Independent Submission
Internet-Draft
Intended status: Informational
Expires: 26 November 2026

V. Research
Vauban Research

25 May 2026

VPSF Claim Algebra for x402 Payment Receipts
draft-vauban-x402-vpsf-algebra-01

Abstract

The x402 protocol ([X402-V2]) defines three wire messages for HTTP-native payment flows but provides no composability model for payment receipts. A recipient holding a SettlementReceipt and a DelegationGrant cannot natively express that the settlement satisfies a delegated spending condition, or that a RefundClaim negates an earlier SettlementReceipt, without bespoke facilitator-side logic. As automated payment pipelines grow, the absence of a normative composability layer leads to fragmented, non-interoperable verification surfaces.

This document specifies the Vauban Proof Stack Framework (VPSF) Claim Algebra for x402 payment receipts: a grammar of five composition operators (Conjunction, Implication, Aggregation, Selective Disclosure, Revocation; abbreviated $\wedge$, $\rightarrow$, , , $\neg$) applied over the four canonical Payment Claim types (PaymentIntent, SettlementReceipt, RefundClaim, DelegationGrant) defined in [LIFECYCLE-FSM]. The algebra is chain-agnostic by invariant; the Starknet-based proof system described in [STARK-RECEIPTS] is the first reference implementation. Composability is grounded in the JCS canonical preimage discipline ([RFC8785]), ensuring operator results are deterministic and cross-implementation consistent. An open-source Rust implementation is provided in [VAUBAN-CRATE].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 November 2026.

Table of Contents

1. Introduction

The x402 protocol ([X402-V2]) is an HTTP-native payment standard that enables machine-to-machine and human-to-server payment flows using three protocol messages: PAYMENT-REQUIRED (402 response), PAYMENT-SIGNATURE (client request), and PAYMENT-RESPONSE (facilitator confirmation). The companion specification [LIFECYCLE-FSM] introduces four canonical Payment Claim types that map to the payment lifecycle states, and [STARK-RECEIPTS] defines a cryptographic receipt extension with offline-verifiable STARK proofs. Together these three documents describe what a payment is and what happens to it over time; they do not describe how payment receipts may be composed into higher-order conditions.

Automated payment pipelines present composition requirements that the base protocol does not address. An agentic delegatee may be authorised to spend up to a bounded amount only after a supervisory SettlementReceipt has been confirmed; a compliance engine may require that a SettlementReceipt is valid and its originating DelegationGrant is still in scope; a refund issuer may need to attest that the negated SettlementReceipt and the replacement RefundClaim satisfy the same subject constraint. Expressing any of these conditions today requires out-of-band logic that is neither portable nor auditable across implementations.

This document defines the VPSF Claim Algebra: five composition operators over x402 Payment Claims. The algebra is chain-agnostic by invariant; the design does not depend on any specific proof system, settlement layer, or on-chain infrastructure. Starknet is the first reference implementation (see [STARK-RECEIPTS]), but the operator semantics and preimage discipline are equally applicable to any receipt format that satisfies the JCS canonical preimage rules defined in [LIFECYCLE-FSM]. The algebra is formalised as part of the Vauban Proof Stack Framework (VPSF), an open-specification claim grammar that targets institutional-grade composability across payment, compliance, and identity contexts.

This document is an Independent Submission. It is not the product of an IETF Working Group. It is published for community review and to establish a stable reference for implementors working with x402 receipt composability.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Terminology

Claim: A cryptographically bound, structured statement about a payment event or authority. In this document, a Claim is always an instance of one of the four Payment Claim types defined in [LIFECYCLE-FSM]: PaymentIntent, SettlementReceipt, RefundClaim, or DelegationGrant. The term is used interchangeably with "Payment Claim".

Subject: The identity field that anchors a Claim to a specific

principal. For PaymentIntent and SettlementReceipt, the subject is the payer address or pseudonym. For DelegationGrant, the subject is the delegator. For RefundClaim, the subject is the merchant. Two Claims have the same subject when their subject fields are byte-identical after NFC normalization.

Claim Composition: The application of one of the five VPSF algebra operators to two or more Claims, producing a composite Claim or a composite verification outcome. Composition does not alter the component Claims; it creates a new logical statement over them.

JCS Preimage Hash: As defined in [LIFECYCLE-FSM]: the SHA-256 digest of the UTF-8 encoding of the JCS canonical form ([RFC8785]) of a Claim’s canonical preimage object, encoded as "sha256:<lowercase-hex-64-chars>" in JSON contexts.

Composite Preimage: The JCS canonical preimage object of a composed Claim. For binary operators (Conjunction, Implication, Aggregation, Revocation), the composite preimage contains the JCS Preimage Hashes of the component Claims and the operator identifier. For selective disclosure (), the composite preimage contains the disclosed fields and the preimage hash of the undisclosed source Claim.

Operator: One of the five composition operators defined in this document: Conjunction (∧), Implication (→), Aggregation (), Selective Disclosure (), Revocation (¬).

Chain-Agnostic Invariant: The property that the VPSF Claim Algebra operators and the JCS preimage discipline do not depend on any specific blockchain, proof system, or settlement layer. Any receipt format that implements the canonical preimage discipline defined in [LIFECYCLE-FSM] can serve as the substrate for VPSF composition.

Reference Implementation: An open-source implementation whose behaviour is normative for resolving specification ambiguities. The Rust crate [VAUBAN-CRATE] is the reference implementation for this document. Its Starknet adapter uses the Stwo Circle STARK prover ([STWO]) as the proof backend; this is the first reference implementation of the chain-agnostic algebra on a specific settlement layer, and is not part of the algebra specification itself.

3. VPSF Overview

3.1. Grammar Layer

The Vauban Proof Stack Framework (VPSF) is a claim-algebraic grammar layer designed to sit above receipt-format and settlement-layer specifics. It provides a composable vocabulary for expressing higher-order conditions over Payment Claims without encoding those conditions in facilitator-specific logic or on-chain contract state.

The grammar operates on a six-tuple structure for each Claim: (subject, predicate, object, proof, context, metadata). In the x402 setting, these fields map as follows:

* subject: the principal whose payment or authority the Claim describes.

* predicate: the lifecycle state (PaymentIntent, SettlementReceipt, RefundClaim, DelegationGrant) or a derived predicate produced by composition.

* object: the payment target (payee, amount, currency) or authority scope.

* proof: the cryptographic evidence that the Claim holds (a STARK proof, a signature, or a composite preimage hash over component proofs).

* context: the temporal and scope constraints under which the Claim is valid (expiry window, nullifier, chain identifier of the reference implementation).

* metadata: operational fields (schema version, issuer, audit reference).

The six-tuple is the abstract schema; concrete instances are the four Payment Claim types defined in [LIFECYCLE-FSM]. The operators defined in this document are applied at the grammar layer; they are neutral with respect to the proof field, provided the proof field satisfies the JCS canonical preimage discipline defined in [LIFECYCLE-FSM].

3.2. Four Payment Claim Types

The four Payment Claim types that serve as the atomic units of VPSF composition in the x402 context are:

PaymentIntent: The initiating state. Records the payer’s commitment to transfer a specified amount in a specified currency to a specified payee. Subject: payer.

SettlementReceipt: The terminal successful state. Records that all payment conditions were verified and the transfer was settled. Subject: payer. Carries a cryptographic linkage to the originating PaymentIntent via its JCS Preimage Hash.

RefundClaim: The reversal state. Records a merchant-initiated cancellation of a prior SettlementReceipt. Subject: merchant (issuer of the refund). Carries a cryptographic linkage to the SettlementReceipt being reversed.

DelegationGrant: The bounded-authority side channel. Records that a principal (delegator) has authorised an agent (delegatee) to spawn PaymentIntents within a scoped constraint set. Subject: delegator.

The normative definitions of these types, including required fields, JCS preimage rules, and lifecycle transitions, are in [LIFECYCLE-FSM]. This document uses them as atoms for composition; it does not modify their definitions.

3.3. Chain-Agnostic Rule

The VPSF Claim Algebra MUST NOT be bound to any specific chain, proof system, or settlement layer. Implementations that restrict operator semantics to a single proof system MUST document that restriction as an implementation constraint, not as a normative constraint of the algebra.

The chain-agnostic invariant has two corollaries:

1. An operator applied to two Claims whose proofs use different proof systems (for example, a STARK-proven SettlementReceipt composed under Conjunction with an ES256K-signed DelegationGrant) produces a well-formed composite Claim. The composite proof field contains the JCS Preimage Hashes of the component Claims; it does not require proof-system unification.

2. A composite Claim produced under the VPSF algebra is portable across settlement layers. A verifier on one chain MAY verify the composite preimage discipline without access to the proof artifacts of the component Claims on another chain, provided the component Claims' JCS Preimage Hashes are included in the composite preimage.

4. Five Composition Operators

4.1. Conjunction (∧)

4.1.1. Semantic

The Conjunction operator (∧) produces a composite Claim that is valid if and only if both component Claims are individually valid. This corresponds to the logical AND of two payment conditions.

Typical use: a compliance engine requires BOTH that a SettlementReceipt is verified AND that the originating DelegationGrant is still in scope and non-expired. Neither condition alone is sufficient; both must hold simultaneously.

4.1.2. Algebraic Properties

* Commutativity: (A ∧ B) and (B ∧ A) produce composites with the same validity outcome, but DIFFERENT JCS Preimage Hashes (key ordering is lexicographic and the field names encode left-right position). Implementations MUST NOT treat these as the same composite.

* Associativity: ((A ∧ B) ∧ C) produces a valid composite if and only if each of A, B, C is individually valid. The associativity holds at the validity level; the preimage hashes differ across groupings.

* Identity element: there is no identity claim for Conjunction; the operator requires exactly two non-null operands.

4.1.3. JCS Preimage Rule

The composite preimage object for A ∧ B is:

```
{
  "operator": "conjunction",
  "left":  "<JCS Preimage Hash of A>",
  "right": "<JCS Preimage Hash of B>",
  "issued_at": <integer Unix timestamp>
}
```

Keys are sorted lexicographically per [RFC8785]. The composite JCS Preimage Hash is SHA-256(UTF-8(JCS(composite_preimage))). The issued_at field MUST reflect the time at which the composition was performed; it binds the composite to a specific evaluation instant, preventing replay of stale compositions after component Claim expiry.

A verifier validating a Conjunction composite MUST:

1. Resolve the JCS Preimage Hash for each component Claim against retained manifests.

2. Verify that each resolved Claim is individually valid (signature or proof check, expiry check, FSM state check per [LIFECYCLE-FSM]).

3. Verify that the composite preimage object is well-formed and that its hashes match the resolved component Claims.

If any check fails, the composite MUST be rejected.

4.2. Implication (→)

4.2.1. Semantic

The Implication operator (→) produces a composite Claim asserting that the valid presence of an antecedent Claim is a precondition for the consequent Claim to be considered authoritative. This corresponds to conditional authority: Claim B is asserted only because Claim A holds.

Typical use: a DelegationGrant (A) implies that a PaymentIntent (B) was spawned under valid authority. The Implication composite expresses the dependency without merging the two Claims into a single object; each retains its individual verifiability.

4.2.2. Algebraic Properties

* Non-commutative: (A → B) is not equivalent to (B → A). The antecedent and consequent positions are semantically distinct.

* Vacuous truth is FORBIDDEN: an Implication composite where the antecedent Claim is invalid or absent MUST be rejected. Implementations MUST NOT interpret "if A is absent then B holds unconditionally".

* Transitivity: (A → B) and (B → C) can be chained to express (A → B → C), but MUST be expressed as two distinct Implication composites, not collapsed into a single three-operand composite.

4.2.3. JCS Preimage Rule

The composite preimage object for A → B is:

```
{
  "operator": "implication",
  "antecedent": "<JCS Preimage Hash of A>",
  "consequent": "<JCS Preimage Hash of B>",
  "issued_at": <integer Unix timestamp>
}
```

A verifier validating an Implication composite MUST:

1. Resolve and verify the antecedent Claim (A). If A is invalid, reject the composite without evaluating B.

2. Resolve and verify the consequent Claim (B).

3. Verify the composite preimage well-formedness.

The Implication composite does not replace the antecedent's own validity period; if the DelegationGrant (A) has expired, the Implication is invalid even if the PaymentIntent (B) was spawned before expiry.

4.3. Aggregation ()

4.3.1. Semantic

The Aggregation operator () combines multiple Claims of the same type into a summary Claim whose total satisfies a threshold condition. This corresponds to the sum-over-intents pattern needed for multi-payment settlement verification.

Typical use: an agentic pipeline spawns three PaymentIntents under a DelegationGrant scoped to a maximum total of 500 USDC. The Aggregation composite over the three PaymentIntents allows a supervisor to verify that the total does not exceed the grant's budget, without examining each PaymentIntent independently against the grant's scope field.

4.3.2. Algebraic Properties

* Commutative: (A B) and (B A) have the same validity outcome (same total). The composite preimage hashes DIFFER (field ordering); implementations MUST NOT treat them as equivalent.

* Associative at the value level: ((A B) C) and (A (B C)) produce the same total amount, but different composite preimage hashes. Verifiers MUST validate the preimage structure as presented; they MUST NOT regroup operands.

* Homogeneity requirement: all operand Claims MUST be of the same Payment Claim type and MUST use the same currency field. Aggregating Claims with different currencies MUST be rejected with a type-mismatch error.

* Subject alignment: all operand Claims SHOULD share the same subject. An implementation MAY enforce strict subject equality or MAY permit multi-subject aggregation with explicit disclosure of the distinct subjects in the composite preimage. See Section 5 for the default discipline.

4.3.3. JCS Preimage Rule

The composite preimage object for A B ... N is:

```
{
  "operator": "aggregation",
  "operands": [
    "<JCS Preimage Hash of A>",
    "<JCS Preimage Hash of B>",
    "<JCS Preimage Hash of N>"
  ],
  "currency": "<shared currency token>",
  "total_amount": <sum of operand amounts as integer>,
  "operand_count": <integer>,
  "issued_at": <integer Unix timestamp>
}
```

The operands array MUST be sorted lexicographically by the JCS Preimage Hash values before serialisation. This ensures a canonical ordering regardless of the order in which component Claims were produced. The total_amount MUST equal the arithmetic sum of the amount fields of all operand Claims; a verifier MUST recompute this sum and reject the composite if it disagrees.

4.4. Selective Disclosure ()

4.4.1. Semantic

The Selective Disclosure operator () produces a derived Claim in which a subset of the source Claim’s fields are disclosed while the remainder are withheld. The composite carries a cryptographic commitment to the undisclosed fields, allowing a verifier to confirm that the disclosed fields are authentic fragments of the source Claim without learning the withheld values.

Typical use: a payer discloses the amount and currency fields of a PaymentIntent to a compliance auditor without disclosing the payer pseudonym or the nonce. The auditor can verify that the disclosed fields are genuine without learning the payer’s identity.

4.4.2. Algebraic Properties

* Non-invertible: a selective disclosure composite cannot be used to reconstruct the undisclosed fields. The commitment in the composite preimage is a one-way function of the withheld fields.

* Composable: a selective disclosure composite can itself be the operand of a Conjunction or Implication. A verifier receiving a Conjunction of two selective disclosure composites validates each independently.

* Minimality: implementations SHOULD disclose the minimum set of fields required for the verifier's purpose. Disclosing more fields than necessary weakens the privacy guarantee without improving composability.

4.4.3. JCS Preimage Rule

The composite preimage object for (A {f1, f2, ...}) is:

```
{
  "operator": "selective_disclosure",
  "source_hash": "<JCS Preimage Hash of source Claim A>",
  "disclosed_fields": {
    "<field_name_1>": <value_1>,
    "<field_name_2>": <value_2>
  },
  "withheld_commitment": "<sha256-hex-of-withheld-jcs>",
  "issued_at": <integer Unix timestamp>
}
```

The disclosed_fields object MUST contain only fields that appear in the source Claim's canonical preimage. The withheld_commitment MUST be derived by applying JCS canonicalization to a JSON object containing only the withheld fields (those present in the source Claim canonical preimage but absent from disclosed_fields), then taking the SHA-256 of the UTF-8 encoding of the result.

A verifier validating a selective disclosure composite MUST:

1. Resolve the source Claim by its source_hash.

2. Recompute the withheld_commitment from the source Claim's canonical preimage minus the disclosed fields.

3. Verify that the recomputed commitment equals the withheld_commitment in the composite.

4. Apply any validity checks (expiry, proof) to the disclosed fields.

If any check fails, the composite MUST be rejected. A verifier that cannot resolve the source Claim by its source_hash MUST treat the composite as unverifiable and MUST NOT accept it.

4.5. Revocation (¬)

4.5.1. Semantic

The Revocation operator (¬) produces a composite Claim asserting that a source Claim is no longer valid. This operator is the algebraic counterpart of the RefundClaim lifecycle transition: whereas the FSM records that a SettlementReceipt has been reversed, the Revocation operator expresses the negation at the grammar layer, making it composable with other operators.

Typical use: a compliance engine produces a Revocation composite over a DelegationGrant that has been administratively withdrawn, and then presents a Conjunction of the Revocation composite and a new DelegationGrant to attest that the old authority has been replaced by the new one.

4.5.2. Algebraic Properties

* Non-idempotent: applying Revocation twice (¬(¬A)) does NOT restore A to validity. A doubly-revoked Claim is invalid. Implementations MUST NOT interpret a Revocation of a Revocation as a restoration.

* Non-composable with itself: a Revocation composite MUST NOT be used as the operand of another Revocation. Implementations MUST reject such nesting.

* Interaction with Implication: if the antecedent of an Implication is revoked, the Implication composite becomes invalid. Verifiers MUST check for Revocation of each component Claim before accepting a composite that depends on it.

4.5.3. JCS Preimage Rule

The composite preimage object for ¬A is:

```
{
  "operator": "revocation",
  "target_hash": "<JCS Preimage Hash of source Claim A>",
  "reason": "<human-readable or machine-readable reason string>",
  "revoked_at": <integer Unix timestamp>,
  "issued_at": <integer Unix timestamp>
}
```

The revoked_at MUST be less than or equal to issued_at. A Revocation composite whose revoked_at is in the future MUST be rejected.

A verifier checking whether a Claim is revoked MUST query a revocation registry or the retained manifest store for any Revocation composite whose target_hash matches the Claim's JCS Preimage Hash. If such a composite exists and is itself valid (well-formed preimage, issued by an authorised revoker), the source Claim MUST be treated as invalid regardless of its own expiry or proof validity.

The identity of the authorised revoker is out of scope for this document; receipt formats and deployments MUST specify the revoker identity and authority chain in their operational documentation.

5. Same-Subject Preimage Discipline

5.1. Generic Invariant

The VPSF Claim Algebra imposes a subject alignment requirement on binary operators (Conjunction, Implication, Revocation) by default: the two operand Claims MUST share the same subject value (as defined in Section 2.1) unless the operator invocation explicitly declares a subject override.

The rationale for this default is that composing Claims from different principals without explicit subject disclosure creates composites whose validity surface is larger than intended. A Conjunction of a SettlementReceipt from principal A with a DelegationGrant from principal B is semantically ambiguous unless the composite explicitly binds the two subjects.

5.2. Strict vs Relaxed Enforcement

Implementations MAY enforce subject alignment in one of two modes:

Strict mode: The implementation MUST reject any operator invocation where the operand Claims do not share the same subject value. No subject override is permitted. This is the RECOMMENDED mode for compliance-critical deployments.

Relaxed mode: The implementation MAY accept operator invocations over Claims with different subject values, provided the composite preimage includes a subject_disclosure field that explicitly lists all distinct subjects present in the composition:

```
{
  "operator": "conjunction",
  "left": "<JCS Preimage Hash of A>",
  "right": "<JCS Preimage Hash of B>",
  "subject_disclosure": {
    "left_subject": "<subject of A>",
    "right_subject": "<subject of B>"
  },
  "issued_at": <integer Unix timestamp>
}
```

A verifier receiving a composite with subject_disclosure MUST verify that the declared subjects match the resolved operand Claims. A composite that omits subject_disclosure when the operand subjects differ MUST be rejected in both strict and relaxed mode.

5.3. Aggregation Subject Handling

For Aggregation (), a homogeneous multi-principal aggregate (where all operands share the same subject) omits subject_disclosure. A heterogeneous aggregate (operands from different principals) MUST include a subjects array in the composite preimage:

```
{
  "operator": "aggregation",
  "operands": ["<hash1>", "<hash2>"],
  "currency": "USDC",
  "total_amount": 500,
  "operand_count": 2,
  "subjects": ["<subject of hash1>", "<subject of hash2>"],
  "issued_at": <integer Unix timestamp>
}
```

The subjects array MUST be ordered to match the operands array. Implementors are RECOMMENDED to avoid heterogeneous aggregation when a strict compliance context requires single-principal attestation.

6. Compatibility with Receipt Formats and FSM

6.1. Relationship to Lifecycle FSM

The VPSF Claim Algebra operates at a layer above the Lifecycle FSM defined in [LIFECYCLE-FSM]. The FSM defines how Payment Claims transition between states; the algebra defines how Claims in any state may be composed into higher-order conditions. The two specifications are complementary and MUST NOT be conflated.

Specifically:

* The Revocation operator (¬) at the algebra layer and the RefundClaim lifecycle state at the FSM layer address different concerns. RefundClaim is a first-class lifecycle transition that produces a new Payment Claim recorded by the facilitator. Revocation is a grammar-layer assertion that an existing Claim is no longer authoritative, without necessarily producing a new FSM state. Implementations that map RefundClaims to Revocation composites MUST document the mapping explicitly.

* The Implication operator (→) and the FSM bounded-authority transition from DelegationGrant to PaymentIntent are related but distinct. The FSM transition records that a PaymentIntent was spawned under a DelegationGrant; the Implication composite expresses this relationship as a verifiable grammar statement. Both SHOULD be present in a conformant implementation for full auditability.

Any receipt format that implements the JCS canonical preimage discipline defined in [LIFECYCLE-FSM] is eligible to serve as the substrate for VPSF composition. The algebra does not impose additional requirements on the proof field of component Claims beyond those of the FSM.

6.2. Relationship to STARK Receipt Format

The x402 STARK Receipt Format Extension [STARK-RECEIPTS] is the first reference implementation of VPSF composition on a specific settlement layer (Starknet, using the Stwo Circle STARK prover [STWO]). The chain-agnostic invariant (see Section 3.3) ensures that the operator semantics in this document are not specific to that implementation.

Implementations based on [STARK-RECEIPTS] MAY use STARK proofs as the proof field in composite Claims. In this case, the composite preimage hash and the STARK proof over it constitute a doubly-grounded composite: the preimage discipline provides chain-agnostic portability; the STARK proof provides post-quantum soundness. Neither property is required by this document for conformance; both are OPTIONAL enhancements. Conformance vectors for the reference Starknet adapter are published in [VAUBAN-STARK-GIST].

6.3. Operator Precedence

When multiple operators are applied in a single expression, implementations MUST respect the following precedence order (highest to lowest):

1. Selective Disclosure (): applied innermost; modifies a single Claim before it is used as an operand.

2. Revocation (¬): applied next; marks a Claim as revoked before composition.

3. Conjunction (∧): evaluated before Implication.

4. Aggregation (): evaluated before Implication.

5. Implication (→): evaluated last; the antecedent is always a fully composed expression.

Implementations MAY override this order by using explicit parenthesisation in their serialised composite preimage structure. A composite preimage that encodes an Implication as an operand of a Conjunction is unambiguous regardless of default precedence.

7. Security Considerations

7.1. Preimage Collision and Second-Preimage Resistance

The JCS Preimage Hash is computed as SHA-256 over the UTF-8 encoding of the JCS canonical form of the composite preimage object. The security of the composition chain depends on the second-preimage resistance of SHA-256. An adversary who can construct a different composite preimage object with the same hash could substitute a fraudulent composite for a genuine one.

Mitigation: SHA-256 provides 128-bit second-preimage resistance, which is considered adequate for payment receipt composability as of 2026. Implementations operating in environments where higher resistance is required (for example, post-quantum threat models)

SHOULD use SHA-3-256 or SHAKE-256 as an alternative hash function, and MUST document the substitution in the composite preimage with an explicit hash_alg field.

7.2. Subject Confusion Attacks

A subject confusion attack occurs when an adversary presents a composite Claim whose operands belong to different principals, without declaring the subject split via subject_disclosure. The verifier, not detecting the mismatch, treats the composite as a single-principal attestation.

Mitigation: the subject alignment requirement in Section 5 is the primary defence. Verifiers MUST implement subject field resolution and comparison for all binary operators. Deployments in strict compliance contexts SHOULD use strict mode to prevent relaxed-mode abuse.

7.3. Replay of Composite Preimages

A composite preimage hash is deterministic: the same set of component Claims, the same operator, and the same issued_at timestamp always produce the same composite hash. An adversary who obtains a valid composite preimage hash for an expired Claim can replay it if the verifier does not check expiry on the component Claims.

Mitigation: verifiers MUST resolve and validate each component Claim individually, including expiry checks, before accepting a composite. The issued_at field in the composite preimage binds the composition to a specific instant; verifiers MAY impose a maximum age on composite preimages (RECOMMENDED: 30 seconds for real-time payment flows, 30 days for audit-trail composites).

7.4. Revocation Race Conditions

Between the instant a Revocation composite is issued and the instant it propagates to all verifiers, the revoked Claim may be accepted by a verifier that has not yet received the Revocation. This window creates a race condition exploitable by an adversary who obtains a revoked DelegationGrant and uses it to spawn PaymentIntents before the Revocation propagates.

Mitigation: implementations SHOULD use a revocation registry with a propagation SLA shorter than the minimum payment processing time in the deployment. For high-frequency payment pipelines, the RECOMMENDED approach is to embed a revocation epoch counter in each DelegationGrant and require that PaymentIntents spawned under the grant carry a reference to the epoch at which the grant was last confirmed valid. Facilitators MUST reject PaymentIntents referencing an epoch older than the current revocation epoch.

7.5. Selective Disclosure Linkability

When selective disclosure composites are produced from the same source Claim but with different field subsets, the source_hash field in each composite is identical. An observer holding two selective disclosure composites for the same source Claim can determine that they derive from the same Claim, even without knowing the withheld fields.

Mitigation: implementations for which source Claim linkability is a privacy concern SHOULD use a claim-level nullifier or blinding factor derived from the source Claim’s canonical preimage. The blinding scheme is out of scope for this document; receipt formats implementing unlinkable selective disclosure MUST document their mechanism.

8. IANA Considerations

This document has no IANA actions. The composition operators defined in this document are identified by string tokens within the VPSF composite preimage schema and do not require IANA registration. Extensions that introduce additional operators SHOULD publish an Internet-Draft registering the new operator tokens in a to-be-defined VPSF Operator Registry; this document does not create that registry.

9. Acknowledgments

The VPSF Claim Algebra specification builds on discussions within the x402 Linux Foundation working group. The composition requirements addressed in this document emerged from coalition activity on the x402 STARK Receipts Extension Proposal ([X402-2357]). The authors thank the participants in the x402 coalition for the shared canonicalisation discipline ([RFC8785]-based JCS preimage alignment) that makes the algebra’s preimage rules portable across implementations.

Known Adopters and Reference Implementations

This appendix documents reference implementations and adopters of this specification confirmed at the time of publication. The list is informational and will be updated in subsequent revisions as additional implementations are reported.

Primary Maintainer

Vauban Pay (https://pay.vauban.tech) maintains the reference VPSF Claim Algebra specification, the published conformance vectors (https://github.com/vauban-org/x402-stark-receipts-conformance), and the reference implementations listed below. The five algebraic operators (conjunction, implication, aggregation, selective disclosure, revocation) are exercised across the published conformance vector suite.

Reference Implementation Matrix

The conformance vector suite maintains an 8-implementation reference matrix : Python (rfc8785@0.1.4 by Trail of Bits), JavaScript (canonicalize@3.0.0 by Erdtman and Rundgren), Go (gowebpki/jcs v1.0.1), Java (cyberphone/json-canonicalization RFC 8785 reference), Rust (serde_jcs 0.2.0 via vauban-x402-jcs-conformance), PHP/Ruby/C# (pure-stdlib reference runners). The first five are validated byte-for-byte ; the last three are published as reference runners pending CI execution. Detailed validation status is documented in _attestations/2026-05-25-vauban-8-impl-extended.md in the conformance vectors repository.

The published Vauban Pay packages across 3 ecosystems : vauban-x402-jcs-conformance@0.1.0, vauban-x402-canonical@0.1.0, vauban-x402-wire@0.1.0 on crates.io ; vauban-x402-stark-receipt@0.1.0 on PyPI ; @vauban-pay/substrate@0.1.0 on npm.

Adoption Process

Implementers SHOULD notify the IETF contact at research@vauban.tech when adopting this specification in production. Adoption notifications include the production endpoint or product identifier, the VPSF operators implemented, the same-subject invariant enforcement strategy, and the contact for follow-on coordination. Reported adopters will be listed in the next revision of this appendix following a verification step against the conformance vector matrix.

References

Normative References

[LIFECYCLE-FSM]
"x402 V2 Payment Lifecycle Finite State Machine", Work in Progress, Internet-Draft, draft-vauban-x402-lifecycle-fsm-00, n.d., <https://datatracker.ietf.org/doc/draft-vauban-x402-lifecycle-fsm/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[STARK-RECEIPTS]
"x402 STARK Receipt Format Extension", Work in Progress, Internet-Draft, draft-vauban-x402-stark-receipts-01, n.d., <https://datatracker.ietf.org/doc/draft-vauban-x402-stark-receipts/>.

[X402-V2] "x402 Linux Foundation V2 Working Group", n.d., <https://github.com/x402-foundation/x402>.

Informative References

[STWO] "Stwo Circle STARK Prover (StarkWare Industries)", n.d., <https://github.com/starkware-libs/stwo>.

[VAUBAN-CRATE]
"vauban-x402-canonical: reference Rust implementation of VPSF Claim Algebra for x402", n.d., <https://github.com/vauban-org/vauban-zkpay>.

[VAUBAN-STARK-GIST]
"Vauban STARK receipt conformance vectors (public)", n.d., <https://github.com/vauban-org/x402-stark-receipts-conformance>.

[X402-2357]
"x402 STARK Receipts Extension Proposal", n.d.,
<https://github.com/x402-foundation/x402/issues/2357>.

Appendix A. References

A.1. Normative References

(Normative references are listed in the document header.)

A.2. Informative References

(Informative references are listed in the document header.)

Author's Address

Vauban Research
Vauban Research
Email: research@vauban.tech
URI: https://pay.vauban.tech