkthrough

This appendix provides reference examples to support implementers.
Examples include valid tokens, minimal tokens, invalid tokens, and a complete verifier walkthrough.

A.1 Valid Token Example (Annotated)

Header:

```
{
  "alg": "ES256",
  "kid": "usg-key-2025-01",
  "typ": "JWT",
  "usg_ver": "1.0"
}
```

Payload:

```
{
  "iss": "usg-clearinghouse",
  "sub": "wallet_8234",
  "aud": ["youtubetv"],
  "jti": "c92d21a1-85bd-4b7d-bf20-52ac5ef97293",

  "iat": 1733359200,
  "nbf": 1733359200,
  "exp": 1733369999,
```

```
  "event_id": "nba_2025_gsw_lal_0410",
  "scope": ["view", "replay"],
  "territory": ["US", "CA"],

  "window": {
    "start": "2025-04-10T19:00Z",
    "end":   "2025-04-10T23:59Z"
  }
}
```

This token is valid under Sections 5–10.
A verifier that enforces Registry data, territory, and replay protection will accept it.

A.2 Minimal Valid Token

This is the smallest valid entitlement token meeting all required claims:

```
{
  "iss": "usg-clearinghouse",
  "sub": "wallet_1111",
  "aud": ["espn"],
  "jti": "3ff9c4b0-fdf1-4b0d-b5f0-3f7b39cf32e1",

  "iat": 1733359200,
  "nbf": 1733359200,
  "exp": 1733362800,

  "event_id": "nba_2025_bos_mia_0410",
  "scope": ["view"],
  "territory": ["US"],

  "window": {
    "start": "2025-04-10T19:00Z",
    "end":   "2025-04-10T22:00Z"
  }
}
```

This demonstrates that optional claims are not required for a valid entitlement.

A.3 Invalid Token Examples

A.3.1 Invalid Audience

```
{
  "iss": "usg-clearinghouse",
```

```
  "aud": ["hulu"]
}
```

Verifier: `youtubetv`
Result: `error_invalid_audience`

A.3.2 Territory Mismatch

```
{
  "territory": ["UK", "IE"]
}
```

User region: `US`
Result: `error_region_block`

A.3.3 Missing Required Claims

```
{
  "event_id": "nba_2025_gsw_lal_0410",
  "scope": ["view"]
}
```

Missing:
`iss`, `sub`, `aud`, `jti`, `iat`, `nbf`, `exp`, `territory`, `window`
Result: `error_missing_claim`

A.3.4 Expired Token

```
{
  "exp": 1600000000
}
```

Result: `error_expired`

A.3.5 Replay Attack

Reuse of a `jti` seen previously:

```
{
  "jti": "c92d21a1-85bd-4b7d-bf20-52ac5ef97293"
}
```

Result: `error_replay_detected`

A.3.6 Window Violation

```
{
  "exp": 1733369999,
  "window": {
    "start": "2025-04-10T19:00Z",
    "end":   "2025-04-10T20:00Z"
  }
}
```

`exp` exceeds `window.end`
Result: `error_invalid_claim`

A.4 Validation Walkthrough (Step-by-Step)

A verifier MUST evaluate tokens in this order:

1. Validate structure: three base64url segments.
2. Decode header and payload.
3. Validate signature with trusted key.
4. Confirm presence of all required claims (Section 5).
5. Validate types (`territory` and `scope` MUST be arrays).
6. Validate `iat`, `nbf`, and `exp` with clock-skew tolerance.
7. Verify `aud` includes the verifier's identifier.
8. Lookup `event_id` in the Rights Registry.
9. Validate `territory` intersection with user region.
10. Validate `"view"` is present in `scope`.
11. Validate `window.start` and `window.end`.
12. Confirm `exp <= window.end`.
13. Check `jti` against replay cache.
14. Accept token.

A compliant verifier MUST reject at the first failing step.

A.5 Reference Pseudocode

```
def validate_usg_token(token, verifier_id, user_region, registry,
key_registry, replay_cache):
    header, payload = decode_and_verify_structure(token)

    # --- Signature and Issuer ---
    key = key_registry.get(header["kid"])
    if key is None or not verify_signature(token, key):
        raise Error("error_invalid_signature")

    # --- Required Claims ---
    for field in REQUIRED_CLAIMS:
        if field not in payload:
            raise Error("error_missing_claim")

    # --- Temporal Validation ---
    now = utc_now()
    if payload["exp"] < now:
        raise Error("error_expired")
    if payload["nbf"] > now:
        raise Error("error_not_yet_valid")
    if payload["iat"] > payload["exp"]:
        raise Error("error_invalid_claim")

    # --- Audience ---
    if verifier_id not in payload["aud"]:
        raise Error("error_invalid_audience")

    # --- Registry Lookup ---
    event = registry.lookup(payload["event_id"])
    if event is None:
        raise Error("error_event_invalid")

    # --- Territory Enforcement ---
    if user_region not in payload["territory"]:
        raise Error("error_region_block")

    # --- Scope ---
    if "view" not in payload["scope"]:
        raise Error("error_scope_denied")

    # --- Window Alignment ---
    window_end = timestamp(payload["window"]["end"])
    if payload["exp"] > window_end:
        raise Error("error_invalid_claim")
    if now > window_end:
        raise Error("error_window_closed")

    # --- Replay Protection ---
    hashed_jti = hash(payload["jti"])
    if replay_cache.seen(hashed_jti):
        raise Error("error_replay_detected")
    replay_cache.add(hashed_jti, expiry=payload["exp"])
```

```
    return "valid"
```

A.6 Notes for Implementers

- The examples provided illustrate typical failure cases but are not exhaustive.
- Platforms SHOULD construct automated validation suites using these examples.
- Tools MAY canonicalize JSON before signature verification, but MUST NOT require key ordering.

Appendix B — Reference Verification Pipeline & State Machine

This appendix defines a complete reference pipeline for token validation, along with an abstract state machine and extended pseudocode for implementers.
It is non-normative but RECOMMENDED for production deployments.

B.1 Validation Pipeline Overview

A verifier MUST evaluate tokens in the following order.
This order enforces "fail fast" semantics and reduces attack surface:

1. **Structural Validation**
    - Confirm token has three segments separated by `.`
    - Confirm header and payload are valid base64url
2. **Signature & Trust**
    - Parse header
    - Retrieve `kid`
    - Validate signature using trusted Key Registry
3. **Required Claim Presence**
    - MUST include all required claims defined in Section 5
    - Reject immediately on missing claim
4. **Claim Type & Format Validation**
    - Arrays MUST be arrays
    - Strings MUST be strings
    - Timestamps MUST parse
    - `window` MUST contain valid ISO-8601 timestamps
5. **Temporal Validation**
    - Enforce `iat`, `nbf`, `exp`
    - Apply minimal clock skew (e.g., ±30 seconds)
    - Reject expired or premature tokens

6. **Audience Validation**
    - Verifier MUST appear in `aud`
7. **Registry Validation**
    - Confirm event exists in Registry
    - Confirm territory is compliant with Registry constraints
8. **Territory Validation**
    - Verify the user's effective region is allowed
    - Apply geo-overrides when permitted
9. **Scope Validation**
    - MUST include `"view"`
10. **Window Validation**
    - `window.start <= now <= window.end`
    - `exp <= window.end`
11. **Replay Protection**
    - Hash `jti`
    - Reject if seen before
    - Store new `jti` until expiry
12. **Authorization Decision**
    - If all checks pass → VALID
    - Otherwise → return specific error code (Section 10)

B.2 Full Extended Pseudocode

This pseudocode is intended to be readable by implementers across languages.
It follows the canonical pipeline.

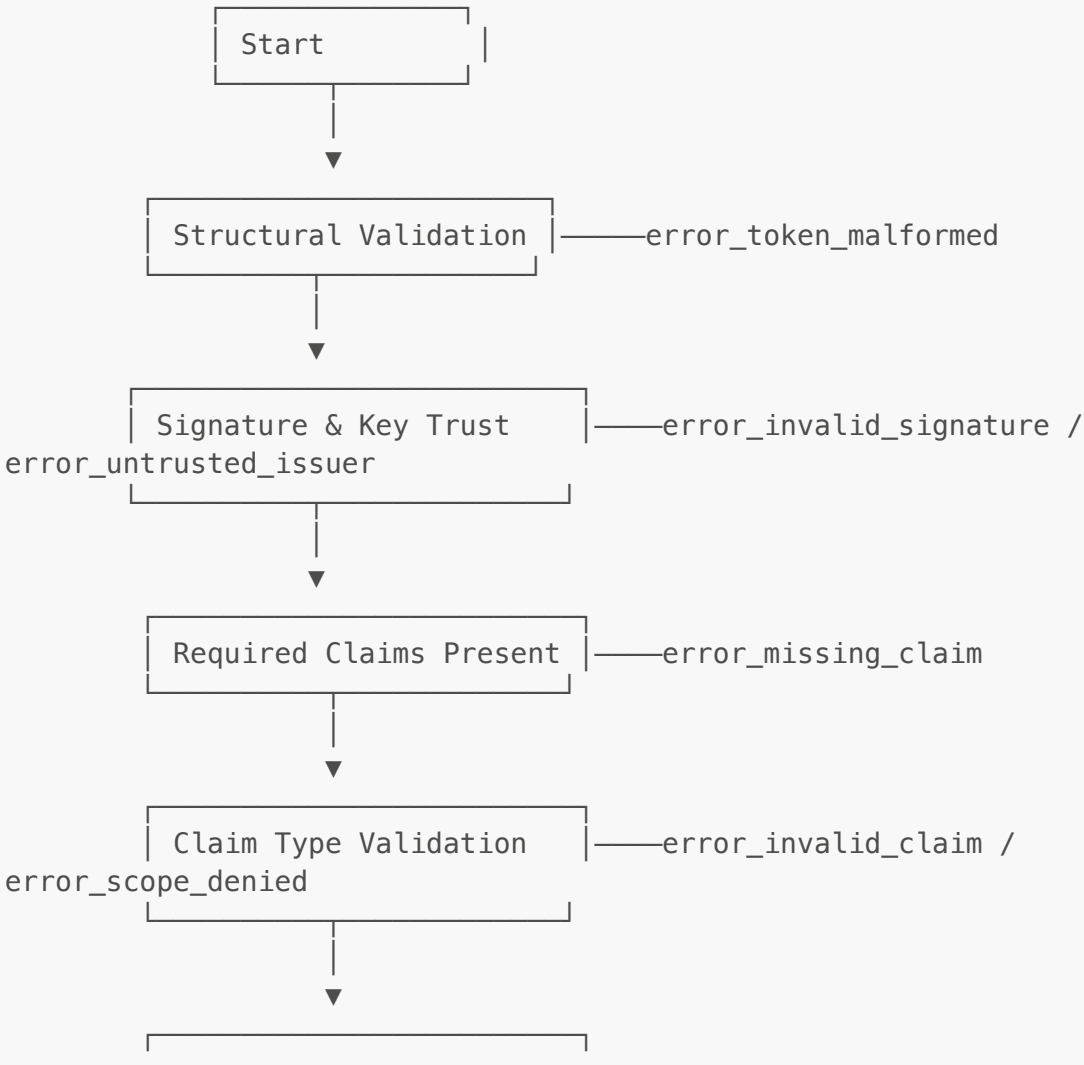
```
def validate_usg_token(
    token,
    verifier_id,
    user_region,
    registry,
    key_registry,
    replay_cache,
    now_utc
):
    # --- 1. Structural Validation ---
    header, payload = decode_and_validate_jwt_structure(token)
    if header is None or payload is None:
        return error("error_token_malformed")
```

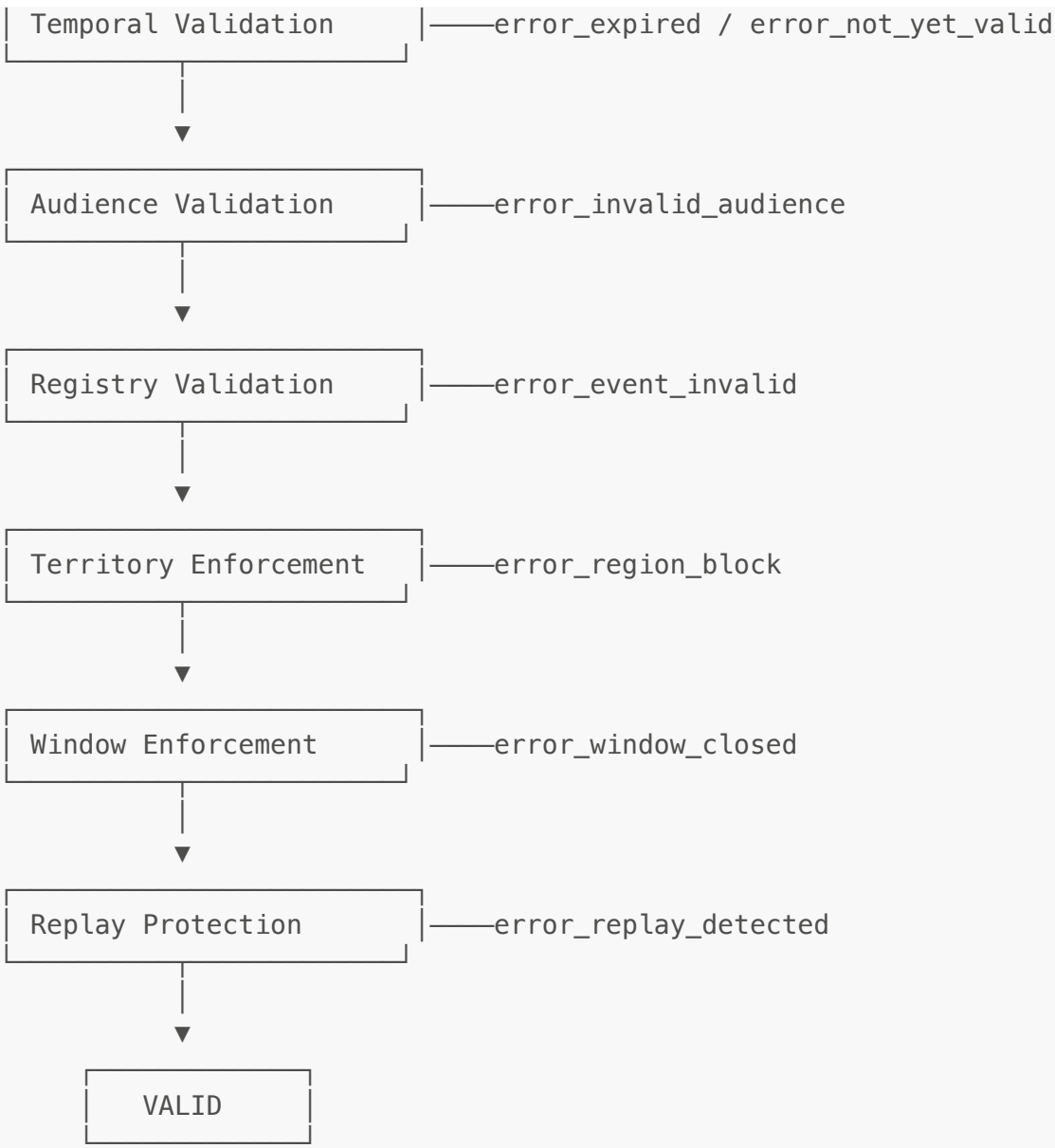
```
    # --- 2. Signature & Trust ---
    kid = header.get("kid")
    if kid is None:
        return error("error_missing_claim")

    key = key_registry.get_public_key(kid)
    if key is None:
        return error("error_untrusted_issuer")

    if not verify_signature(token, key, header.get("alg")):
        return error("error_invalid_signature")

    # --- 3. Required Claims ---
    for field in REQUIRED_CLAIMS:
        if field not in payload:
            return error("error_missing_claim")

    # --- 4. Type & Format Validation ---
    if not isinstance(payload["territory"], list):
        return error("error_invalid_claim")
    if not isinstance(payload["scope"], list):
        return error("error_invalid_claim")
    if "view" not in payload["scope"]:
        return error("error_scope_denied")

    # --- 5. Temporal Validation ---
    if payload["exp"] < now_utc:
        return error("error_expired")
    if payload["nbf"] > now_utc:
        return error("error_not_yet_valid")
    if payload["iat"] > payload["exp"]:
        return error("error_invalid_claim")

    # --- 6. Audience ---
    if verifier_id not in payload["aud"]:
        return error("error_invalid_audience")

    # --- 7. Registry ---
    event = registry.lookup(payload["event_id"])
    if event is None:
        return error("error_event_invalid")

    # --- 8. Territory ---
    if user_region not in payload["territory"]:
        return error("error_region_block")

    # Registry territory consistency
    if not set(payload["territory"]).issubset(set(event.territories)):
        return error("error_region_block")

    # --- 9. Access Window ---
    window = payload.get("window")
    window_end = timestamp(window["end"])
```

```
    window_start = timestamp(window["start"])

    if now_utc < window_start or now_utc > window_end:
        return error("error_window_closed")

    if payload["exp"] > window_end:
        return error("error_invalid_claim")

    # --- 10. Replay Protection ---
    hashed_jti = hash(payload["jti"])
    if replay_cache.seen(hashed_jti):
        return error("error_replay_detected")

    replay_cache.add(hashed_jti, expiry=payload["exp"])

    # --- 11. Final Authorization ---
    return success("valid")
```

B.3 Validation State Machine (Abstract)

Below is the state machine representation of the validation pipeline.





B.4 Failure Mode Prioritization

Verifiers SHOULD return errors in the following priority order to minimize attack ambiguity:

1. **Structural errors**
2. **Signature / trust errors**
3. **Missing claims**
4. **Invalid claims**
5. **Temporal errors**
6. **Audience mismatch**
7. **Registry mismatch**
8. **Territory mismatch**
9. **Scope errors**
10. **Replay detection**

This ensures both implementer clarity and reduced exploit surface.

B.5 Recommended Data Structures

Replay Cache

- hashed-jti → expiry timestamp
- eviction: time-based, O(1) lookup

Key Registry

- kid → public key
- key rotation metadata
- issuer identity bindings

Registry Lookup Table

- event_id → canonical event metadata
- includes territory, window, and rights constraints

B.6 Operational Notes for Production Environments

- Verifiers SHOULD monitor sudden spikes in specific error codes.
- Clearinghouse SHOULD track anomalous issuance or replay behavior.
- Platforms SHOULD tightly restrict access to token introspection tools.
- Token generation SHOULD be rate-limited to prevent flood attacks.
- Clock synchronization MUST be maintained across infrastructure.

---

14. References

14.1 Normative References

These documents are REQUIRED to implement or reason about this specification.

- **RFC 7519 — JSON Web Token (JWT)**
  Jones, M., Bradley, J., Sakimura, N. (IETF, 2015)
- **RFC 7515 — JSON Web Signature (JWS)**
  Jones, M., Bradley, J., Sakimura, N. (IETF, 2015)
- **RFC 2119 — Key words for use in RFCs to Indicate Requirement Levels**
  Bradner, S. (IETF, 1997)
- **RFC 0001 — The Universal Sports Graph**
  Jellen, S. (2025). DOI: 10.5281/zenodo.17565794

14.2 Informative References

These documents provide supporting context but are **not required** for implementation.

- **RFC 3986 — Uniform Resource Identifier (URI): Generic Syntax**
  Berners-Lee, T., Fielding, R., Masinter, L. (IETF, 2005)

Revision History

v1.1 — December 2025

- Removed placeholder ISSN from document metadata.
- Updated header to adopt the unified USG-RFC series format.
- Aligned top matter with RFC 0001 revisions for consistency.
- No technical or normative changes were made to the specification.

v1.0 — December 2025

- Initial publication of RFC 0002 (USG Entitlement Token Profile).
- Introduced the normative token schema for authorization within the USG ecosystem.
- Defined required and optional claims for entitlement tokens.
- Established issuer and verifier rules for signature handling, key lifecycle management, and replay protection.
- Formalized enforcement rules for territory, window, audience, and scope.
- Added canonical error codes, failure modes, and prioritization guidance.
- Included full reference verifier pipeline, pseudocode, and validation state machine.
- Added provisional namespace definitions (URNs) for token and schema identifiers.

End of RFC 0002