YONA Ruleset 1.0

Ruleset ID: `yona:ruleset:v1.0`

Status: Active

Published: April 5, 2026

Companion Documents:

- *Base Interoperability Profile (BIP)*
- *BIP Conformance Test Suite*
- *BIP Conformance Report Template* *(optional, non-normative)*

Table of Contents

License

Overview

YONA gives one Virtual Asset Service Provider (VASP) a standard way to ask another VASP a simple question before regulated data is disclosed and before settlement begins:

Can we move forward with this payment on your side?

In Ruleset 1.0, that question begins from a valid beneficiary-issued YONA reference, not from a raw blockchain address entered by the user. For pull-payments, that reference is an `intent_locator`. For push-payments, it is a `beneficiary_handle`. Those references can be presented through merchant checkout, a payment link, a QR code, or another presentation method outside this ruleset.

That question is answered before any:

- disclosure of personally identifiable information (PII)
- Travel Rule data exchange
- settlement-instruction exchange
- settlement submission

Simply put, YONA is a pre-disclosure, pre-settlement authorization layer for interVASP payments.

For each authorization request, the beneficiary VASP returns one of only two terminal decisions:

- `ACCEPT` — the parties may proceed to post-`ACCEPT` steps outside YONA, such as Travel Rule exchange and later settlement, using their existing providers, channels, and local policy
- `REJECT` — the flow **MUST NOT** proceed

YONA defines no intermediate states.

YONA is not a runtime network. It does not move funds or carry PII, and it does not define Travel Rule payloads, PII exchange procedures, settlement execution, custody, settlement-routing logic, or address-only payment entry.

How to Read This Document

The key words **MUST**, **MUST NOT**, **SHOULD**, and **MAY** are to be interpreted as described in **BCP 14** (RFC 2119 and RFC 8174) when, and only when, they appear in all capitals.

Ruleset 1.0 uses signed messages. “JWS” refers to JSON Web Signature in **RFC 7515**. When used, “JWS Compact Serialization” means the compact serialization defined by RFC 7515.

YONA Ruleset 1.0 is the normative source for YONA authorization semantics, message validity, repeated-request behavior, request/response binding, and message-field requirements.

For `yona:ruleset:v1.0`, the normative documents are:

1. this Ruleset 1.0;
2. the companion *Base Interoperability Profile (BIP)*; and
3. the applicable *BIP Conformance Test Suite*.

The companion *Base Interoperability Profile (BIP)* defines the interoperable discovery, transport, and endpoint profile used for BIP conformance. The *BIP Conformance Test Suite* defines how BIP conformance is verified.

Non-normative implementation guidance and examples appear in separately published YONA materials, such as *Implementation Notes and Reference Messages and DID Examples*. The *BIP Conformance Report Template* is optional and non-normative.

The conformance requirements for `yona:ruleset:v1.0` are defined in Section 13.

---

1. Problem Scope

1.1 What problem YONA solves

Existing standards and regulations define *what* information must accompany virtual-asset transfers and *when* that information must be exchanged.

However, they generally do not define a clear authorization step before any regulated data is disclosed and before any settlement submission.

That missing pre-disclosure authorization boundary creates recurring risk and friction:

1. **Counterparty identification is inconsistent and unreliable.** If a sender has only a destination address, it may be impossible to determine with confidence whether the wallet is VASP-managed, which VASP manages it, or which endpoint, key set, or local policy context applies. This problem is compounded when VASPs operate through multiple entities, environments, or jurisdictions. Some firms address it through bilateral integrations, shared discovery infrastructure, network-based solutions, or existing counterparty relationships, but those approaches still do not provide a distinct pre-disclosure authorization step for previously unknown counterparties.
2. **Disclosure may occur before counterparty authorization is clear.** If the beneficiary VASP is misidentified, regulated data may be disclosed to the wrong organization, creating avoidable privacy, compliance, and incident-handling risk. Existing Travel Rule protocols do not by themselves provide a distinct, pre-disclosure authorization boundary. In practice, a VASP may need to exchange regulated customer or compliance data before receiving any clear authorization signal from the counterparty.
3. **User-entered counterparty information is error-prone.** Requiring users to enter addresses, tags, memos, beneficiary details, or other payment or compliance information creates avoidable errors, failed processing, and manual repair work. Those errors can misroute the payment attempt, misdirect the related counterparty exchange, or lead to incorrect disclosure of regulated data.

1.2 Scope of YONA

YONA standardizes pre-disclosure authorization semantics only for payment flows where the originator VASP already has a valid ruleset-defined beneficiary reference.

In Ruleset 1.0, the only beneficiary references are:

- `intent_locator` for pull-payments
- `beneficiary_handle` for push-payments

This ruleset defines:

- authorization-request and terminal-decision semantics
- message validity and field requirements
- cryptographic request/response binding
- repeated-request behavior

A full BIP conformance claim additionally depends on the *Base Interoperability Profile (BIP)* and the *BIP Conformance Test Suite*.

YONA ends at the authorization decision. PII disclosure, Travel Rule exchange, and settlement submission are outside this ruleset. After `ACCEPT`, counterparties continue those steps through their existing providers, channels, bilateral arrangements, and policy controls.

1.3 Explicit non-responsibilities

This section states what Ruleset 1.0 does not define.

Ruleset 1.0 does not define flows where the originator VASP has only a blockchain address, destination coordinate, settlement instruction, or other out-of-scope payment detail.

YONA explicitly does not:

- custody assets or private keys
- submit, execute, intermediate, or guarantee settlement
- operate runtime network services such as routing, relay, directory, or message delivery
- perform AML, sanctions screening, or trust determinations
- define Travel Rule transport or Travel Rule payload schemas
- define address-to-VASP mapping or address proofs

PII exchange and settlement-routing information are outside YONA.

Ruleset 1.0 prohibits the following content from appearing in YONA-defined fields:

- IVMS 101.2023 or other Travel Rule payload objects
- destination addresses, tags, memos, or account identifiers
- settlement instructions or other settlement destination details
- explicit personal-data structures

These prohibitions apply to the content itself, not just to particular field names or object labels. Ruleset 1.0 does not define how implementations should detect or classify such content outside the ruleset's defined fields. Any such handling is local policy and outside this ruleset.

---

2. YONA Flow Model

2.1 Actors and roles

This section defines the actors and roles used throughout this ruleset.

Originator VASP: the VASP that initiates authorization on behalf of a user.

Beneficiary VASP: the VASP that evaluates an authorization request and returns `ACCEPT` or `REJECT`.

Originator (user): the person or entity using the originator VASP application to initiate or confirm a payment.

Beneficiary (user): the person or entity serviced by the beneficiary VASP.

2.2 Pull-payment flow

In a pull-payment flow, `intent_locator` is presented to the originator side by a means outside this ruleset, for example:

- merchant checkout
- a payment link
- a QR code
- another presentation method outside this ruleset

`intent_locator` is only a **non-authoritative reference**. It allows the presentation method to carry a lightweight beneficiary reference instead of the full beneficiary-authored payment intent.

The originator VASP uses this reference to retrieve the **authoritative payment intent** directly from the beneficiary VASP, presents the payment terms to the originator (user) for confirmation, and then requests authorization from the beneficiary VASP.

The beneficiary VASP then returns either `ACCEPT` or `REJECT`.

2.3 Push-payment flow

Push-payment does not begin by retrieving a payment intent.

In a push-payment flow, `beneficiary_handle` is provided to the originator side by a means outside this ruleset, for example:

- a payment link
- a QR code
- another presentation method outside this ruleset

The originator VASP uses that reference to identify the beneficiary VASP and present the beneficiary context to the originator (user), either from `beneficiary_handle` itself or from a user-facing display outside this ruleset.

After the originator VASP obtains user confirmation of the intended transaction, it derives `payment_terms` and `intended_asset_type` from the transaction context and sends an authorization request to the beneficiary VASP.

The beneficiary VASP then returns either `ACCEPT` or `REJECT`.

2.4 What happens after the decision

If the beneficiary VASP returns `ACCEPT`, the parties may proceed to post-`ACCEPT` steps outside YONA, such as Travel Rule exchange and later settlement.

If the beneficiary VASP returns `REJECT`, or if no valid terminal authorization response is obtained in time, the flow does not proceed.

---

3. Authorization Model

Authorization has only two terminal decisions: `ACCEPT` or `REJECT`.

YONA defines no provisional, partial, intermediate, or multi-stage authorization states.

Pull-payment intent retrieval is separate. Failure to obtain a valid, verified `yona.payment_intent` is a retrieval failure, not an authorization outcome (Section 5.2).

3.1 Meaning of ACCEPT

`ACCEPT` means the beneficiary VASP authorizes the originator VASP to proceed to post-`ACCEPT` steps that are outside this ruleset for the referenced intent, such as Travel Rule exchange and later settlement.

A `yona.authorization_response` with `decision = ACCEPT` **MUST NOT** contain:

- PII, whether encrypted or unencrypted
- Travel Rule payloads, including IVMS 101.2023
- destination addresses, tags, memos, or account identifiers
- settlement instructions or other settlement destination details
- explicit personal-data structures

3.2 Meaning of `REJECT`

`REJECT` means authorization is denied for the specified intent.

After `REJECT`, the originator VASP **MUST NOT** proceed to out-of-scope, post-`ACCEPT` steps for that intent.

No inference may be drawn from `REJECT` regarding the reason for denial.

3.3 Timeouts and `NO_RESPONSE`

If the originator VASP does not receive a valid terminal authorization response (`yona.authorization_response`) within 60 seconds, the outcome is `NO_RESPONSE`.

`NO_RESPONSE` **MUST** be treated as `REJECT`.

After `NO_RESPONSE`, the originator VASP **MUST NOT** proceed to out-of-scope, post-`ACCEPT` steps for that intent.

---

4. Message Model and Field Requirements

4.1 Message types

Ruleset 1.0 defines four signed message types:

1. `yona.retrieve_intent` — payment intent retrieval request (pull-payment)
2. `yona.payment_intent` — authoritative payment intent response (pull-payment)
3. `yona.authorization_request` — authorization request (pull-payment and push-payment)
4. `yona.authorization_response` — terminal authorization response (pull-payment and push-payment)

Ruleset 1.0 defines no dedicated signed error or `REJECT` message for `yona.retrieve_intent`.

4.2 Identifiers, common claims, and parsing rules

Ruleset 1.0 uses signed JWS messages. This section defines how counterparties are identified in those messages, which common claims they **MUST** carry, and how receivers **MUST** parse them.

Additional non-normative examples and explanations appear in YONA materials published by YONA, including *Reference Messages and DID Examples*.

4.2.1 Counterparty identifiers

YONA uses **Decentralized Identifiers (DIDs)** to identify counterparties and bind each interaction to the correct VASP context.

For BIP conformance, that context is discovered through the counterparty's `did:web` identifier and the DID document resolved from it. This helps the sender distinguish which VASP it is interacting with and which published verification material and YONA service endpoints apply to that interaction.

Because `did:web` resolution is tied to the counterparty's web domain, it helps the sender identify the correct VASP context and obtain the published verification material and YONA service endpoints for the interaction.

The companion *Base Interoperability Profile (BIP)* defines the interoperable `did:web` discovery and endpoint profile for BIP conformance.

4.2.2 Common required claims

In YONA, signed messages identify the sender and receiver by DID values.

All signed messages **MUST** include:

- `iss` (string): sender DID
- `aud` (string): receiver DID
- `iat` (integer): issued-at, seconds since Unix epoch
- `exp` (integer): expiry, seconds since Unix epoch
- `jti` (string): unique token identifier
- `message_type` (string): one of the defined message types
- `ruleset_id` (string): **MUST** equal `yona:ruleset:v1.0`

Receivers **MUST** reject messages with missing or invalid common claims.

4.2.3 Encoding and parsing requirements

This subsection ensures that the same message is interpreted the same way across implementations.

The JWS payload **MUST** be valid UTF-8 JSON.

Duplicate JSON object member names are invalid and **MUST** be rejected. Receivers **MUST** detect duplicate member names, either through parser support or by inspecting the raw message content.

`iat` and `exp` **MUST** be integers. Receivers **MUST** enforce `exp`.

This ruleset requires `iat`, but does not define a global skew window. Receivers **MAY** enforce skew or freshness under local policy, but **MUST** do so deterministically within a deployment.

Only fields defined in this ruleset have interoperability semantics.

Senders **MUST NOT** require unknown or additional fields to influence validation, binding, timeouts, or outcomes.

Receivers **MUST** ignore unknown or additional fields for those purposes, except where they violate parsing rules, for example, by duplicating a defined field name.

4.3 Shared field contracts

4.3.1 Security-critical identifier constraints

This subsection defines which identifiers are security-critical and how they are constrained.

The following identifiers are security-critical and **MUST** be validated as ASCII-restricted identifiers:

- `jti`
- `intent_id` (originator VASP-issued)

Additionally, when present in pull-payment locator content:

- `intent_locator.beneficiary_intent_id` (beneficiary VASP-issued)

ASCII-restricted identifier means ASCII-only with:

- allowed characters: `A-Z a-z 0-9 : _ -`
- length: 8–128 inclusive
- regex: `^[A-Za-z0-9:_-]{8,128}$`

4.3.2 Amount and currency canonical form (`payment_terms`)

This subsection defines the canonical representation of the payment amount and currency.

`payment_terms` carries the priced amount and currency for the payment. In pull-payment, it appears in the authoritative `yona.payment_intent`. In push-payment, it appears in `yona.authorization_request` as proposed by the originator VASP.

Within `payment_terms`, the following members **MUST** be present:

- `amount` **MUST** be a base-10 integer string in minor units: `^(0|[1-9][0-9]{0,31})$`
- `amount_units` **MUST** equal `minor`
- `currency` **MUST** match `^[A-Z0-9]{2,16}$`

4.3.3 Settlement asset-type identifiers (`acceptable_asset_types`, `intended_asset_type`)

This subsection defines how Ruleset 1.0 identifies settlement asset types in YONA messages.

Under this ruleset, settlement asset types are expressed only as asset-type identifiers. They do not define destination addresses, account identifiers, routing data, or other settlement instructions.

A settlement asset type under this section **MUST** be expressed as a Chain Agnostic Improvement Proposal (CAIP)-19 asset type identifier in the form:

`chain_id/asset_namespace:asset_reference`

The `chain_id` portion of that identifier **MUST** be a CAIP-2 blockchain identifier.

Practical examples appear in the non-normative *Reference Messages and DID Examples* published by YONA.

4.3.3.1 `acceptable_asset_types` (pull-payment)

This subsection defines which settlement asset types a pull-payment intent allows for satisfying `payment_terms`.

In Ruleset 1.0, `acceptable_asset_types` appears in the authoritative beneficiary VASP-issued `yona.payment_intent`.

For `acceptable_asset_types`:

- `acceptable_asset_types` **MUST** be a non-empty array of strings
- Each element **MUST** be a CAIP-19 asset type identifier whose `chain_id` is CAIP-2
- Matching **MUST** be by exact string match
- Ordering is non-semantic and does not affect message validity under this ruleset

`acceptable_asset_types` identifies acceptable asset types only. It does not identify a destination, account, route, or other settlement instruction.

4.3.3.2 `intended_asset_type` (push-payment)

This subsection defines which settlement asset type a push-payment authorization request proposes to use for satisfying `payment_terms`.

In Ruleset 1.0, `intended_asset_type` appears in push-payment `yona.authorization_request`.

Pull-payment does not include `intended_asset_type`, because the beneficiary-authored `yona.payment_intent` already defines `acceptable_asset_types`.

For `intended_asset_type`:

- `intended_asset_type` **MUST** be a CAIP-19 asset type identifier whose `chain_id` is CAIP-2

`intended_asset_type` identifies an asset type only. It does not identify a destination, account, route, or other settlement instruction.

4.4 Flow-specific beneficiary references

4.4.1 `beneficiary_handle` (push-payment)

This subsection defines the beneficiary reference used in push-payment flows.

`beneficiary_handle` identifies the beneficiary context within the beneficiary VASP.

It **MUST**:

- be issued and managed by the beneficiary VASP
- be opaque to the originator VASP and end users
- include the beneficiary VASP DID and an issuer-scoped opaque alias
- **MUST NOT** encode destination addresses, settlement rails, assets, amounts, currency, or settlement instructions

Normative form:

`did=<beneficiary_vasp_did>;alias=<opaque_alias>`

Constraints:

- MUST be ASCII-only, using characters in the range `0x21–0x7E` (no spaces or control characters)
- MUST contain exactly one `did=` and one `;alias=` and no additional fields
- `<opaque_alias>` MUST match: `^[A-Za-z0-9._:-]{8,128}$`

The beneficiary VASP **MUST** deterministically reject if:

- `beneficiary_handle` does not conform to the normative form or constraints
- `aud` does not equal `<beneficiary_vasp_did>`
- the alias is not recognized by the beneficiary VASP

Originator VASPs **MUST** treat `beneficiary_handle` as opaque and **MUST NOT** infer settlement instructions from it.

4.4.2 `intent_locator` object (pull-payment)

This subsection defines the beneficiary reference used in pull-payment flows.

A pull-payment QR code or link conveys an `intent_locator` only, issued by the beneficiary VASP. An `intent_locator` is non-authoritative. It is used only to retrieve the authoritative payment intent from the beneficiary VASP.

`intent_locator` **MUST** be a plain data object, not a signed YONA message, and **MUST NOT** be a JWS Compact Serialization.

It **MUST** include:

- `type`: **MUST** equal `yona.intent_locator`
- `beneficiary_vasp_did`: beneficiary VASP DID

- `beneficiary_intent_id`: beneficiary VASP-issued intent identifier

4.5 Authorization request forms

This section defines how receivers determine whether an authorization request is the pull-payment form or the push-payment form.

Ruleset 1.0 supports exactly two valid `yona.authorization_request` forms. The receiver **MUST** classify them deterministically.

The pull-payment form is valid only when `embedded_payment_intent` is present and `beneficiary_handle`, `payment_terms`, and `intended_asset_type` are absent.

The push-payment form is valid only when `beneficiary_handle`, `payment_terms`, and `intended_asset_type` are present and `embedded_payment_intent` is absent.

Any other combination is invalid and **MUST** be deterministically rejected.

4.6 Request/response binding claim (`request_jws_sha256`)

This section ties each terminal authorization response to the exact authorization request that produced it.

For `yona.authorization_response`, `request_jws_sha256` **MUST** equal:

`base64url(SHA-256(<exact authorization request JWS Compact Serialization as received>))`

`base64url` **MUST** be unpadded.

The beneficiary VASP computes the hash over the exact authorization request JWS Compact Serialization as received on the wire.

The originator VASP recomputes the hash over the exact authorization request JWS Compact Serialization as originally sent.

Implementations **MUST** compute the hash over those exact authorization request JWS Compact Serialization and **MUST NOT** hash any transformed representation, including decoded

payloads, parsed JSON, reserialized JSON, reconstructed JWS, trimmed or normalized strings, or Unicode-normalized strings.

If the recomputed value does not match the received `request_jws_sha256`, the originator VASP **MUST** deterministically treat the response as invalid and **MUST NOT** act on `decision`.

4.7 YONA response messages

This section defines what this ruleset means by a YONA response message.

For purposes of this ruleset, a YONA response message is the BIP-defined HTTP 200 response with `Content-Type: application/jose` whose body carries a defined YONA message for the relevant interaction.

When this ruleset says a party **MUST NOT** return a YONA response message, it means the party **MUST NOT** return that response form for the relevant interaction.

---

5. Payment Intent Retrieval (pull-payment)

5.1 Payment intent retrieval request (`yona.retrieve_intent`)

This section defines how the originator VASP asks the beneficiary VASP for the authoritative payment intent.

The retrieval message (`yona.retrieve_intent`) **MUST** be signed by the originator VASP.

The beneficiary VASP **MUST** verify the signature against the sender's DID document.

Required fields:

- common claims (Section 4.2.2)
- `message_type = yona.retrieve_intent`
- `intent_locator` (Section 4.4.2)

If `aud` does not equal `intent_locator.beneficiary_vasp_did`, the beneficiary VASP **MUST** deterministically treat the request as invalid.

If `intent_locator` is missing, malformed, non-conforming, or fails that binding check, the beneficiary VASP **MUST NOT** return a YONA response message. The originator VASP **MUST** treat this as a retrieval failure and **MUST NOT** proceed to authorization.

5.2 Pull-payment retrieval outcome

This section defines the outcome of pull-payment retrieval.

Pull-payment uses `intent_locator` to retrieve the authoritative `yona.payment_intent` from the beneficiary VASP.

The outcome is either:

- a valid, verified `yona.payment_intent`
- payment intent retrieval failure

Failure to obtain a valid, verified `yona.payment_intent` is a retrieval failure. A retrieval failure is not an authorization outcome and **MUST NOT** be treated as `NO_RESPONSE`, `ACCEPT`, or `REJECT`.

5.3 Payment intent (`yona.payment_intent`)

This section defines the authoritative signed payment intent returned in a pull-payment flow.

The beneficiary VASP returns either a valid signed `yona.payment_intent` or no YONA response message.

Required fields:

- common claims (Section 4.2.2)
- `message_type = yona.payment_intent`
- `intent_locator` (Section 4.4.2)
- `payment_terms` (Section 4.3.2)
- `acceptable_asset_types` (Section 4.3.3.1)

The originator VASP **MUST** verify:

- signature and `kid` resolution under the beneficiary VASP DID
- the `iss` value in `yona.payment_intent` equals the `aud` value in the corresponding `yona.retrieve_intent`
- the `aud` value in `yona.payment_intent` equals the `iss` value in the corresponding `yona.retrieve_intent`
- that the locator matches what was requested
- the canonical forms and constraints above

If the beneficiary VASP cannot produce a valid `yona.payment_intent`, it **MUST NOT** return a YONA response message.

Failure to obtain a valid, verified `yona.payment_intent` is a **payment intent retrieval failure**, not an authorization outcome, and the originator VASP **MUST NOT** proceed to authorization for that pull-payment flow.

5.4 Pull-payment authorization request form

This section defines the pull-payment form of `yona.authorization_request`.

Required fields:

- common claims (Section 4.2.2)
- `message_type = yona.authorization_request`
- `intent_id` (originator VASP-issued)
- `embedded_payment_intent`: exact JWS Compact Serialization of a previously returned `yona.payment_intent`

The beneficiary VASP **MUST** validate:

- the outer request:
  - signature, claims, `exp`, and replay/freshness policy, subject to Sections 7.3 through 7.8
- the embedded payment intent:
  - parse as JWS, verify signature under the beneficiary VASP DID, enforce `exp`, and apply local replay/freshness policy in a manner consistent with Sections 7.3 through 7.8
- binding:
  - `authorization_request.iss` **MUST** equal the `aud` value in the embedded `yona.payment_intent`

- `authorization_request.aud` **MUST** equal the `iss` value in the embedded `yona.payment_intent`

5.5 Pull-payment ordering

This section defines the required order for the pull-payment flow.

1. The originator VASP **MUST** retrieve and verify `yona.payment_intent` before presenting it to the originator (user).
2. User confirmation **MUST** occur in the originator UX before authorization.
3. Authorization **MUST** occur before any out-of-scope disclosure.
4. Settlement **MUST NOT** be submitted before `ACCEPT`. Any settlement processing that occurs after `ACCEPT` is out of scope.

---

6. Push-Payment Authorization

6.1 Push-payment authorization request form

This section defines the push-payment form of `yona.authorization_request`.

Required fields:

- common claims (Section 4.2.2)
- `message_type = yona.authorization_request`
- `intent_id` (originator VASP-issued)
- `beneficiary_handle` (Section 4.4.1)
- `payment_terms` (Section 4.3.2)
- `intended_asset_type` (Section 4.3.3.2)

The beneficiary VASP **MUST** validate:

- request gating: signature, claims, `exp`, and local replay/freshness policy, applied consistently with Sections 7.3 through 7.8
- `beneficiary_handle` form, plus `aud` matching the DID inside the handle
- `payment_terms` canonical form

- `intended_asset_type` canonical form

6.2 Push-payment ordering

This section defines the required order for push-payment authorization.

1. The originator VASP **MUST** have a valid `beneficiary_handle` before sending `yona.authorization_request`.
2. Before sending `yona.authorization_request`, the originator VASP **MUST** present the beneficiary context to the originator (user) and obtain confirmation to proceed.
3. After confirmation and before sending `yona.authorization_request`, the originator VASP **MUST** derive `payment_terms` and `intended_asset_type` from the transaction context.
4. Authorization **MUST** occur before disclosure and before settlement submission.
5. Settlement **MAY** occur only after `ACCEPT` and is out of scope.

---

7. Beneficiary VASP Authorization Response

7.1 Authorization response (`yona.authorization_response`)

This section defines the signed terminal authorization response returned by the beneficiary VASP.

Required fields:

- common claims (Section 4.2.2)
- `message_type = yona.authorization_response`
- `intent_id = authorization_request.intent_id`
- `decision = ACCEPT` or `REJECT`
- `request_jws_sha256` (Section 4.6)

The originator VASP **MUST** validate:

- signature and `kid` resolution under the beneficiary VASP DID
- `iss = authorization_request.aud`
- `aud = authorization_request.iss`
- `intent_id = authorization_request.intent_id`
- `request_jws_sha256` according to Section 4.6

The originator VASP **MUST NOT** act on `decision` unless all of those checks succeed.

7.2 Terminal authorization response behavior

This section defines how the beneficiary VASP returns a terminal authorization response once it has enough information to do so.

For any `yona.authorization_request` that the beneficiary VASP can bind to a terminal response, it returns one `yona.authorization_response` with `decision = ACCEPT` or `REJECT`.

If a received `yona.authorization_request` fails validation gating and the beneficiary VASP can construct a structurally valid bound response, it **MUST** return a signed terminal `REJECT` and **MUST NOT** proceed to authorization evaluation.

If the beneficiary VASP cannot construct a bound terminal response, it **MUST NOT** return a YONA response message. The originator VASP then treats that outcome as `NO_RESPONSE`, and `NO_RESPONSE` **MUST** be treated as `REJECT`.

7.3 Same authorization context (`iss`, `intent_id`)

This section defines when two requests are treated as repeats of the same authorization context.

Repeated-request handling is scoped to (`iss`, `intent_id`) within an implementation-defined retention window.

Within this section, `intent_id` identifies the originator-defined authorization context.

Requests within the same (`iss`, `intent_id`) scope are treated as repeats of the same authorization context for purposes of this section.

7.4 Repeated requests and response binding

This section explains what must stay the same, and what must be rebound, when an authorization request is repeated.

Repeated requests within the same (`iss`, `intent_id`) scope **MAY** differ at the byte level, for example, because they carry a new `jti`, `iat`, `exp`, `kid`, or signature.

The beneficiary VASP **MUST NOT** blindly reuse a prior response message.

For each repeated request that receives a terminal response, the beneficiary VASP **MUST** construct a new response bound to the exact repeated request bytes through `request_jws_sha256`.

A repeated request **MAY** therefore receive a newly generated response message even when the terminal decision remains the same.

7.5 Outcome rules for repeated requests

This section defines what happens when the beneficiary VASP receives a repeated authorization request.

Within the locally defined retention window, the beneficiary VASP **MUST** evaluate repeated requests as follows:

- If the repeated request is materially equivalent to the earlier request, the beneficiary VASP **MUST** return the same terminal decision, newly bound to the exact repeated request bytes.
- If the repeated request changes one or more material authorization inputs, the beneficiary VASP **MUST** deterministically reject if it can construct a bound terminal response.
- If the beneficiary VASP cannot construct a bound terminal response for that materially changed request, it **MUST NOT** return a YONA response message. The originator VASP then treats the outcome as `NO_RESPONSE`, which **MUST** be treated as `REJECT`.

7.6 Material authorization inputs

This section defines which request changes are capable of changing the authorization decision.

For repeated-request evaluation under Sections 7.3 through 7.5, the beneficiary VASP **MUST** first:

1. identify the repeated-request scope using (`iss`, `intent_id`) under Section 7.3; and
2. deterministically classify the request as pull-payment or push-payment under Section 4.5.

Within that scope and request form, the following inputs are material to the authorization decision.

Common:

- `ruleset_id`
- `aud`

Pull-payment:

- exact `embedded_payment_intent` JWS Compact Serialization

Push-payment:

- `beneficiary_handle`
- `payment_terms.amount`
- `payment_terms.amount_units`
- `payment_terms.currency`
- `intended_asset_type`

A change to any material authorization input causes the repeated request to be treated as materially changed for purposes of Section 7.5.

7.7 Non-material differences

This section defines request differences that do not, by themselves, create a new authorization decision.

Within the retention window, the following differences are non-material by themselves and **MUST NOT** justify a different terminal decision for a materially equivalent repeat:

- `jti`
- `iat`
- `exp`
- `kid`
- signature bytes

These fields **MAY** differ across repeated requests, but such differences alone do not create a new authorization decision.

7.8 Replay protection and freshness consistency

This section prevents local replay or freshness controls from breaking the repeated-request behavior defined in Sections 7.3 through 7.7.

When evaluating repeated `yona.authorization_request` messages within the same (`iss`, `intent_id`) scope, a beneficiary VASP **MUST NOT** treat a materially equivalent repeat within the retention window as a replay solely because the repeated request carries a different `jti`.

A beneficiary VASP **MAY** enforce local replay, freshness, and retention policies for `yona.authorization_request`, including replay protection for (`iss`, `jti`), but those policies **MUST** preserve the repeated-request behavior defined in Sections 7.3 through 7.7.

---

8. Validation and Ruleset Binding

8.1 Validation gating

This section defines the gate that determines whether a received message is eligible to be processed under this ruleset.

`ruleset_id` **MUST** be present and **MUST** be supported locally.

For conformance to this ruleset, implementations **MUST** support `ruleset_id = yona:ruleset:v1.0`.

Messages with unknown or unsupported `ruleset_id` **MUST** be rejected deterministically and **MUST NOT** proceed to authorization evaluation.

Receivers **MUST** deterministically reject messages that violate required structure or constraints under this ruleset, including:

- invalid JWS or JSON structure
- missing required claims
- duplicate JSON member names
- non-integer timestamps
- non-conforming identifiers
- non-canonical amounts or currency
- invalid CAIP identifiers where required

8.2 Ruleset binding

This section defines how a signed message is bound to the ruleset whose requirements apply to it.

Each signed YONA message defined in Section 4.1 **MUST** include `ruleset_id`.

`ruleset_id` selects the ruleset whose semantics, including validity checks, required fields, and binding behavior, **MUST** be applied to that message.

Receivers **MUST** deterministically reject a message if:

- `ruleset_id` is missing
- `ruleset_id` is not supported by local configuration
- the message fails the selected ruleset's requirements

Ruleset support is purely local policy. Implementations **MUST** be able to validate and process YONA messages using locally configured ruleset information and normal counterparty DID resolution, without runtime approval from, registration with, or queries to YONA LLC or any future YONA successor body.

---

9. Signed Messages

9.1 Cryptographic algorithm profile

This section defines the required JWS format and protected-header values for signed YONA messages.

All YONA messages **MUST** be transmitted as JWS Compact Serialization (RFC 7515).

The JWS Protected Header **MUST** include:

- `alg = EdDSA`
- `typ = JWT`
- `kid =` a DID URL identifying the signing key in the sender DID document

---

9.2 Signature verification

This section defines how a receiver confirms that a signed YONA message was created by the expected counterparty.

Receivers **MUST** resolve the sender DID document and verify the signature using the verification method referenced by `kid`.

Verification **MUST** be local and **MUST NOT** rely on YONA at runtime.

9.3 Replay and freshness

This section defines the baseline replay and freshness checks for signed YONA messages.

Receivers **MUST** enforce:

- expiry (`exp`)
- replay protection for (`iss`, `jti`) under local policy, with deterministic rejection of a replayed message within the locally defined replay-retention window

Receivers **MAY** enforce additional `iat` freshness checks under local policy.

10. Role of the Companion BIP

10.1 What the companion BIP defines

The companion *Base Interoperability Profile (BIP)* defines the interoperable `did:web` discovery, transport, and endpoint profile for BIP conformance under `yona:ruleset:v1.0`.

This companion BIP applies to `yona:ruleset:v1.0`; any future YONA ruleset release may publish its own identified companion BIP for that release.

A BIP conformance claim under `yona:ruleset:v1.0` **MUST** implement the companion BIP and pass the *BIP Conformance Test Suite*.

The companion BIP does not change YONA message validity, authorization semantics, request/response binding, or terminal outcomes, which are defined by this ruleset independently of transport.

10.2 Additional arrangements

Implementations **MAY** support additional transports or discovery arrangements for YONA by mutual agreement, but those arrangements are outside BIP conformance and **MUST** preserve the same YONA message bytes and semantics.

10.3 Optional post-`ACCEPT` and assurance discovery pointers

The companion BIP defines optional DID service entries for steps that happen after a VASP receives an `ACCEPT` decision. These include discovery pointers for Travel Rule data exchange, settlement-instruction exchange, and assurance material.

It also defines an optional machine-readable capability-discovery pattern for `YonaTravelRuleService` and `YonaSettlementService`, so a counterparty can determine

which post-`ACCEPT` step a published discovery pointer is for and how the advertising VASP expects that out-of-band step to be initiated.

Such assurance material **MAY** include conformance claims, reports, certificates, or equivalent assurance artifacts.

A VASP **MAY** use such capability information or assurance material under local policy to decide whether and how to proceed after `ACCEPT`, whether to establish a counterparty relationship, or whether to transact with that counterparty.

The presence, absence, or use of these optional service entries, or of any capability or assurance material discovered through them, **MUST NOT** change YONA message validity, authorization evaluation, request/response binding, terminal outcomes, or conformance semantics.

---

11. Versioning, Governance, and Lifecycle

11.1 Release stewardship and governance scope

This section defines who stewards this release and what governance does and does not control.

YONA LLC publishes and currently stewards Ruleset 1.0 and its companion materials.

Governance under YONA applies only to the proposal, review, approval, publication, deprecation, and retirement of YONA releases.

Governance does not affect YONA message validity, request/response binding, authorization outcomes, conformance semantics, or any runtime interaction between counterparties.

Further organizational and governance detail is described in the *YONA Governance Statement* published on YONA's website.

If a future YONA successor body stewards later YONA releases, that change **MUST NOT** alter the meaning of Ruleset 1.0.

11.2 Publication authority, immutability, and upgrade model

Each published ruleset release is immutable.

Approval or publication of a future release (through the applicable YONA governance process) does not modify the immutable semantics of an already published release.

Any change to normative requirements **MUST** be published as a new release with a new `ruleset_id`.

`ruleset_id` is the immutable protocol identifier used for interoperability decisions.

Human-readable labels such as `v1.0` are descriptive only.

VASPs independently choose which `ruleset_id` values they accept through local configuration.

11.3 Lifecycle metadata

For each release, the entity then serving as release steward for that release publishes:

- normative ruleset text
- immutable `ruleset_id`
- lifecycle metadata (`active`, `deprecated`, or `retired`)
- license notice for that release
- where applicable, references to companion conformance materials

Lifecycle metadata does not modify the immutable semantics of an already published release.

---

12. Security Considerations

This section explains the main failures the ruleset is designed to prevent. It adds no new protocol requirements.

The security of a YONA implementation depends on these invariants:

1. the originator VASP correctly identifies the intended beneficiary VASP and authenticates the endpoint used for the YONA interaction
2. signed messages are verified against the correct counterparty keys
3. invalid, ambiguous, expired, replayed, or unsupported messages do not reach authorization evaluation
4. each terminal authorization response is bound to the exact authorization request bytes that produced it
5. no post-`ACCEPT` action occurs unless a valid, bound `ACCEPT` has been received

These invariants are enforced by the participating VASP systems and their local orchestration. YONA does not define settlement-rail controls or require the settlement rail itself to enforce YONA authorization state.

Failure to preserve these invariants can cause wrong-counterparty disclosure, unauthorized disclosure, incorrect authorization outcomes, substitution or replay across contexts, or premature settlement.

---

13. Conformance

This section defines the minimum basis for making a BIP conformance claim under `yona:ruleset:v1.0`.

Conformance is optional. It does not affect message validity, authorization semantics, request/response binding, terminal outcomes, or create any runtime dependency on YONA.

A BIP conformance claim for `yona:ruleset:v1.0` MUST NOT be made unless the implementation:

1. satisfies all applicable normative requirements in *YONA Ruleset 1.0*;
2. satisfies all applicable normative requirements in the companion *Base Interoperability Profile (BIP)*; and
3. passes the applicable normative test cases in the *BIP Conformance Test Suite*.

The *BIP Conformance Test Suite* verifies conformance to *YONA Ruleset 1.0* and the *Base Interoperability Profile (BIP)*. It adds no new protocol requirements beyond those documents.

A push-payment-only pilot is not a full BIP conformance claim and MUST be described accurately, consistent with the *Base Interoperability Profile (BIP)* and *BIP Conformance Test Suite*.