Workgroup:	Network Working Group
Internet-Draft:	draft-solana-subscription-00
Published:	26 October 2026
Intended Status:	Informational
Expires:	29 April 2027
Authors:	L. Galabru	*Solana Foundation*
---	---	---
	Solana Foundation	

Solana Subscription Intent for HTTP Payment Authentication

Abstract

This document defines the Solana profile of the "subscription" payment intent for use with the Payment HTTP Authentication Scheme. It specifies how clients grant servers permission to collect a fixed SPL token payment once per billing period using a subscription delegation held by an audited on-chain program. This profile also defines a reusable payer-signed bearer proof for access under an active subscription. It intentionally models the recurring transfer authorization itself, not a richer billing object.

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 29 April 2027.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 4
   1.1. Relationship to the Subscription Intent 4
2. Requirements Language 4
3. Terminology 5
4. Intent Semantics 5
   4.1. Definition 5
   4.2. Properties 6
   4.3. Flow 6
5. Request Schema 7
   5.1. Shared Fields 7
      5.1.1. Required Fields 8
      5.1.2. Optional Fields 8
   5.2. Currency Formats 9
   5.3. Method Details 9
   5.4. Implementor Guidance 10
   5.5. Examples 11
6. Credential Requirements 11
   6.1. Payload 12
   6.2. Subscription Bearer Proof 12
   6.3. Activation Single-Use 15
7. Subscription Lifecycle 15
   7.1. Activation 15
      7.1.1. Source Verification 16
      7.1.2. Authorization Scope Verification 17
   7.2. Renewal 17
   7.3. Subscription Identifier 18

7.4. Server Accounting and Idempotency 19
7.5. Cancellation 20
7.6. Error Responses 20
8. Illustrative Lifecycle Examples 21
8.1. Daily Billing Example 21
8.2. Cancellation Example 24
8.3. Failed Renewal Example 24
8.4. Expiry Example 25
9. Security Considerations 25
9.1. Recurring Charge Awareness 25
9.2. Amount and Period Verification 25
9.3. Duplicate Charge Prevention 26
9.4. Server Accountability 26
9.5. Caching 26
9.6. Bearer Proof Theft 26
9.7. Destination Scoping 26
9.8. Plan Scope Minimization 27
9.9. Subscription Authority Isolation 27
9.10. Token-2022 Extension Policy 27
9.11. Account Ownership and Program-ID Validation 27
10. IANA Considerations 28
10.1. Payment Intent Registration 28
11. References 28
11.1. Normative References 28
11.2. Informative References 29
Appendix A. Acknowledgements 29
Authors' Addresses 29

1. Introduction

The "subscription" intent on Solana represents a recurring fixed-amount SPL token payment. The client grants the server a recipient-scoped subscription delegation with a per-period spending limit. Activation creates the delegation and collects the first billing-period charge in the same transaction.

This intent is useful for recurring API plans, content subscriptions, and other Solana-priced services with a stable amount per billing period.

This profile is intentionally narrower than a general billing subscription. It standardizes a recurring token-transfer authorization, not price catalogs, quantities, prorations, deferred starts, or billing-anchor resets.

1.1. Relationship to the Subscription Intent

This document inherits the shared request semantics of the "subscription" intent from [I-D.payment-intent-subscription]. It defines only the Solana-specific `methodDetails`, `payload`, activation transaction shape, on-chain lifecycle, and verification procedures for the "solana" payment method.

Solana subscriptions support only subscription-delegation fulfillment. Solana transactions containing standalone SPL Token `Approve` instructions and push-mode hash credentials do not provide the per-period enforcement required for this intent.

Solana also imposes an additional constraint that is not part of the shared intent: the recurring authorization **MUST** be created against a `Plan` account that the merchant has published on-chain prior to the challenge. This method therefore requires the base58 address of that on-chain plan in `methodDetails.planAddress`. The shared `externalId` field retains its meaning as the merchant's reference for the subscription.

Solana subscriptions also require the per-period spending limit, recipient scoping, and missed-period non-accumulation described in this document, including the Token-2022 mint constraints in Section 9.10. Servers **MUST** reject request objects on deployments that cannot enforce those restrictions.

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.

3. Terminology

This document uses the terms `Subscription`, `Billing Period`, `Activation`, `Renewal`, `Cancellation`, and `Subscription Identifier` as defined by [I-D.payment-intent-subscription]. The following additional terms are specific to the Solana profile.

SPL Token Solana's standard token program. SPL Token-2022 is its extension- enabled successor and is supported subject to Section 9.10.

Subscriptions Program The audited on-chain program implementing the account model and instructions referenced by this specification [SUBSCRIPTIONS-PROGRAM].

Plan An on-chain PDA published by the merchant that defines subscription terms and lifecycle controls. Core billing terms -- token mint, amount per billing period, period length, creation timestamp, and recipient destinations -- are immutable, while status, end timestamp, allowed pullers, and metadata URI are mutable until the plan is sunset. Derived from `["plan", owner, plan_id]`.

Puller The plan owner or an address in the on-chain `Plan.pullers` array that signs a `transfer_subscription` instruction. This profile uses the program's `puller` name for the server-controlled signing address authorized to collect subscription charges.

Subscription Delegation A per-subscriber on-chain PDA that snapshots the plan terms at subscription time and tracks current-period accounting state. Derived from `["subscription", plan_pda, subscriber]`.

Subscription Authority A per-(subscriber, mint) on-chain PDA that holds the SPL Token delegate authority over the associated token account for that subscriber. The subscriptions program signs transfers as this PDA. Derived from `["SubscriptionAuthority", subscriber, mint]`.

4. Intent Semantics

4.1. Definition

The "subscription" intent on Solana represents a request for a recurring fixed-amount SPL token payment of `amount`, charged once per billing period until explicit cancellation, until the on-chain subscription delegation is otherwise invalidated, or until the optional `subscriptionExpires` timestamp is reached.

On Solana, the recurring authorization is held by an audited on-chain program [SUBSCRIPTIONS-PROGRAM]. Servers **MUST** pin its deployment address in `methodDetails.subscriptionProgram`, and clients **MUST** validate that address before signing. The program defines three on-chain accounts referenced by this specification:

- **Plan**, published off the critical path of the 402 challenge by the merchant. The plan's core terms (mint, amount per billing period, period length, destinations) are immutable once published.
- **SubscriptionDelegation**, created when the subscriber activates a subscription. The delegation snapshots the plan's terms and tracks the current billing-period start time and the amount already pulled in the current period.
- **SubscriptionAuthority**, created per `(subscriber, mint)` and acting as the on-chain delegate over the subscriber's token account.

The program enforces per-period spending limits, recipient scoping, and missed-period non-accumulation. When a transfer occurs after one or more billing periods have elapsed, renewals advance the current billing-period start by whole multiples of the period length and reset the in-period counter to zero. Although the on-chain primitive permits partial transfers up to a per-period cap, this HTTP profile permits only one transfer of exactly `amount` per billing period.

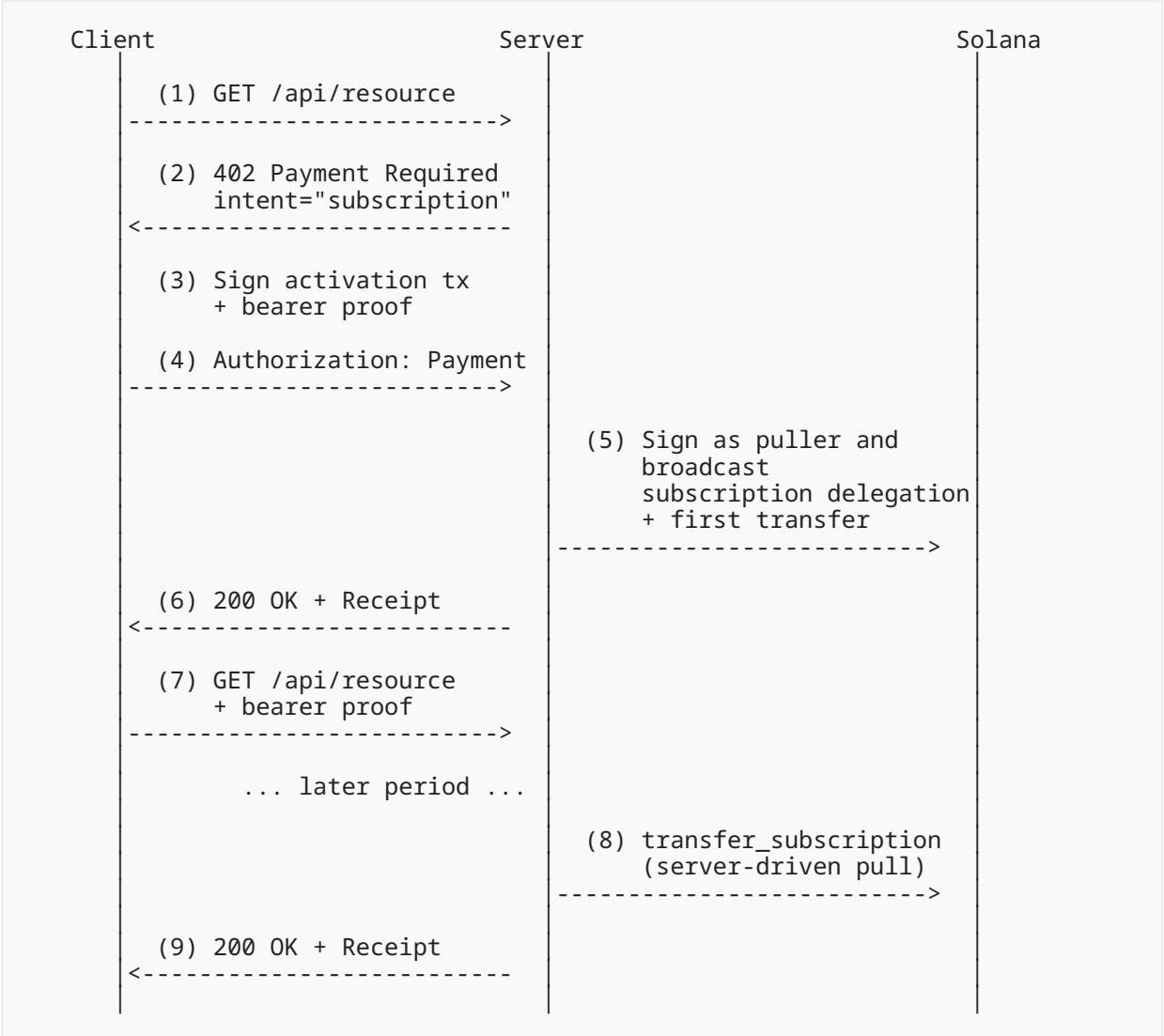
4.2. Properties

Property	Value
Intent Identifier	`subscription`
Payment Timing	Recurring (activation charge atomic with delegation creation, then once per period via server-driven pulls)
Idempotency	Activation credential single-use; bearer proof reusable while the subscription remains usable; on-chain delegation reusable across billing periods
Reversibility	Cancellable on-chain; effective at end of currently-paid billing period

Table 1

4.3. Flow

The following diagram illustrates the Solana subscription flow:



5. Request Schema

The `request` parameter in the `WWW-Authenticate` challenge contains a base64url-encoded JSON object. The `request` JSON **MUST** be serialized using JSON Canonicalization Scheme (JCS) [RFC8785] and base64url-encoded without padding per [I-D.httpauth-payment].

5.1. Shared Fields

Solana uses the shared `amount`, `currency`, `periodUnit`, `periodCount`, `subscriptionExpires`, `recipient`, `description`, and `externalId` fields from [I-D.payment-intent-subscription], with their meanings preserved. The Solana profile elevates `recipient` from **OPTIONAL** to **REQUIRED** and constrains the values that `periodUnit` may take.

5.1.1. Required Fields

Field	Type	Description
`amount`	string	Fixed payment amount per billing period in base units
`currency`	string	SPL Token or Token-2022 mint address (see Section 5.2)
`periodUnit`	string	Billing period unit. The value **MUST** be `day` or `week`
`periodCount`	string	Positive integer count of `periodUnit` values per billing period
`recipient`	string	Wallet address that receives every subscription charge. The on-chain `Plan` and each transfer **MUST** bind this address as described below
`methodDetails`	object	Solana-specific extension data (see Section 5.3)

Table 2

5.1.2. Optional Fields

Field	Type	Description
`subscriptionExpires`	string	Subscription expiry timestamp in [RFC3339] format. When omitted, the subscription has no HTTP-layer maximum lifetime
`description`	string	Human-readable subscription description
`externalId`	string	Merchant's reference for the subscription

Table 3

The `amount` value **MUST** be a string representation of a positive integer in base 10 with no sign, decimal point, exponent, or surrounding whitespace. Leading zeros **MUST NOT** be used.

The `periodCount` value **MUST** be a string representation of a positive integer in base 10 with no sign, decimal point, exponent, or surrounding whitespace. Leading zeros **MUST NOT** be used.

Solana subscription delegations use fixed elapsed-time periods and cannot represent calendar-month billing exactly. Servers **MUST** reject `periodUnit="month"`. The shared period fields map to a per-billing- period interval in hours as follows:

- `periodUnit="day"` maps to `periodCount * 24` hours
- `periodUnit="week"` maps to `periodCount * 168` hours

Servers **MUST** reject request objects where the mapped per-billing- period interval is zero or exceeds 8760 hours.

When `subscriptionExpires` is present, it defines an HTTP-layer maximum subscription lifetime: after that timestamp the server ceases renewal submissions and serves fresh challenges (see Section 7.6). When omitted, the subscription runs until on-chain cancellation, plan end, or another invalidation event takes effect.

5.2. Currency Formats

The `currency` field on Solana is the base58-encoded mint address of an SPL Token or SPL Token-2022 mint:

Format	Example	Description
Token mint	`"EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"`	Base58 address of an SPL Token or Token-2022 mint

Table 4

Implementations **MUST** treat `currency` and `methodDetails.mint` as the same value; servers **MUST** reject request objects where they differ. Native SOL is not a valid currency for the "subscription" intent on Solana.

Base58 values in this profile use the standard Solana alphabet [BASE58]. Address comparisons are by decoded value, not raw string form.

5.3. Method Details

All Solana-specific request parameters live in `methodDetails`:

Field	Type	Required	Description
`methodDetails.mint`	string	**REQUIRED**	Mint address echoed from `currency`. **MUST** equal the on-chain `plan.mint`
`methodDetails.decimals`	number	**REQUIRED**	Decimal precision of the mint
`methodDetails.tokenProgram`	string	**REQUIRED**	Token program ID. The value **MUST** be the SPL Token program or the SPL Token-2022 program
`methodDetails.puller`	string	**REQUIRED**	Base58 of the server's puller pubkey. **MUST** be `plan.owner` or appear in `plan.pullers`

Field	Type	Required	Description
`methodDetails.planAddress`	string	**REQUIRED**	Base58 address of the on-chain `Plan` account
`methodDetails.subscriptionProgram`	string	**REQUIRED**	Base58 address of the subscriptions program deployment
`methodDetails.network`	string	**OPTIONAL**	`"mainnet"`, `"devnet"`, `"testnet"`, or `"localnet"`. If omitted, the default value is `"mainnet"`
`methodDetails.feePayer`	boolean	**OPTIONAL**	If `true`, the client constructs the activation transaction with the server as fee payer
`methodDetails.feePayerKey`	string	**OPTIONAL**	Base58 of the server fee-payer pubkey. **REQUIRED** when `feePayer` is `true`
`methodDetails.recentBlockhash`	string	**OPTIONAL**	Pre-fetched blockhash to bind to the activation transaction

Table 5

Servers **MUST** reject request objects where the on-chain `Plan` at `methodDetails.planAddress` does not exist, has been closed, is not owned by `methodDetails.subscriptionProgram`, or has terms that diverge from `currency`, `amount`, or the mapped per-billing-period interval. Servers **MUST** also verify that the mint account is owned by `methodDetails.tokenProgram` and that `methodDetails.decimals` matches the decimal precision recorded in the mint account.

The non-zero entries in `plan.destinations` **MUST** consist of exactly one address, and that address **MUST** equal `recipient`. Servers and clients **MUST** reject request objects referencing a plan with an empty destination whitelist, more than one non-zero destination, or a destination different from `recipient`. The subscriptions program compares this whitelist against the owner of the receiver token account on every transfer, so this constraint binds activation and all renewals to the advertised recipient.

5.4. Implementor Guidance

This section is non-normative.

The Solana profile is a deliberately narrow projection of what an on-chain subscription system could express. Implementations should:

- Treat the on-chain `Plan` as the authoritative source of recurring terms. The challenge `request` **MUST** agree with the plan exactly; servers should reject mismatched challenges before any client signing.
- Refuse to map `periodUnit="month"` rather than approximate it with 30-day or 31-day fixed periods. Clients receiving a `month` request for the Solana method should treat it as a server bug.
- Publish a separate `Plan` for each recipient. This profile requires exactly one non-zero `Plan.destinations` entry so that the program can enforce the challenge's `recipient` on every transfer.
- Submit at most one `transfer_subscription` per billing period per subscription, and never retry past the end of the period the transaction was constructed against.
- Maintain durable server state sufficient to prevent duplicate charges across retries, concurrent requests, and out-of-band on- chain events such as failed renewals or independent cancellations.

5.5. Examples


```
{
  "amount": "10000000",
  "currency": "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v",
  "periodUnit": "day",
  "periodCount": "30",
  "subscriptionExpires": "2026-07-14T12:00:00Z",
  "recipient": "9xQeWvG816bUx9EPjHmaT23yvVM2ZWbrrpZb9PusVFin",
  "description": "Monthly Pro plan",
  "externalId": "merchant-subscription-270",
  "methodDetails": {
    "subscriptionProgram": "De1egAFMkMWZSN5rYXRj9CAdheBamobVNubTsi9avR44",
    "planAddress": "8tWbqLkUJoYy7zXc5h2EvCRoaQEv2xnQjUuYhc3rzCgT",
    "mint": "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v",
    "tokenProgram": "TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA",
    "decimals": 6,
    "puller": "5fKb5cF22cFybZB1H4hLDydFhwoQy9JzKzRWaSbMkB6h",
    "network": "mainnet",
    "feePayer": true,
    "feePayerKey": "5fKb5cF22cFybZB1H4hLDydFhwoQy9JzKzRWaSbMkB6h"
  }
}
```


6. Credential Requirements

The credential in the `Authorization` header contains a base64url- encoded JSON object per [I-D.httpauth-payment].

6.1. Payload

The credential `payload` for a Solana "subscription" intent either activates the subscription or authenticates later use of an active subscription. Renewals are server-driven on-chain transactions and do not produce HTTP credentials. Cancellations are out-of-band on-chain operations and use no credential.

Field	Type	Required	Description
`challenge`	object	**REQUIRED**	Echo of the challenge from the server
`payload`	object	**REQUIRED**	Solana-specific activation or access payload
`source`	string	**OPTIONAL**	Subscriber identifier (e.g., `did:pkh:solana:...`)

Table 6

Solana subscription activation uses `type="transaction"`. The client signs the activation transaction as the subscriber and submits the serialized bytes. The server verifies the transaction, signs it as `methodDetails.puller`, adds the fee-payer signature when `methodDetails.feePayer` is `true`, and broadcasts it.

Field	Type	Required	Description
`type`	string	**REQUIRED**	The string `"transaction"`
`transaction`	string	**REQUIRED**	Standard-base64 of the activation transaction signed by the subscriber. The server adds the puller signature and, when configured, its fee-payer signature
`authentication`	object	**REQUIRED**	Reusable payer proof defined in Section 6.2

Table 7

6.2. Subscription Bearer Proof

The client signs one reusable proof when it constructs the activation credential. The proof authorizes access under the resulting subscription; it does not authorize activation, a renewal transfer, or any other movement of funds.

The client derives the `SubscriptionDelegation` PDA from the plan and subscriber, then constructs this JCS [RFC8785] object:

```
{
  "domain": "mpp-subscription-auth-v1",
  "payer": "<base58 subscriber public key>",
  "subscriptionChallengeId": "<activation challenge id>",
  "subscriptionDelegation": "<base58 delegation address>"
}
```

The client signs the UTF-8 bytes of the JCS serialization with the Ed25519 private key corresponding to `payer`. The resulting `authentication` object has this shape:

Field	Type	Required	Description
`type`	string	**REQUIRED**	The string `"proof"`
`challengeId`	string	**REQUIRED**	Activation challenge ID signed as `subscriptionChallengeId`
`payer`	string	**REQUIRED**	Base58 Ed25519 public key that produced the proof
`signature`	string	**REQUIRED**	Base58 encoding of the 64-byte Ed25519 signature

Table 8

During activation, the server **MUST** verify that:

- `authentication.challengeId` equals the activation challenge ID;
- the signed `subscriptionDelegation` equals the canonical PDA derived from `methodDetails.planAddress`, the subscriber, and `methodDetails.subscriptionProgram`;
- `authentication.payer` equals the subscriber signer extracted from the activation transaction; and
- the Ed25519 signature verifies over the exact JCS object above.

The server **MUST** bind the verified `authentication` object to the subscription only after the activation transaction confirms and the resulting delegation passes the post-settlement checks in Section 7.1. A signature over a derivable PDA whose account does not exist is not evidence of an active subscription.

To use an active subscription on a later request, the client sends a credential whose outer `challenge` echoes the activation challenge and whose payload has this shape:

```
{
  "type": "proof",
  "subscriptionDelegation": "<base58 delegation address>",
  "authentication": {
    "type": "proof",
    "challengeId": "<activation challenge id>",
    "payer": "<base58 subscriber public key>",
    "signature": "<base58 Ed25519 signature>"
  }
}
```

The server **MUST** verify the proof against the delegation and the activation binding before granting access. It **MUST** require the same `authentication` field values that it bound at activation. The activation challenge's `expires` auth-param limits creation of the binding; it does not expire an already-bound proof. Repeated presentation is the expected bearer- proof behavior and **MUST NOT** be rejected merely because the proof was previously presented.

The proof remains usable only while the identified subscription is usable under Section 7.4. It becomes invalid when cancellation takes effect, `subscriptionExpires` is reached, the delegation or its authority is invalidated, or the current billing period is not paid. Servers **MAY** provide application-layer early revocation or narrower audience and lifetime policies.

The proof is a bearer credential after signing. Clients and servers **MUST** send it only over authenticated encrypted transports, **MUST NOT** log it, and **MUST** prevent it from appearing in URLs, analytics, or error reports. An `Idempotency-Key` prevents replay of an application request; it does not constrain use of a leaked bearer proof.

The subscriber-signed activation transaction **MUST**:

- contain a `subscribe` instruction creating the `SubscriptionDelegation` PDA from the on-chain `Plan` snapshot;
- contain a `transfer_subscription` instruction collecting the first- period charge atomically with the subscription creation;
- contain an `initialize_subscription_authority` instruction prepended only when the `(subscriber, mint)` authority does not yet exist on-chain;
- target the subscriptions program identified by `methodDetails.subscriptionProgram`;
- use the SPL Token or Token-2022 program identified by `methodDetails.tokenProgram` for all token-touching instructions;
- pull funds from the subscriber's associated token account for `methodDetails.mint`;
- use `methodDetails.puller` as the required signer for the `transfer_subscription` instruction;
- direct the first-period charge to a receiver ATA whose token-account owner equals `recipient`;
- set the fee payer to `methodDetails.feePayerKey` when `methodDetails.feePayer` is `true`, and to the subscriber otherwise;

- contain no instructions other than those above plus optional compute-budget and memo instructions.

The subscriber-signed activation transaction **MUST NOT** contain SPL Token `Approve` or any other non-subscriptions-program instruction that could move the subscriber's tokens outside the per-period limit, and **MUST NOT** reference writable accounts that could redirect funds away from `recipient`.

6.3. Activation Single-Use

Each "subscription" activation credential **MUST** be usable only once per challenge. Servers **MUST** reject replayed credentials.

This single-use requirement applies to `type="transaction"` activation credentials. It does not apply to a `type="proof"` access credential defined in Section 6.2.

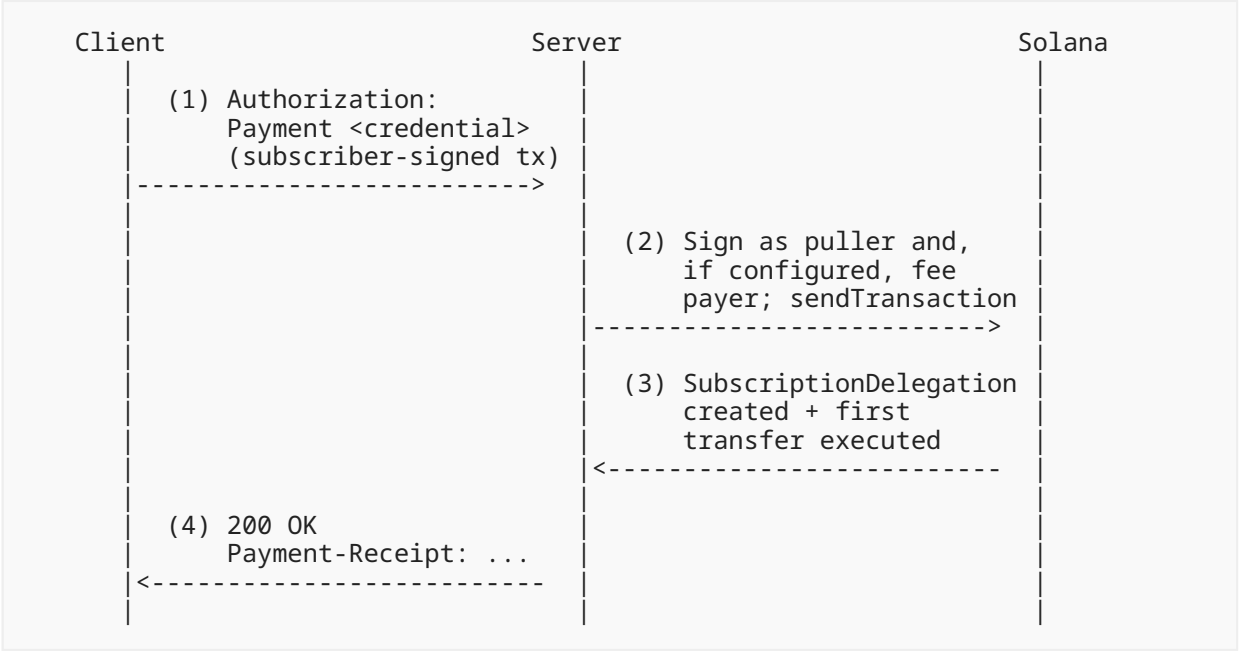
A successfully activated subscription may be reused for later billing periods until:

- The subscriber cancels it on-chain;
- The on-chain delegation is otherwise invalidated (e.g., closure of the `SubscriptionAuthority`);
- The `subscriptionExpires` timestamp, if present, is reached.

7. Subscription Lifecycle

7.1. Activation

For `intent="subscription"`, activation and the first billing-period charge are a single atomic transaction:



When the server receives a Solana "subscription" credential, it **MUST**:

1. Verify the activation transaction matches the challenge: program ID, plan address, token program, mint, puller, recipient, and per-period amount as described in Section 7.1.2.
2. Verify the subscriber identity per Section 7.1.1.
3. Verify and retain the subscription bearer proof per Section 6.2.
4. Sign the transaction as `methodDetails.puller`. When `methodDetails.feePayer` is `true`, also sign as `methodDetails.feePayerKey`, then broadcast.
5. Wait for confirmation and read the resulting on-chain state.
6. Initialize durable subscription state, including the exact bound proof, for later access and renewals.
7. Return `200 OK` with a `Payment-Receipt` for the first charge, including a `subscriptionId` as defined in Section 7.3.

Servers **MUST** treat the subscription as active only after the activation transaction settles successfully and the on-chain `SubscriptionDelegation` account reflects `amount_pulled_in_period == amount_per_period` for period 0.

Servers **MUST NOT** treat activation as successful if `subscriptionExpires` is present and the activation transaction settles at or after that timestamp.

7.1.1. Source Verification

If a credential includes the optional `source` field, servers **MUST NOT** trust this value without verification.

Servers **MUST** determine the payer identity from the activation transaction itself, by extracting the subscriber signer (the signer whose role corresponds to the `delegator` field of the resulting `SubscriptionDelegation`).

If `source` is present, servers **MUST** verify that it uses the `did:pkh:solana:` method and that its address matches the subscriber signer extracted from the activation transaction.

7.1.2. Authorization Scope Verification

When validating a Solana subscription credential, servers **MUST** verify that the activation transaction:

- invokes only the subscriptions program identified by `methodDetails.subscriptionProgram` for subscription instructions, and only the token program identified by `methodDetails.tokenProgram` for token instructions;
- uses `methodDetails.planAddress` as the `Plan` account and uses `methodDetails.puller` as the authorized puller signer;
- uses a receiver ATA whose token-account owner equals `recipient`;
- contains exactly one subscribe instruction and exactly one first-period transfer instruction on the subscriptions program, ordered with subscribe first;
- conditionally contains one subscription-authority initialization instruction, only if the `(subscriber, mint)` authority does not yet exist on-chain;
- contains no other subscriptions-program, system-program, or token-program instructions; compute-budget and memo instructions are permitted.

After the activation transaction settles, servers **MUST** read the resulting `SubscriptionDelegation` account and assert that its snapshotted terms (plan, mint, amount per billing period, mapped per-billing-period interval) equal the corresponding challenge fields, and that the in-period counter equals the per-billing-period amount (i.e., the first-period charge was executed atomically with the subscription's creation).

Servers **MUST** reject activation transactions that pull funds through broader scopes than those required above.

7.2. Renewal

For each later billing period, the server **MAY** submit one `transfer_subscription` transaction using the registered subscription delegation. The transaction **MUST** pull exactly `amount` to a receiver ATA whose token-account owner equals the `recipient` bound at activation.

If the server grants access for a later billing period, it **MUST** ensure that the renewal charge for that period has been collected before, or atomically with, delivering the corresponding service.

Servers **MUST NOT** submit more than one successful renewal charge for the same billing period and **MUST NOT** use the program's partial-transfer capability for this profile.

The on-chain `transfer_subscription` instruction advances `current_period_start_ts` by whole multiples of the period length and resets `amount_pulled_in_period` to zero when the period rolls over. The program then accepts transfers up to the remaining per-period cap. Conforming servers **MUST** use that broader primitive only to collect one transfer of exactly `amount` in the current billing period. If one or more billing periods elapse without a successful charge, a later transaction authorizes only the then-current period's charge; missed periods do not accumulate.

7.3. Subscription Identifier

After successful activation, the server **MUST** return a `subscriptionId` in the `Payment-Receipt`. The value **MUST** be a server-issued opaque base64url string without padding as defined by [I-D.payment-intent-subscription]. The server **MUST** also return the base58 address of the on-chain `SubscriptionDelegation` account in the Solana-specific `subscriptionDelegation` receipt field.

Servers **MUST NOT** include a `Payment-Receipt` header on error responses. On renewal, servers **MUST** return the same `subscriptionId` for the active subscription.

The receipt payload for a Solana subscription:

Field	Type	Description
`method`	string	`"solana"`
`reference`	string	Base58 of the settlement transaction signature
`status`	string	`"success"`
`subscriptionId`	string	Server-issued opaque base64url identifier for the subscription
`subscriptionDelegation`	string	Base58 address of the `SubscriptionDelegation` account
`periodIndex`	number	Zero-based integer index of the billing period (`0` on activation)
`periodStart`	string	[RFC3339] start of the current period
`periodEnd`	string	[RFC3339] end (exclusive) of the current period
`expiresAt`	string	**OPTIONAL**. [RFC3339] effective subscription expiry
`timestamp`	string	[RFC3339] settlement time
`externalId`	string	**OPTIONAL**. Echoed from the challenge request

Table 9

Clients **MAY** retain the `subscriptionId` as application data when referring to the active subscription in later interactions, but the `subscriptionId` is only a receipt identifier unless an application explicitly assigns it additional application-layer meaning.

Clients **MUST** use the bearer proof from Section 6.2 to authenticate later access. Possession or presentation of a `subscriptionId` alone is insufficient.

7.4. Server Accounting and Idempotency

The billing anchor for a Solana subscription is the on-chain `current_period_start_ts` written by `subscribe` when the activation transaction settles. Servers **MUST** derive this anchor from chain settlement data rather than local wall-clock time.

Billing periods are defined as:

- Period 0: `[anchor, anchor + mappedPeriodSeconds)`
- Period 1: `[anchor + mappedPeriodSeconds, anchor + 2*mappedPeriodSeconds)`
- Period N: `[anchor + N*mappedPeriodSeconds, anchor + (N+1)*mappedPeriodSeconds)`

`mappedPeriodSeconds` is `period_hours * 3600` for the on-chain delegation.

Servers **MUST** maintain durable subscription state sufficient to enforce per-period charging rules across retries and concurrent requests. At minimum, servers **MUST** track:

- server-issued subscription identifier
- subscription delegation address
- activation challenge and bound bearer proof
- plan address
- billing anchor
- last successfully charged billing-period index
- any in-flight billing-period index and renewal transaction signature
- subscription expiry
- cancellation timestamp (`delegation.expires_at_ts`; `0` means active, non-zero means cancellation is scheduled, and cancellation is effective once the current time reaches that timestamp)

When granting access in a later billing period, servers **MUST**:

- Verify the payer signature, activation binding, and delegation binding defined in Section 6.2.
- Verify the `SubscriptionDelegation` account exists, is owned by `methodDetails.subscriptionProgram`, has the expected discriminator and version, and still identifies the bound plan and payer.
- Derive the payer's `SubscriptionAuthority` from the payer and mint, verify that it exists and is owned by `methodDetails.subscriptionProgram`, and require its current `init_id` to equal the delegation header's `init_id`. A missing authority or mismatched incarnation invalidates the subscription.

- Verify the subscription is still usable by reading `delegation.expires_at_ts` on-chain; a non-zero timestamp only blocks renewal once the current time is at or after that timestamp.
- Determine the current billing-period index from the anchor and the mapped period in seconds.
- Verify that `delegation.amount_pulled_in_period` is zero before submitting a renewal for the current billing period.
- Atomically record any renewal attempt for the current billing period as in-flight before submitting `transfer_subscription`.
- Mark the current billing period as charged only after the renewal transaction settles successfully and the on-chain `delegation.amount_pulled_in_period` equals `amount`.
- Grant access only after, or atomically with, durably recording the successful renewal charge.

For non-idempotent requests, clients **SHOULD** send an `Idempotency-Key` header per [I-D.ietf-httpapi-idempotency-key-header]. Servers **MUST NOT** charge the same activation or billing period more than once for duplicate idempotent requests.

7.5. Cancellation

Subscribers cancel a subscription by submitting `cancel_subscription` against their `SubscriptionDelegation`. The normal cancellation path sets `delegation.expires_at_ts` to the end of the current billing period, after which `transfer_subscription` fails with `SubscriptionCancelled`. Implementations should also handle program-defined immediate-expiry cases, such as closed or mismatched plan accounts.

Subscribers can additionally revoke every subscription tied to a `(subscriber, mint)` by closing and reopening their `SubscriptionAuthority`. Reopening creates a new authority instance, causing existing subscriptions that reference the prior authority instance to fail transfer validation.

Servers **MUST NOT** submit renewal charges for billing periods after cancellation takes effect. Servers **SHOULD** handle revocation gracefully by returning a fresh subscription challenge once the on-chain delegation has expired.

7.6. Error Responses

When a Solana subscription cannot be used to fulfill a request, the server **MUST** return an appropriate HTTP status code:

Condition	Status Code	Behavior
`subscriptionExpires` present and reached	402 Payment Required	Issue new challenge
On-chain cancellation effective or `SubscriptionAuthority` closed	402 Payment Required	Issue new challenge

Condition	Status Code	Behavior
Current billing period unpaid or `transfer_subscription` failed	402 Payment Required	Issue new challenge
Activation transaction failed verification	402 Payment Required	Issue new challenge
Invalid credential	402 Payment Required	Issue new challenge

Table 10

For all 402 responses, the server **MUST** include a `WWW-Authenticate` header with a fresh challenge. Clients receiving a 402 after a previously valid subscription **SHOULD** treat the subscription as no longer usable and initiate a new subscription flow.

8. Illustrative Lifecycle Examples

This section is non-normative.

8.1. Daily Billing Example

Challenge:

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="qT8wErYuI3oPlKjH6gFdSa",
  realm="api.example.com",
  method="solana",
  intent="subscription",
  expires="2026-01-15T12:05:00Z",
  request="<base64url-encoded JSON below>"
```

The `request` decodes to:

```
{
  "amount": "10000000",
  "currency": "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v",
  "periodUnit": "day",
  "periodCount": "30",
  "subscriptionExpires": "2026-07-14T12:00:00Z",
  "recipient": "9xQeWvG816bUx9EPjHmaT23yvVM2ZWbrrpZb9PusVFin",
  "description": "Monthly Pro plan",
  "externalId": "merchant-subscription-270",
  "methodDetails": {
    "subscriptionProgram": "De1egAFMkMWZSN5rYXRj9CAdheBamobVNubTsi9avR44",
    "planAddress": "8tWbqLkUJoYy7zXc5h2EvCRoaQEv2xnQjUuYhc3rzCgT",
    "mint": "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v",
    "tokenProgram": "TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA",
    "decimals": 6,
    "puller": "5fKb5cF22cFybZB1H4hLDydFhwoQy9JzKzRWaSbMkB6h",
    "network": "mainnet",
    "feePayer": true,
    "feePayerKey": "5fKb5cF22cFybZB1H4hLDydFhwoQy9JzKzRWaSbMkB6h"
  }
}
```

This requests a recurring payment of 10.00 USDC every 30 days until 2026-07-14T12:00:00Z, with the server (5fKb...kB6h) sponsoring transaction fees.

Activation transaction (composed by client):

```
Instruction 0: SetComputeUnitLimit(200000)
Instruction 1: SetComputeUnitPrice(1)
Instruction 2: subscriptions.initialize_subscription_authority
               signer: subscriber
               accounts: [authority_pda, subscriber, mint, ata, ...]
Instruction 3: subscriptions.subscribe
               signer: subscriber
               accounts: [plan_pda, subscription_pda, authority_pda,
                          subscriber, system_program]
Instruction 4: subscriptions.transfer_subscription
               signer: puller
               accounts: [subscription_pda, plan_pda, authority_pda,
                          subscriber_ata, destination_ata, mint,
                          token_program]
Fee payer: 5fKb5cF22cFybZB1H4hLDydFhwoQy9JzKzRWaSbMkB6h (server)
Signatures: subscriber (partial), puller (server, added at co-sign)
Destination ATA owner: 9xQeWvG816bUx9EPjHmaT23yvVM2ZWbrrpZb9PusVFin
```

Credential:

```
{
  "challenge": {
    "id": "qT8wErYuI3oPlKjH6gFdSa",
    "realm": "api.example.com",
    "method": "solana",
    "intent": "subscription",
    "request": "eyJ...",
    "expires": "2026-01-15T12:05:00Z"
  },
  "payload": {
    "type": "transaction",
    "transaction": "AQAAAA...base64 of partially signed tx...",
    "authentication": {
      "type": "proof",
      "challengeId": "qT8wErYuI3oPlKjH6gFdSa",
      "payer": "7YWHMfk9JZe0LMQDBiTkBXK9LmJRxB6Kuf7nBzvuN7tP",
      "signature": "4vJ9...base58 Ed25519 signature...Qd"
    }
  }
}
```

Later access credential:

```
{
  "challenge": {
    "id": "qT8wErYuI3oPlKjH6gFdSa",
    "realm": "api.example.com",
    "method": "solana",
    "intent": "subscription",
    "request": "eyJ...",
    "expires": "2026-01-15T12:05:00Z"
  },
  "payload": {
    "type": "proof",
    "subscriptionDelegation": "BXQGmO5VwTrl5RfFr6Y8XQZ4nPj9QqMOiKkRn3pZ4ZE",
    "authentication": {
      "type": "proof",
      "challengeId": "qT8wErYuI3oPlKjH6gFdSa",
      "payer": "7YWHMfk9JZe0LMQDBiTkBXK9LmJRxB6Kuf7nBzvuN7tP",
      "signature": "4vJ9...base58 Ed25519 signature...Qd"
    }
  }
}
```

Receipt:

If activation settles at `2026-01-15T12:03:10Z`, the `Payment-Receipt` payload decodes to:

```
{
  "method": "solana",
  "intent": "subscription",
  "status": "success",
  "reference": "5J8...base58 transaction signature...Kt",
  "subscriptionId": "c3ViX3NvbGFuYV8wMQ",
  "subscriptionDelegation": "BXQGmO5VwTrl5RfFr6Y8XQZ4nPj9QqMOiKkRn3pZ4ZE",
  "externalId": "merchant-subscription-270",
  "periodIndex": 0,
  "periodStart": "2026-01-15T12:03:10Z",
  "periodEnd": "2026-02-14T12:03:10Z",
  "expiresAt": "2026-07-14T12:00:00Z",
  "timestamp": "2026-01-15T12:03:10Z"
}
```

The resulting billing periods are:

- Period 0: `[2026-01-15T12:03:10Z, 2026-02-14T12:03:10Z)`
- Period 1: `[2026-02-14T12:03:10Z, 2026-03-16T12:03:10Z)`
- Period 2: `[2026-03-16T12:03:10Z, 2026-04-15T12:03:10Z)`

Requests during Period 0 do not require another renewal charge. When Period 1 begins, the server's renewal worker submits one `transfer_subscription` instruction to the subscriptions program. On success, the on-chain delegation advances to `current_period_start_ts = 2026-02-14T12:03:10Z` and `amount_pulled_in_period = 10000000`. After that renewal succeeds, additional requests during Period 1 do not permit another charge for Period 1.

8.2. Cancellation Example

Suppose Period 2 has been charged and the subscriber cancels by submitting `cancel_subscription` on-chain at `2026-03-20T09:00:00Z`. The program sets `delegation.expires_at_ts = 2026-04-15T12:03:10Z` (end of Period 2).

Cancellation takes effect at the end of the current paid billing period. The server continues honoring access through that time. The server's renewal worker **MUST NOT** submit a Period 3 charge: the on-chain `transfer_subscription` would fail with `SubscriptionCancelled`.

Requests at or after `2026-04-15T12:03:10Z` receive `402 Payment Required` with a fresh challenge.

8.3. Failed Renewal Example

If the subscriber's token balance is insufficient when the server attempts a Period 3 renewal, the on-chain transaction fails. The server records the failure but does not retry continuously; it returns `402 Payment Required` on subsequent gated requests.

If a later retry within Period 3 succeeds, the server may grant access for Period 3 and update its local state.

If Period 4 begins before any successful Period 3 charge, the next successful `transfer_subscription` collapses to Period 4 (the program advances `current_period_start_ts` to Period 4's start and resets `amount_pulled_in_period = 0`). The skipped Period 3 charge does not become additional on-chain spending authority.

8.4. Expiry Example

When `subscriptionExpires = 2026-07-14T12:00:00Z` is reached, the server stops submitting renewal transactions. A subsequent `transfer_subscription` would still succeed on-chain (the program itself does not enforce `subscriptionExpires`; it is a server-side contract derived from the credential), so the server **MUST** honor the expiry by ceasing renewal submissions and serving fresh challenges.

Requests after that time receive `402 Payment Required` with a fresh challenge.

If the server sponsored activation, it can reclaim rent only when program revocation rules allow `revoke_delegation`, such as after on-chain cancellation has taken effect or the referenced plan has ended or been closed.

9. Security Considerations

9.1. Recurring Charge Awareness

Clients **MUST** clearly communicate that a Solana subscription authorizes future recurring on-chain transfers without requiring a new user action for each billing period. Wallets and client UIs **SHOULD** display the per-period amount, the period length, and the `subscriptionExpires` value (when present) at activation time.

9.2. Amount and Period Verification

Clients **MUST** parse and verify the `request` payload before signing:

1. Verify `amount` is reasonable for the service
2. Verify `currency` is the expected mint
3. Verify `periodUnit` and `periodCount` match expectations
4. Verify `recipient` is controlled by the expected party
5. Verify `subscriptionExpires` is acceptable when present
6. Verify the on-chain `Plan` referenced by `methodDetails.planAddress` carries matching mint, per-period amount, and per-billing-period interval; authorizes the server's puller as the plan owner or in `plan.pullers`; and has exactly one non-zero destination equal to `recipient`
7. Verify the activation `transfer_subscription` instruction uses a receiver ATA whose token-account owner equals `recipient`

Clients **MUST NOT** sign an activation transaction whose on-chain `Plan` does not match the challenge. Clients **MUST NOT** rely on the `description` field for payment verification.

9.3. Duplicate Charge Prevention

On-chain per-period accounting prevents overspending within a billing period, but it does not by itself make HTTP service delivery idempotent. Servers **MUST** implement durable local state to prevent duplicate activation and renewal charges caused by retries, parallel requests, or races between charging and service delivery.

Servers **SHOULD** use a per-subscription lease (e.g., a database row lock with a short TTL) when submitting a renewal, releasing the lease only after the on-chain state has advanced.

9.4. Server Accountability

Servers operating Solana subscriptions are responsible for:

- Secure storage of subscription state and any sponsoring fee-payer keys;
- Not charging more than once per billing period;
- Honoring on-chain cancellation and revocation promptly;
- Providing transaction signatures and billing records to payers on request.

9.5. Caching

Responses to subscription challenges (402 Payment Required) **MUST** include `Cache-Control: no-store` to prevent sensitive payment data from being cached by intermediaries.

Responses containing `Payment-Receipt` headers **MUST** include `Cache-Control: private` to prevent shared caches from storing payment receipts.

9.6. Bearer Proof Theft

The subscription authentication proof is intentionally reusable. A party that obtains it can impersonate the subscriber within the server's protection space until the subscription or an additional application-layer authorization expires or is revoked. Domain separation and the activation challenge bind the proof to one subscription and server realm, but they do not make a copied proof non-transferable.

Servers **MUST** re-check subscription usability for every presentation; signature verification alone is insufficient. Servers **SHOULD** support early application-layer proof revocation when immediate logout or credential rotation is required.

9.7. Destination Scoping

Solana subscription delegations **MUST** be bound to a receiver identified by `recipient`. Clients and servers **MUST** reject credentials whose activation transaction uses a receiver ATA owned by any other address. Servers **MUST** use a receiver ATA owned by the same recipient for every renewal.

9.8. Plan Scope Minimization

`plan.destinations` is a whitelist, not a payout split. The underlying program permits an empty whitelist or several destinations, but this profile does not. The non-zero destination set **MUST** contain only `recipient`, making the recurring recipient restriction immutable and program-enforced for the lifetime of the plan.

9.9. Subscription Authority Isolation

The `SubscriptionAuthority` is shared across every subscription the same payer holds for the same mint. Closing the authority invalidates every delegation created against it, terminating all subscriptions tied to that authority simultaneously.

Servers **MUST NOT** rely on long-term existence of a particular authority instance beyond the lifetime of their own delegations. Applications **SHOULD** expose authority closure only as an explicit "revoke all subscriptions for this token" affordance, never as a side effect of routine operations.

9.10. Token-2022 Extension Policy

Implementations **MUST** validate Token-2022 mints and token accounts at activation and before each token-touching instruction according to the deployed validation policy of the subscriptions program.

The canonical subscriptions program rejects Token-2022 mints with a configured `TransferHook` extension. An inert `TransferHook` extension whose authority and program id are both unset is allowed, because it cannot be activated later.

Implementations **MUST NOT** resolve transfer-hook extra accounts for configured hooks or otherwise bypass the mint and token account validation performed by the program.

9.11. Account Ownership and Program-ID Validation

Before deserializing or mutating any account, implementations **MUST** validate the expected program owner for:

- the `Plan`, `SubscriptionDelegation`, and `SubscriptionAuthority` PDAs (owned by the subscriptions program);
- the subscriber and destination token accounts (owned by `methodDetails.tokenProgram`);
- the mint (owned by `methodDetails.tokenProgram`).

Implementations **MUST NOT** allow user-controlled program accounts to influence the activation, transfer, or revocation paths.

10. IANA Considerations

10.1. Payment Intent Registration

The `subscription` payment intent is registered by [I-D.payment-intent-subscription]. This document does not register it again.

11. References

11.1. Normative References

[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/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[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/info/rfc8785>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/info/rfc9457>.

[I-D.httpauth-payment] Moxey, J., "The 'Payment' HTTP Authentication Scheme", January 2026, <https://datatracker.ietf.org/doc/draft-ryan-httpauth-payment/>.

[I-D.payment-intent-subscription] Moxey, J., Ryan, B., and T. Meagher, "Subscription Intent for HTTP Payment Authentication", April 2026, <https://datatracker.ietf.org/doc/draft-payment-intent-subscription/>.

[I-D.ietf-httpapi-idempotency-key-header] Jena, J., "The Idempotency-Key HTTP Header Field", June 2024, <https://datatracker.ietf.org/doc/draft-ietf-httpapi-idempotency-key-header/>.

11.2. Informative References

[SOLANA-DOCS] Solana Foundation, "Solana Documentation", 2026, <https://solana.com/docs>.

[SPL-TOKEN] Solana Foundation, "SPL Token Program", 2026, <https://solana.com/docs/tokens>.

[SPL-TOKEN-2022] Solana Foundation, "SPL Token-2022 Program", 2026, <https://solana.com/docs/tokens/extensions>.

[BASE58] Sporny, M., "Base58 Encoding Scheme", 2023, <https://datatracker.ietf.org/doc/html/draft-msporny-base58-03>.

[SUBSCRIPTIONS-PROGRAM] Solana Foundation, "Subscriptions Solana Program", 2026, <https://github.com/solana-program/subscriptions>.

Appendix A. Acknowledgements

The authors thank the MPP community and the Tempo team for their feedback on this specification, and the Solana Foundation subscriptions program team for the on-chain primitives this profile builds on.

Authors' Addresses

Ludo Galabru
Solana Foundation
Email: ludo.galabru@solana.org

Jo D.
Solana Foundation
Email: jo.desormeaux@solana.org