Workgroup: Network Working Group
Internet-Draft: draft-usdc-charge-00
Published: 16 June 2026
Intended Status: Informational
Expires: 18 December 2026
Authors:
H. Bhangale
Circle Internet Group, Inc.
H. Gu
Circle Internet Group, Inc.
B. Agarwal
Circle Internet Group, Inc.

"usdc" Payment Method for HTTP Payment Authentication

Abstract

This document defines the `usdc` payment method for the `charge` intent in the Payment HTTP Authentication Scheme [I-D.httpauth-payment]. It gives merchants one USDC acceptance surface across supported chain families while leaving chain-specific signing and broadcast mechanics in the relevant chain profile.

This version covers direct USDC charges on EVM and Solana by profiling the existing PaymentAuth EVM and Solana charge specifications. The EVM profile is intentionally limited to EIP-3009 authorization credentials in v00. It also defines a direct USDCx on Stacks profile because USDCx on Stacks is backed by USDC through xReserve and is not covered by a generic MPP chain method today.

This version also defines a Gateway Transfer charge profile for cross-chain USDC payments through Circle Gateway. The merchant chooses the destination chain where it wants to receive USDC and advertises the Gateway source chains it accepts from payers.

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 18 December 2026.

Copyright Notice

Table of Contents

1. Introduction

HTTP Payment Authentication [I-D.httpauth-payment] defines a challenge-response mechanism that gates access to resources behind payments. This document defines `method="usdc"` for settled `intent="charge"` payments.

The method is a USDC-specific acceptance surface, not a new generic chain method. Direct EVM USDC is mechanically close to `method="evm"` with USDC selected, and direct Solana USDC is close to `method="solana"` with the USDC SPL token mint in `request.currency`. The value of this method is the merchant-facing USDC abstraction and the USDC-specific rules around native USDC issuance, supported asset forms, third-party lookalike assets, token controls, and receipts.

This version keeps the base direct charge path small. Gateway Transfer is an optional cross-chain profile for merchants that want USDC on one destination chain while accepting payer funds from any advertised Gateway source chain.

1.1. Scope of This Version

Normatively specified:

- EVM direct USDC charges, by reference to [I-D.evm-charge].
- Solana direct USDC charges, by reference to [I-D.solana-charge].
- Stacks direct USDCx charges using SIP-010 transfers.
- Gateway Transfer charges through Circle Gateway that settle on the merchant's selected destination network before a successful receipt is returned.
- USDC-specific asset identity, token-control, replay, and receipt requirements.

1.2. Relationship to Other Methods

This document does not replace the EVM or Solana charge methods. It profiles them for USDC.

For EVM, the request and credential envelope inherits from [I-D.evm-charge]. The `usdc` profile restricts the token to native USDC, restricts the v00 credential payload to EIP-3009 `authorization`, and adds USDC-specific asset identity and control checks. Deployments that want Permit2, raw transaction, or hash-based EVM settlement SHOULD advertise `method="evm"` directly.

For Solana, the request and credential semantics inherit from [I-D.solana-charge]. The `usdc` profile restricts the token to the native USDC SPL mint published by Circle and the legacy SPL Token program in v00.

For Stacks, this document defines USDCx on Stacks directly because it is not covered by a generic MPP chain method today.

For Gateway Transfer, this document defines a Circle Gateway-specific cross-chain charge profile. The merchant sets the destination network where it wants to receive USDC. The payer chooses one of the advertised source networks, signs a Gateway authorization, and the server submits that authorization to Circle Gateway. A successful charge receipt means Gateway has completed settlement on the merchant's destination network.

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.

3. Terminology

Native USDC USDC natively issued by Circle on a supported network. Circle publishes the contract and mint addresses for each supported deployment.

USDCx on Stacks The SIP-010 token issued by the partner-deployed USDCx on Stacks contract and backed 1:1 by USDC deposited into a Circle xReserve smart contract on supported source chains. USDCx is not native USDC, so this profile treats it as a distinct asset form.

Direct Charge A payment where the server returns a successful receipt only after the underlying chain transaction has reached the server's local confirmation threshold.

Gateway Transfer A charge profile where Circle Gateway moves native USDC from a payer-selected source network to the merchant's selected destination network before a successful receipt is returned.

4. Method Identifier

The payment method identifier is:

`text usdc`

5. Supported Intents

This v00 document supports:

`text charge`

6. Intent: "charge"

For `intent="charge"`, the `request` auth-param contains the fields defined by the charge intent plus `usdc` method details. The request JSON MUST be serialized with JSON Canonicalization Scheme [RFC8785] before base64url encoding.

6.1. Shared Request Fields

Field	Type	Required	Description
`amount`	string	REQUIRED	Positive integer amount in USDC base units.
`currency`	string	REQUIRED	Profile-specific USDC token identifier. Direct profiles use chain-native identifiers. Gateway Transfer uses the literal `usdc` identifier.

Field	Type	Required	Description
`recipient`	string	REQUIRED	Chain-native recipient identifier.
`description`	string	OPTIONAL	Human-readable payment description.
`externalId`	string	OPTIONAL	Merchant reference identifier.

Table 1

For EVM, `currency` follows [I-D.evm-charge] and is the native USDC token contract address published by Circle [CIRCLE-USDC-ADDRESSES].

For Solana, `currency` follows [I-D.solana-charge] and is the native USDC mint address published by Circle [CIRCLE-USDC-ADDRESSES].

For Stacks, `currency` MUST be the full USDCx SIP-010 [SIP-010] asset identifier, `<contractAddress>.<contractName>::<assetName>`. `methodDetails.stacks` carries the same identity as parsed fields for transaction verification.

For Gateway Transfer, `currency` MUST be the case-sensitive literal `usdc`. This is a method-defined asset identifier, not an ISO currency code. The concrete source and destination token identities are resolved through Circle Gateway discovery and estimate APIs or a conforming SDK.

6.2. Method Details

`methodDetails.type` selects the active `usdc` profile. Its value MUST be one of `evm`, `solana`, `stacks`, or `gateway`.

The `methodDetails` object MUST include exactly one profile details object, and that object MUST use the same name as `methodDetails.type`. For example, when `methodDetails.type = "solana"`, `methodDetails.solana` MUST be present and `methodDetails.evm`, `methodDetails.stacks`, and `methodDetails.gateway` MUST be absent.

Field	Type	Required	Description
`type`	string	REQUIRED	One of `evm`, `solana`, `stacks`, or `gateway`.
`evm`	object	CONDITIONAL	EVM details. Required for EVM direct USDC charge.
`solana`	object	CONDITIONAL	Solana details. Required for Solana direct USDC charge.
`stacks`	object	CONDITIONAL	Stacks details. Required for USDCx on Stacks charge.
`gateway`	object	CONDITIONAL	Circle Gateway Transfer details. Required for Gateway Transfer charge.

Table 2

This `methodDetails.type` discriminator with a nested per-profile object is intentionally not the flat `methodDetails` shape used by [I-D.evm-charge] and [I-D.solana-charge]. A parser written for the base specs' flat `methodDetails` will not work here; implementations MUST select the active profile object by `methodDetails.type`.

Identifier formats otherwise follow each profile's base specification. The direct EVM and Solana profiles use the payer and recipient identifier formats defined by [I-D.evm-charge] and [I-D.solana-charge]. CAIP-2 [CAIP-2] network identifiers and CAIP-10 [CAIP-10] account identifiers are used normatively in the Gateway Transfer profile, where one charge can route across chain families and a chain-native identifier alone would be ambiguous, and in the Stacks profile, which has no base MPP chain method to inherit from. The receipt `network` field is a single, deliberate cross-profile identifier described in Section 10.

6.3. EVM Profile

The EVM profile inherits [I-D.evm-charge]. The following restrictions apply:

Field	Type	Required	Description
`chainId`	number	REQUIRED	Decimal EVM chain identifier.
`decimals`	number	REQUIRED	MUST be 6.
`credentialTypes`	array	OPTIONAL	If present, MUST contain only `authorization` in v00. If absent, `authorization` is implied.

Table 3

Servers MUST verify that `request.currency` is the native USDC token contract published by Circle for `methodDetails.evm.chainId`. EVM credentials for this profile MUST use `payload.type="authorization"`. The authorization nonce MUST bind the selected challenge so a signed authorization cannot be replayed across payment challenges or intents. Servers MUST verify the EIP-3009 signature against the token contract's actual EIP-712 domain. For v00, the token domain MUST match native USDC for the selected chain and contract, including `chainId = methodDetails.evm.chainId` and `verifyingContract = request.currency`. Implementations MAY discover the domain through `eip712Domain()` where available, `DOMAIN_SEPARATOR`, or a trusted native-USDC registry. Deployments that require an alternate EIP-712 domain shape, including salt-based domains, are out of scope for this profile. Because the client signs an offchain EIP-3009 authorization, the server submits the transaction and pays EVM gas.

6.4. Solana Profile

The Solana profile inherits [I-D.solana-charge]. The following restrictions apply:

Field	Type	Required	Description
`network`	string	REQUIRED	`mainnet`, `devnet`, or `localnet`.

Field	Type	Required	Description
`decimals`	number	REQUIRED	MUST be 6.
`tokenProgram`	string	REQUIRED	MUST be the legacy SPL Token program ID in v00.
`feePayer`	boolean	OPTIONAL	Whether the server pays network fees.
`feePayerKey`	string	CONDITIONAL	Required when `feePayer=true`; absent otherwise.

Table 4

v00 profiles native USDC on the legacy SPL Token program only. Servers MUST reject credentials whose `tokenProgram` is not:

```
text TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA
```

The Token-2022 program is out of scope in v00. The field remains in the schema so a later revision can admit Token-2022 USDC without changing the wire shape.

Servers SHOULD verify that `tokenProgram` equals the owner program of the mint account returned by Solana RPC. A mismatch MUST cause credential rejection.

Solana credentials for this profile MUST use `payload.type="transaction"` and inherit base Solana pull-mode verification from [I-D.solana-charge]. The credential echoes the challenge, and the server binds settlement to it through challenge consumption and transaction-signature replay protection as defined by [I-D.solana-charge].

Deployments that need third-party verifier or facilitator proof MAY define a stricter challenge-bound authorization profile in a later version.

6.5. Stacks Profile

Stacks uses SIP-005 [SIP-005] consensus-serialized transactions.

Unlike EVM and Solana, Stacks has no base MPP chain method to inherit from, so this document defines server-broadcast transaction verification directly. The server verifies the SIP-010 transfer, its post-condition, and the origin signature, then binds settlement to the challenge through challenge consumption and transaction-id replay protection.

Field	Type	Required	Description
`network`	string	REQUIRED	`mainnet` or `testnet`.
`chainId`	string	REQUIRED	Decimal Stacks chain id, `1` for mainnet or `2147483648` for testnet.

Field	Type	Required	Description
`contractAddress`	string	REQUIRED	Stacks standard principal for the USDCx SIP-010 token contract.
`contractName`	string	REQUIRED	Contract name component of the SIP-010 token contract.
`assetName`	string	REQUIRED	Fungible-asset identifier inside the SIP-010 contract.
`functionName`	string	REQUIRED	MUST be `transfer`.
`decimals`	number	REQUIRED	MUST be 6.
`feePayer`	boolean	OPTIONAL	Whether the server sponsors fees using Stacks sponsored transaction authorization.
`feePayerAddress`	string	CONDITIONAL	Required when `feePayer=true`; absent otherwise.

Table 5

The USDCx on Stacks mainnet token identity at publication time is:

```
text SP120SBRBQJ00MCWS7TM5R8WJNTTKD5K0HFRC2CNE.usdcx::usdcx-token
```

The USDCx on Stacks testnet token identity used by the examples is:

```
text ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.usdcx::usdcx-token
```

Servers MUST verify the advertised token tuple against the Circle xReserve registry or an explicit implementation allowlist. Until a public registry is available, a v00 allowlist entry MUST include `network`, `chainId`, `contractAddress`, `contractName`, `assetName`, `decimals`, and the xReserve control surface used for issuance and redemption checks. A token identifier that appears only in a partner-published registry MUST NOT be accepted if it contradicts the Circle xReserve registry or allowlist. The parsed `methodDetails.stacks` tuple MUST match `request.currency`.

6.6. Gateway Transfer Profile

The Gateway Transfer profile uses Circle Gateway to move native USDC from a payer-selected Gateway source network to the merchant's chosen destination network. The merchant receives USDC on `destinationNetwork`, so the payer's source-network choice is constrained by `acceptedSources` rather than by the merchant's settlement network.

Field	Type	Required	Description
`acceptedSources`	array	REQUIRED	CAIP-2 Gateway source networks the payer may use for this charge.
`destinationNetwork`	string	REQUIRED	CAIP-2 destination network where the merchant wants to receive USDC.
`maxFee`	string	REQUIRED	Absolute upper bound on the signed Gateway authorization fee cap, in USDC base units.
`maxFeeBps`	number	OPTIONAL	Additional ratio cap. When present, `authorizationFeeCap * 10000 <= request.amount * maxFeeBps`.
`credentialTypes`	array	OPTIONAL	If present, MUST contain only `transfer` in v00. If absent, `transfer` is implied.

Table 6

Gateway network fields use CAIP-2 [CAIP-2] identifiers. Gateway account identifiers use CAIP-10 [CAIP-10] account IDs in the form `<CAIP-2 network>:<account address>`.

Circle Gateway maps source and destination networks to Gateway domains, token identifiers, wallet contracts, minter contracts, recipient setup options, and signing bytes. Those details are resolved by Circle Gateway APIs or a conforming SDK. They are not carried in `methodDetails`.

v00 allows any Circle Gateway-supported source and destination pair, including EVM to Solana, Solana to EVM, and same-family transfers. Servers MUST reject credentials for a route that Circle Gateway does not support at authorization time.

For v00, a Gateway Transfer credential selects one source network from `acceptedSources`. `acceptedSources` is the merchant's accepted set; `payload.sourceNetwork` is the payer's selected source for this charge.

For Solana destinations, `request.recipient` identifies the merchant's Solana owner address. The signed Gateway authorization settles to a USDC token account for that owner. Associated token account creation and recipient setup options are Gateway authorization fields, not PaymentAuth `methodDetails`.

`maxFee` is separate from `amount`. It is a cap, not the fee paid. The payer pays Gateway Transfer fees from the source depositor's Gateway balance, in addition to the merchant amount. The server MUST reject a Gateway authorization whose fee cap exceeds `methodDetails.gateway.maxFee`.

If `maxFeeBps` is present, the signed Gateway authorization fee cap MUST also satisfy the ratio cap: `authorizationFeeCap * 10000 <= request.amount * maxFeeBps`. When both caps are present, the signed fee cap MUST satisfy both.

The Gateway TransferSpec value MUST equal `request.amount`. Circle Gateway validates that the source depositor has enough Gateway balance to cover the TransferSpec value plus the fee charged for the accepted transfer. Clients SHOULD present `amount + maxFee` as the payer's worst-case spend.

Servers verify the Gateway authorization against the PaymentAuth request before submission.

This draft relies on Circle Gateway for route discovery [CIRCLE-GATEWAY-INFO], fee estimation [CIRCLE-GATEWAY-ESTIMATE], signing material and submission [CIRCLE-GATEWAY-TRANSFER], transfer status [CIRCLE-GATEWAY-TRANSFER-STATUS], and TransferSpec lookup [CIRCLE-GATEWAY-TRANSFER-SPEC]. It does not copy Gateway attestation bytes, contract ABI details, or SDK routing tables into the PaymentAuth request. Circle Gateway validates Gateway encoding, signatures, Gateway replay, route support, fee calculation, and transfer validity.

6.6.1. Gateway Transfer Salt Binding

Each Gateway TransferSpec used for this profile MUST carry a challenge-bound `salt`. In v00, the salt binding is:

```
text keccak256(UTF-8 bytes of JCS({ "id": "CHALLENGE_ID", "method": "usdc",
"realm": "CHALLENGE_REALM", "intent": "charge", "type": "gateway", "requestHash":
"REQUEST_HASH", "sourceNetwork": "SELECTED_SOURCE_NETWORK", "destinationNetwork":
"DESTINATION_NETWORK", "sourceDepositor": "SOURCE_DEPOSITOR_ACCOUNT",
"sourceSigner": "SOURCE_SIGNER_ACCOUNT", "recipient": "REQUEST_RECIPIENT",
"destinationRecipient": "DESTINATION_RECIPIENT_ACCOUNT", "amount":
"TRANSFER_SPEC_VALUE", "maxFee": "GATEWAY_AUTHORIZATION_FEE_CAP" }))
```

`requestHash` is `keccak256` of the UTF-8 bytes of the exact JCS-canonicalized request JSON before base64url encoding. The server MUST recompute this binding before treating the Gateway authorization as valid for the selected challenge.

All account values in the salt preimage use PaymentAuth-normalized strings, not raw Gateway ABI byte strings. `sourceDepositor` is the exact `credential.source` CAIP-10 account after normal CAIP-10 validation. `sourceSigner` is the Gateway account whose signature authorizes the transfer. It MAY equal `sourceDepositor` or be an account that Circle Gateway accepts as an authorized delegate for that depositor. `destinationRecipient` is the chain account that receives the settled funds, encoded as CAIP-10 on `destinationNetwork`. For EVM networks, address comparisons and CAIP-10 account strings use the lowercase 20-byte `0x` address. For Solana networks, account strings use the base58 public key.

If the Gateway authorization carries recipient setup options, the PaymentAuth verifier MUST confirm that those options are bound to `request.recipient` before submitting the authorization. For Solana destinations, this means any associated-token-account setup must use `request.recipient` as the token account owner.

The PaymentAuth verifier MUST inspect the Gateway authorization material before submitting it to Circle Gateway. The verifier MUST be able to read, either directly or through a conforming SDK, the signed Gateway fields needed by Section 8. It MUST submit the same signed authorization package it inspected. Because `salt` is part of the signed Gateway TransferSpec, changing the salt changes the Gateway authorization and causes Circle Gateway validation to fail. The client does not supply `transferSpecHash` as a credential field; Circle Gateway returns it later as receipt evidence for the transfer item that was accepted.

7. Credential Schema

EVM credentials use [I-D.evm-charge] authorization payloads, except for the nonce derivation, which this profile overrides as defined below. Solana credentials inherit [I-D.solana-charge] pull-mode transaction payloads. Stacks and Gateway Transfer define their profile-specific payloads below.

For EVM authorization credentials, the EIP-3009 fields are carried directly in `payload` as defined by [I-D.evm-charge]. `payload.nonce` MUST be a `0x`-prefixed lowercase zero-padded 66-character hex string representing exactly 32 bytes. The nonce MUST equal:

```
text keccak256(UTF-8 bytes of JCS({ "id": "CHALLENGE_ID", "method": "usdc",
"realm": "CHALLENGE_REALM", "intent": "charge", "requestHash": "REQUEST_HASH" }))
```

`requestHash` is `keccak256` of the UTF-8 bytes of the exact JCS-canonicalized request JSON before base64url encoding.

This nonce derivation overrides, and does not inherit, the base EVM charge profile's derivation in [I-D.evm-charge]. The base EVM profile binds EIP-3009 authorization nonces to `challenge.id` and `challenge.realm`. The `usdc` profile also binds the method, intent, and request hash, so an authorization cannot move between `evm`, `usdc`, `charge`, another intent, or a different request with the same amount and recipient. Because the derivation differs, a generic `method="evm"` verifier computes a different expected nonce and will reject these credentials; `usdc` EVM credentials are therefore not interchangeable with `method="evm"` credentials.

When present, the EVM credential `source` follows [I-D.evm-charge] and is OPTIONAL; the RECOMMENDED form is `did:pkh:eip155:<chainId>:<address>`.

For Solana transaction credentials, the credential object contains:

Field	Type	Required	Description
`challenge`	object	REQUIRED	Echo of the server challenge.
`payload`	object	REQUIRED	Solana payment payload.

Field	Type	Required	Description
`source`	string	OPTIONAL	Payer account, per [I-D.solana-charge]. MAY be a base58 public key or a DID (RECOMMENDED `did:pkh:solana:<genesis-hash-prefix>:<pubkey>`).

Table 7

The Solana `payload.type` MUST be `transaction`. The payload carries:

Field	Type	Required	Description
`type`	string	REQUIRED	MUST be `transaction`.
`transaction`	string	REQUIRED	Base64-encoded serialized Solana transaction.

Table 8

For Stacks, the credential object contains:

Field	Type	Required	Description
`challenge`	object	REQUIRED	Echo of the server challenge.
`payload`	object	REQUIRED	Payment payload.
`source`	string	REQUIRED	CAIP-10 account ID using `stacks:<chainId>:<standard-principal>`.

Table 9

The Stacks `source` principal MUST be a c32check-encoded standard principal. Contract principals MAY appear as `request.recipient`, but MUST NOT appear as `source`.

The Stacks `payload.type` MUST be `transaction`. The payload carries:

Field	Type	Required	Description
`type`	string	REQUIRED	MUST be `transaction`.
`transaction`	string	REQUIRED	Base64-encoded Stacks consensus-serialized transaction.
`transactionFormat`	string	OPTIONAL	MUST be `stacks_transaction_v1` when present.

Table 10

The Stacks transaction MUST call the SIP-010 `transfer` function with `amount`, `sender`, `recipient`, and optional memo arguments. It MUST include a post-condition that pins a `SentEq` transfer of `request.amount` for the advertised USDCx asset.

When present, `transactionFormat` MUST be `stacks_transaction_v1`. For Stacks, origin and sponsor signatures are carried in the transaction auth field, not in a separate credential field.

7.1. Gateway Transfer Credential

For Gateway Transfer, the credential object contains:

Field	Type	Required	Description
`challenge`	object	REQUIRED	Echo of the server challenge.
`payload`	object	REQUIRED	Gateway Transfer payload.
`source`	string	REQUIRED	CAIP-10 source depositor account ID.

Table 11

The Gateway Transfer `payload.type` MUST be `transfer`. This value is scoped to `methodDetails.type="gateway"`. The payload carries the selected source for this charge. It does not repeat the full `acceptedSources` list from the request.

Field	Type	Required	Description
`type`	string	REQUIRED	MUST be `transfer`.
`sourceNetwork`	string	REQUIRED	Selected CAIP-2 source network. MUST be one entry from `methodDetails.gateway.acceptedSources`.
`destinationNetwork`	string	REQUIRED	CAIP-2 destination network.
`maxFee`	string	REQUIRED	Maximum Gateway fee authorized by the payer, in USDC base units.
`authorization`	object	REQUIRED	Circle Gateway signed authorization package, or SDK-produced versioned object, for the selected source network.

Table 12

The Gateway authorization package represents one Gateway transfer for the selected source network. Its internal encoding is Circle Gateway versioned data, not a PaymentAuth extension point. When this document names Gateway fields such as source depositor, source signer, TransferSpec value, fee cap, destination recipient, recipient setup options, or salt, it refers to

their Gateway semantics. The wire object MAY carry those values under versioned Gateway field names. The authorization object MUST expose the signed fields required by Section 8 to the verifier.

The surrounding PaymentAuth credential binds that Gateway package to `credential.challenge.id` and `credential.challenge.realm` by requiring the signed TransferSpec salt to equal the challenge-bound salt defined in Section 6.6.1. `sourceSigner` is the account that signs the Gateway authorization. It can be the same account as `sourceDepositor`, or an account Circle Gateway accepts as an authorized delegate for that depositor. If `sourceSigner` is a contract account, Circle Gateway also applies its contract-signature validation rules.

8. Verification Procedure

All profiles MUST perform these common checks before chain-specific verification:

1. Decode and JCS-verify the request and credential envelopes.
2. Verify `method="usdc"` and `intent="charge"`.
3. Verify `credential.challenge` matches the selected challenge.
4. Verify the challenge has not expired.
5. Verify `methodDetails.type` is present, exactly one profile object is present, and the present object key matches `methodDetails.type`.
6. Verify `amount` is a positive integer in base units.
7. Verify the selected token is a supported USDC asset form for the selected profile.
8. Verify the payer and recipient are not present in the applicable blocklist.
9. Verify the token contract or mint is not paused where that control exists.
10. Verify replay protection for the selected credential type.

For USDCx on Stacks, the applicable controls include the Circle xReserve controls and any partner-chain token controls required by local policy.

For EVM, servers then apply [I-D.evm-charge] verification with the USDC restrictions in Section 6.3. Servers MUST verify `methodDetails.type = "evm"`. For `payload.type="authorization"`, the EIP-3009 fields are carried directly in `payload` as defined by [I-D.evm-charge]. The server MUST verify `payload.to` and `request.recipient` identify the same 20-byte EVM address, `payload.value = request.amount`, and `payload.nonce` equals the challenge-bound nonce derivation before it submits the authorization. EVM address equality is byte equality after hexadecimal decoding, not case-sensitive string equality.

For Solana, servers then apply [I-D.solana-charge] verification with the USDC restrictions in Section 6.4. Servers MUST verify `methodDetails.type = "solana"`. If `methodDetails.solana.feePayer=true`, the transaction MUST set `methodDetails.solana.feePayerKey` as fee payer and the only missing required signature MUST be the server fee-payer signature. The server MUST reject any Solana credential whose transaction bytes have already been consumed. The server MUST reject stale transactions by verifying that the transaction uses a currently valid recent blockhash.

USDC Solana verification is intentionally narrower than the generic Solana transaction profile. Servers MUST reject transactions with instructions outside the allowed set for this profile: SPL Token transfer instructions for the advertised mint, associated-token-account setup for the advertised recipient and mint when needed, bounded Compute Budget instructions, and optional Memo instructions. Token-2022, delegate authority, and multisig authority flows are out of scope in v00. Servers MUST verify that the source token account is owned by the transfer authority, that the recipient token account is the associated token account for `request.recipient` and `request.currency` unless an equivalent explicit token account is allowed by local policy, and that the token transfer amount equals `request.amount`. An equivalent explicit token account MUST be initialized for `request.currency` and owned by `request.recipient`.

When `feePayer=true`, the server MUST simulate the final transaction after adding the fee-payer signature and MUST reject transactions whose compute units, account writes, or fee exposure exceed local policy.

For Stacks, servers MUST verify:

1. `payload.type = "transaction"`.
2. `methodDetails.type = "stacks"`.
3. `source` uses `stacks:<chainId>:<standard-principal>` and the chain id matches `methodDetails.stacks.chainId`.
4. `methodDetails.stacks.decimals = 6`.
5. `request.currency` equals `<contractAddress>.<contractName>::<assetName>` using the parsed values in `methodDetails.stacks`.
6. `(contractAddress, contractName, assetName)` matches a USDCx SIP-010 token in the Circle xReserve registry or an explicit implementation allowlist for the selected chain.
7. `payload.transaction` decodes as a SIP-005 consensus-serialized transaction.
8. The transaction version byte matches `methodDetails.stacks.network` and the transaction chain id matches `methodDetails.stacks.chainId`.
9. `anchor_mode = OnChainOnly`.
10. `auth` matches `feePayer`: `Sponsored` with origin signed and sponsor slot empty when `feePayer=true`; otherwise `Standard` with a single origin signature.
11. The origin auth signature verifies, uses a low-s secp256k1 signature, and recovers to a public key whose principal equals `source`.
12. When `feePayer=true`, the sponsor principal equals `methodDetails.stacks.feePayerAddress`; the server MUST co-sign only after fee estimation satisfies local fee policy.
13. The payload is a `ContractCall` to `contractAddress.contractName` with `function_name = "transfer"` and exactly the Clarity arguments `(uint amount, principal sender, principal recipient, (optional (buff 34)) memo)`.
14. The sender argument equals the principal in `source`.
15. The recipient argument equals `request.recipient`.
16. The amount argument equals `request.amount`.

17. `post_condition_mode = Deny`.
18. The transaction has exactly one `FungiblePostCondition` with `principal = source-principal`, `asset_info = (contractAddress, contractName, assetName)`, `condition_code = SentEq`, and `amount = request.amount`.
19. The origin auth nonce is fresh under the server's local Stacks nonce policy.
20. The transaction reaches the server's local Stacks confirmation threshold.

For Gateway Transfer, servers MUST verify:

1. `payload.type = "transfer"`.
2. `methodDetails.type = "gateway"`.
3. `request.currency = "usdc"`.
4. `credential.source` is a CAIP-10 account for `payload.sourceNetwork` and identifies the source depositor.
5. `payload.sourceNetwork` is included in `methodDetails.gateway.acceptedSources`.
6. `payload.destinationNetwork = methodDetails.gateway.destinationNetwork`.
7. `payload.maxFee <= methodDetails.gateway.maxFee`, with both values parsed as unsigned decimal base-unit integers and compared numerically.
8. Circle Gateway supports the selected source network and the destination network for native USDC settlement.
9. The Gateway authorization package exposes one signed Gateway transfer item to the verifier, either directly or through a conforming SDK.
10. The server inspects the Gateway transfer item before submission and submits the same signed package to Circle Gateway.
11. The Gateway TransferSpec inside the transfer item matches the request recipient model, source depositor, source signer, `payload.sourceNetwork`, `payload.destinationNetwork`, and a Circle-supported route. For EVM destinations, the TransferSpec destination recipient MUST equal `request.recipient`. For Solana destinations, the TransferSpec destination recipient MUST be the USDC token account for the owner in `request.recipient`. When Gateway recipient setup options are present, they MUST encode `request.recipient` as the recipient owner.
12. The Gateway TransferSpec salt matches the challenge-bound salt defined in Section 6.6.1.
13. The Gateway TransferSpec value equals `request.amount`.
14. The signed Gateway authorization fee cap equals `payload.maxFee`, does not exceed `methodDetails.gateway.maxFee`, and, when `methodDetails.gateway.maxFeeBps` is present, satisfies the `maxFeeBps` ratio cap defined in Section 6.6.
15. PaymentAuth replay protection has not already consumed the selected challenge or the challenge-bound Gateway salt for this source depositor.
16. Circle Gateway accepts the signed authorization. Gateway validation covers the Gateway signature, source signer authorization, route support, source balance, Gateway replay, TransferSpec encoding, TransferSpec hash, and transfer validity.
17. Circle Gateway reports destination settlement before the server returns success.

9. Settlement Procedure

For `intent="charge"`, settlement is complete only after the underlying chain transaction has reached the server's local confirmation threshold.

For Gateway Transfer, settlement is complete only after Circle Gateway reports destination settlement and exposes the destination transaction hash, signature, transaction id, or equivalent final settlement reference. After PaymentAuth verification, the server submits the Gateway authorization through Circle Gateway or a conforming SDK. A server MAY use Gateway forwarding [CIRCLE-GATEWAY-FORWARDING] to complete the destination settlement, but it MUST NOT return a successful charge receipt while the Gateway transfer is only estimated, submitted, attested, pending, or confirmed but not finalized.

9.1. Gateway Transfer Non-Success Outcomes

If Circle Gateway reports `failed` or `expired` before destination settlement, the server MUST return a new `402` challenge. The client MUST treat the original Gateway authorization as no longer reusable and sign a new credential if it wants to retry the payment.

If a Gateway transfer remains `pending` past the server's payment deadline, the server SHOULD return `402` with `Retry-After` [RFC9110] and a status reference. The client SHOULD NOT sign a replacement authorization for the same resource until the original transfer reaches a terminal status or the server reports that it is safe to retry.

If the server loses transport state while submitting a Gateway authorization, it MUST reconcile with Circle Gateway using any available transfer or TransferSpec reference before requesting a replacement authorization. This profile does not define repeated submission of the same signed Gateway authorization as an idempotent client retry.

The server MUST NOT return a successful receipt before the selected profile's settlement has completed. If the server wants admission before onchain settlement, it MUST use a separate deferred-settlement method or intent outside this document.

10. Receipt Schema

Upon successful settlement, servers MUST return a `Payment-Receipt` header per [I-D.httpauth-payment]. The decoded receipt payload contains the following fields. Fields are REQUIRED unless the description says OPTIONAL.

The receipt is a settlement pointer interpreted together with the original challenge and request. It does not repeat every request or verification field.

Field	Type	Description
`method`	string	MUST be `usdc`.

Field	Type	Description
`type`	string	Selected USDC profile: `evm`, `solana`, `stacks`, or `gateway`.
`challengeId`	string	Original challenge ID.
`reference`	string	Final settlement reference.
`status`	string	MUST be `success` only after the selected profile has completed settlement.
`timestamp`	string	RFC3339 settlement time.
`network`	string	CAIP-2 [CAIP-2] settlement network identifier.
`externalId`	string	OPTIONAL. Echo of `request.externalId`.

Table 13

For direct EVM, `reference` is the EVM transaction hash. For direct Solana, `reference` is the Solana transaction signature. For direct Stacks, `reference` is the Stacks transaction ID.

For Gateway Transfer, `reference` is the final destination settlement reference exposed by Circle Gateway, and `network` MUST be `methodDetails.gateway.destinationNetwork`. The receipt MAY include a `gateway` object with Gateway audit handles. SDKs that do not need Gateway reconciliation MAY ignore this object.

When present, the `gateway` object contains:

Field	Type	Required	Description
`transferId`	string	OPTIONAL	Circle Gateway transfer UUID when available.
`sourceNetwork`	string	OPTIONAL	CAIP-2 source network used by the Gateway transfer.
`destinationNetwork`	string	OPTIONAL	CAIP-2 destination network. When present, MUST equal receipt `network`.
`transferSpecHash`	string	OPTIONAL	Gateway-returned hash of the TransferSpec used by the transfer.

Table 14

The receipt `gateway.transferSpecHash` is evidence returned by Circle Gateway. It is not supplied by the client credential. A verifier MAY use Circle Gateway TransferSpec lookup [CIRCLE-GATEWAY-TRANSFER-SPEC] to inspect the settled TransferSpec after acceptance or settlement.

The receipt uses a single CAIP-2 [CAIP-2] `network` field for every profile. This is a deliberate, method-wide settlement locator and intentionally differs from the base EVM charge receipt, which uses a numeric `chainId`, and the base Solana charge receipt, which has no network field. A `method="usdc"` receipt consumer reads `network` for every profile instead of branching on the profile type.

11. Security Considerations

11.1. Supported USDC Asset Forms

This profile defines which USDC asset forms can satisfy `method="usdc"` in v00. Servers MUST verify that the selected token matches a native USDC deployment published by Circle, or, for Stacks, a USDCx token listed in the Circle xReserve registry or an explicit implementation allowlist. Other bridged, wrapped, or synthetic USDC-like assets need a separate profile or explicit method details.

Gateway Wallet deposits remain subject to Circle Gateway recovery and withdrawal rules. Gateway supports delayed trustless withdrawal from Gateway Wallet when the service is unavailable. That recovery path is outside PaymentAuth settlement and does not make a pending Gateway Transfer idempotently retryable.

11.2. Blocklist and Pause Controls

Native USDC and USDCx on Stacks have different control surfaces. Native USDC has token-level pause and blocklist controls on the selected chain. For USDCx on Stacks, xReserve [CIRCLE-XRESERVE] source-chain controls gate deposits and withdrawals, but those controls do not freeze same-chain partner token transfers by themselves. Same-chain USDCx transfers depend on the partner-chain token controls.

Servers MUST apply every control surface that is relevant to the selected profile before accepting a credential. For native USDC, a payer or recipient present in the applicable token blocklist MUST cause rejection. For USDCx on Stacks, servers MUST check the Circle xReserve control surface and the partner-chain token control surface required by local policy. Cached control data MUST have an explicit freshness bound.

11.3. Replay Protection

Servers MUST bind each credential to the selected challenge and MUST maintain replay protection for consumed credentials. Challenge consumption and receipt issuance MUST be atomic: concurrent requests presenting the same valid credential MUST produce at most one successful receipt.

For EVM authorization credentials, the persistent replay key MUST include `(chainId, verifyingContract, payload.from, payload.nonce)`. The nonce already commits to `method`, `intent`, `challenge.id`, `challenge.realm`, and `requestHash`, so this key protects both onchain replay and cross-context replay.

For Solana transaction credentials, replay protection follows [I-D.solana-charge]. Servers MUST maintain consumed transaction signatures and atomically consume the selected challenge before returning a successful receipt. Servers that admit pull-mode transactions before broadcast SHOULD also deduplicate on `(network, transactionBytesDigest)` to avoid concurrent admission of the same serialized transaction.

For Stacks transaction credentials, the replay key MUST include the transaction id after broadcast and `(stacks:CHAIN_ID, ORIGIN_PRINCIPAL, ORIGIN_NONCE)`. Servers that admit transactions before broadcast SHOULD also deduplicate on `(stacks:CHAIN_ID, transactionBytesDigest)` to avoid concurrent admission of the same serialized transaction.

For Gateway Transfer credentials, replay protection MUST cover the selected challenge and the challenge-bound Gateway salt before the server submits the authorization to Circle Gateway. A server that only keys replay protection by source transaction or destination transaction can submit the same signed Gateway authorization more than once before final settlement. Circle Gateway separately enforces Gateway-side replay when it accepts the transfer. The TransferSpec salt binds the Gateway authorization to the PaymentAuth challenge, request, source, destination, recipient, amount, and fee cap, as specified in Section 6.6.1.

If a deployment advertises the same merchant order through both `method="usdc"` and a chain-specific method such as `evm` or `solana`, it MUST enforce one logical purchase across those offers. The deduplication key MUST be a stable merchant order key, normally `request.externalId` when present. If `externalId` is absent, the deployment MUST maintain an explicit equivalent-offer group outside the credential. This is merchant order deduplication, not a replacement for challenge replay protection. A successful receipt for one offer MUST atomically consume the shared key so that a later credential for the same order is rejected.

11.4. Gateway Transfer Fees

Gateway Transfer can charge fees in addition to the merchant amount. The request binds a maximum fee in `methodDetails.gateway.maxFee`. Servers MUST reject Gateway authorization material that exceeds that fee cap. If `methodDetails.gateway.maxFeeBps` is present, servers MUST also reject authorization material whose fee cap exceeds that ratio. Clients SHOULD present the payer's worst-case spend as `amount + maxFee`.

Gateway Transfer fees are payer-paid in v00. The merchant server does not sponsor those Gateway fees, even if it waits for destination settlement before returning a successful `charge` receipt.

11.5. Sensitive Fields

USDC credentials are bearer-equivalent between signing and settlement. Servers SHOULD avoid logging raw authorization signatures, serialized transactions, Gateway authorization packages, or full credential headers.

12. IANA Considerations

This specification registers the `usdc` payment method in the HTTP Payment Methods Registry.

Field	Value
Method Identifier	`usdc`
Description	USDC payments across supported networks
Reference	This document
Contact	Harshal Bhangale, Circle Internet Group, Inc.

Table 15

This document does not register `charge`; that intent is registered by [I-D.payment-intent-charge]. This document states that `method="usdc"` supports `intent="charge"` in v00.

13. Appendix A. Examples

This appendix is informative. Field shapes are normative, example values are not. Long signatures and chain transaction bytes are shortened when the underlying chain format is already defined by the referenced profile or SDK.

13.1. A.1 EVM Direct Charge

The decoded request is:

```
json { "amount": "1000000", "currency":
"0x3600000000000000000000000000000000000000", "recipient":
"0xc04193C50cD2E6a1C79593e46364496Fe5fcd9b6", "description": "Arc Testnet USDC
charge", "externalId": "invoice-evm-001", "methodDetails": { "type": "evm",
"evm": { "chainId": 5042002, "decimals": 6, "credentialTypes":
[ "authorization" ] } } }
```

The server advertises the challenge:

```
http HTTP/1.1 402 Payment Required WWW-Authenticate: Payment
realm="api.example.com", method="usdc", intent="charge",
id="usdc_evm_direct_001",
request="eyJhbW91bnQiOiIxMDAwMDAwIiwiY3VycmVuY3kiOiIweDM2MDAwMDAwMDAwMDAwMDAwMDAw
MDAwMDAwMDAwMDAwMDAwMDAwMDAiLCJkZXNjcmlwdGlvbiI6IkFyYyBUZXN0bmV0IFVTREMgY2hhcmdlI
iwiZXh0ZXJuYWxJZCI6Imludm9pY2UtZXZtLTAwMSIsIm1ldGhvZERldGFpbHMiOnsiZXZtIjp7ImNoYW
```

luSWQiOjUwNDIwMDIsImNyZWRlbnRpYWxUeXBlcyI6WyJhdXRob3JpemF0aW9uIl0sImRlY2ltYWxzIjo2fSwidHlwZSI6ImV2bSJ9LCJyZWNpcGllbnQiOiIweGMwNDE5M0M1MGNEMkU2YTFDNzk1OTNlNDYzNjQ0OTZGZTVmY2Q5YjYifQ", expires="2026-04-01T12:05:00Z"

The client returns an EIP-3009 authorization credential:

json { "challenge": { "id": "usdc_evm_direct_001", "realm": "api.example.com", "method": "usdc", "intent": "charge", "request": "eyJhbW91bnQiOiIxMDAwMDAwIiwiY3VycmVuY3kiOiIweDM2MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAiLCJkZXNjcmlwdGlvbiI6IkFyYyBUZXN0bmV0IFVTREMgY2hhcmdlIiwiZXh0ZXJuYWxJZCI6Imludm9pY2UtZXZtLTAwMSIsIm1ldGhvZERldGFpbHMiOnsiZXZtIjp7ImNoYWluSWQiOjUwNDIwMDIsImNyZWRlbnRpYWxUeXBlcyI6WyJhdXRob3JpemF0aW9uIl0sImRlY2ltYWxzIjo2fSwidHlwZSI6ImV2bSJ9LCJyZWNpcGllbnQiOiIweGMwNDE5M0M1MGNEMkU2YTFDNzk1OTNlNDYzNjQ0OTZGZTVmY2Q5YjYifQ", "expires": "2026-04-01T12:05:00Z" }, "source": "did:pkh:eip155:5042002:0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266", "payload": { "type": "authorization", "from": "0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266", "to": "0xc04193c50cd2e6a1c79593e46364496fe5fcd9b6", "value": "1000000", "validAfter": "0", "validBefore": "1775045100", "nonce": "0x03e1d1aa38e2c56a0bb12e2d4562082c1c26496553f838064f3e6b4c3db9d2c2", "signature": "0xeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee" } }

The server submits the authorization, pays EVM gas, waits for the EVM receipt, and returns:

```
http HTTP/1.1 200 OK Payment-Receipt: BASE64URL_JCS_RECEIPT
```

The decoded receipt payload is:

json { "method": "usdc", "type": "evm", "challengeId": "usdc_evm_direct_001", "reference": "0x3c5b4a1f00000000000000000000000000000000000000000000000000008d0a2c4e6b", "status": "success", "timestamp": "2026-04-01T12:00:04Z", "network": "eip155:5042002", "externalId": "invoice-evm-001" }

13.1.1. A.2 Solana Direct Charge

The decoded request is:

json { "amount": "1000000", "currency": "4zMMC9srt5Ri5X14GAgXhaHii3GnPAEERYPJgZJDncDU", "recipient": "AKnL4NNf3DGWZJS6cPknBuEGnVsV4A4m5tgebLHaRSZ9", "description": "Solana devnet USDC charge", "externalId": "invoice-sol-001", "methodDetails": { "type": "solana", "solana": { "network": "devnet", "decimals": 6, "tokenProgram": "TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA" } } }

The client returns the Solana transaction credential defined by [I-D.solana-charge]:

```
json { "challenge": { "id": "usdc_solana_direct_001", "realm": "api.example.com",
"method": "usdc", "intent": "charge", "request":
"eyJhbW91bnQiOiIxMDAwMDAwIiwiY3VycmVuY3kiOiI0ek1NQzlzcnQ1Umk1WDE0R0FnWGhhSGlpM0du
UEFFRVJZUEpnWkpEbmNEVSIsImRlc2NyaXB0aW9uIjoiU29sYW5hIGRldm5ldCBVU0RDIGNoYXJnZSIsI
mV4dGVybmFsSWQiOiJpbnZvaWNlLXNvbC0wMDEiLCJtZXRob2REZXRhaWxzIjp7InNvbGFuYSI6eyJkZW
NpbWFscyI6NiwibmV0d29yayI6ImRldm5ldCIsInRva2VuUHJvZ3JhbSI6IlRva2Vua2VnUWZlWnlpTnd
BSmJOYkdLUEZYQ1d1QnZmOVNzNjIzVlE1REEifSwidHlwZSI6InNvbGFuYSJ9LCJyZWNpcGllbnQiOiJB
S25MNE5OZjNER1daSlM2Y1BrbkJ1RUduVnNWNEE0bTV0Z2ViTEhhUlNaOSJ9", "expires":
"2026-04-01T12:05:00Z" }, "source":
"did:pkh:solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1:DRpbCBMxVnDK7maPM5tGv6MvB3v1sRMC
86PZ8okm21hy", "payload": { "type": "transaction", "transaction":
"BASE64_SOLANA_TRANSACTION" } }
```

This example omits `methodDetails.solana.feePayer`, so the client pays the Solana transaction fee. If `feePayer=true`, the transaction leaves the server fee-payer signature as the only missing required signature.

After the transaction reaches the required commitment level, the server returns:

```
http HTTP/1.1 200 OK Payment-Receipt: BASE64URL_JCS_RECEIPT
```

The decoded receipt payload is:

```
json { "method": "usdc", "type": "solana", "challengeId":
"usdc_solana_direct_001", "reference":
"5j7s2KpP4uYc8LmZqEhNwR3vJbXt6yA1DsVfBgCoM9TpHxUeQk", "status": "success",
"timestamp": "2026-04-01T12:00:05Z", "network":
"solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1", "externalId": "invoice-sol-001" }
```

13.1.2. A.3 Stacks Direct Charge

The decoded request is:

```
json { "amount": "1000000", "currency":
"ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM.usdcx::usdcx-token", "recipient":
"ST3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8NQXMNRQ", "description": "Stacks testnet USDCx
charge", "externalId": "invoice-stx-001", "methodDetails": { "type": "stacks",
"stacks": { "network": "testnet", "chainId": "2147483648", "contractAddress":
"ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM", "contractName": "usdcx",
"assetName": "usdcx-token", "functionName": "transfer", "decimals": 6,
"feePayer": true, "feePayerAddress":
"ST4488BK2MKPFQBWPC7YYZKCRMQN52ST0ZV6EWT5" } } }
```

The client returns a Stacks transaction credential:

```
json { "challenge": { "id": "usdc_stacks_direct_001", "realm": "api.example.com",
"method": "usdc", "intent": "charge", "request":
"eyJhbW91bnQiOiIxMDAwMDAwIiwiY3VycmVuY3kiOiJTVDFQUUhRS1YwUkpYWkZZMURHWDhNTlNOWVZF
```

M1ZHWkpTUlRQR1pHTS51c2RjeDo6dXNkY3gtdG9rZW4iLCJkZXNjcmlwdGlvbiI6IlN0YWNrcyB0ZXN0bmV0IFVTREN4IGNoYXJnZSIsImV4dGVybmFsSWQiOiJpbnZvaWNlLXN0eC0wMDEiLCJtZXRob2REZXRhaWxzIjp7InN0YWNrcyI6eyJhc3NldE5hbWUiOiJ1c2RjeC10b2tlbiIsImNoYWluSWQiOiIyMTQ3NDgzNjQ4IiwiY29udHJhY3RBZGRyZXNzIjoiU1QxUFFIUUtWMFJKWFpGWTFER1g4TU5TTllWRTNWR1pKU1JUUEdaR00iLCJjb250cmFjdE5hbWUiOiJ1c2RjeCIsImRlY2ltYWxzIjo2LCJmZWVQYXllciI6dHJ1ZSwiZmVlUGF5ZXJBZGRyZXNzIjoiU1Q0NDg4QksyTUtQRlFCV1BDN1lZWktDUk1RTjUyU1QwWlY2RVdUNSIsImZ1bmN0aW9uTmFtZSI6InRyYW5zZmVyIiwibmV0d29yayI6InRlc3RuZXQifSwidHlwZSI6InN0YWNrcyJ9LCJyZWNpcGllbnQiOiJTVDNGQlIyQUdLNUg5UUJESDNFRU42REY4RUs4Slk3Ulg4TlFYTU5SUSJ9",
"expires": "2026-04-01T12:05:00Z" }, "source":
"stacks:2147483648:ST8H248H248H248H248H248H248H248H26RCPJ4T", "payload":
{ "type": "transaction", "transaction": "BASE64_SIP005_SERIALIZED_TRANSACTION",
"transactionFormat": "stacks_transaction_v1" } }

The server sponsors, broadcasts, waits for the Stacks transaction, and returns:

```
http HTTP/1.1 200 OK Payment-Receipt: BASE64URL_JCS_RECEIPT
```

The decoded receipt payload is:

json { "method": "usdc", "type": "stacks", "challengeId":
"usdc_stacks_direct_001", "reference":
"0x9a1b2c3d4e5f60718293a4b5c6d7e8f9a0b1c2d3e4f50617283a4b5c6d7e8f90", "status":
"success", "timestamp": "2026-04-01T12:01:45Z", "network": "stacks:2147483648",
"externalId": "invoice-stx-001" }

13.1.3. A.4 Gateway Transfer Charge

This example shows an Arc Testnet source paying a Solana Devnet merchant through Circle Gateway. The `recipient` is the merchant's Solana owner address; the Gateway transfer uses the owner's USDC token account as `spec.destinationRecipient`. The same profile also supports Solana to EVM when Circle Gateway supports that route. The example lists two accepted sources for readability.

The decoded request is:

json { "amount": "25000000", "currency": "usdc", "recipient":
"AKnL4NNf3DGWZJS6cPknBuEGnVsV4A4m5tgebLHaRSZ9", "description": "Arc Gateway
balance to Solana merchant", "externalId": "invoice-gw-001", "methodDetails":
{ "type": "gateway", "gateway": { "acceptedSources": ["eip155:5042002",
"solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1"], "destinationNetwork":
"solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1", "maxFee": "500000", "credentialTypes":
["transfer"] } } }

The server advertises the challenge:

```
http HTTP/1.1 402 Payment Required WWW-Authenticate: Payment
realm="api.example.com", method="usdc", intent="charge",
id="usdc_gateway_transfer_001",
```

```
request="eyJhbW91bnQiOiIyNTAwMDAwMCIsImN1cnJlbmN5IjoidXNkYyIsImRlc2NyaXB0aW9uIjoi
QXJjIEdhdGV3YXkgYmFsYW5jZSB0byBTb2xhbmEgbWVyY2hhbnQiLCJleHRlcm5hbElkIjoiaW52b2ljZ
S1ndy0wMDEiLCJtZXRob2REZXRhaWxzIjp7ImdhdGV3YXkiOnsiYWNjZXB0ZWRTb3VyY2VzIjpbImVpcD
E1NTo1MDQyMDAyIiwic29sYW5hOkV0V1RSQUJaYVlxNmlNZmVZS291UnUxNjZWVTJ4cWExIl0sImNyZWR
lbnRpYWxUeXBlcyI6WyJ0cmFuc2ZlciJdLCJkZXN0aW5hdGlvbk5ldHdvcmsiOiJzb2xhbmE6RXRXVFJB
QlphWXE2aU1mZVlLb3VSdTE2NlZVMnhxYTEiLCJtYXhGZWUiOiI1MDAwMDAifSwidHlwZSI6ImdhdGV3Y
XkifSwicmVjaXBpZW50IjoiQUtuTDROTmYzREdXWkpTNmNQa25CdUVHblZzVjRBNG01dGdlYkxIYVJTWj
kifQ", expires="2026-04-01T12:05:00Z"
```

The client returns a Gateway Transfer credential. The nested Gateway authorization object below is SDK-produced and illustrative; signatures, salts, hashes, and transaction bytes are not self-verifying test vectors.

```
json { "challenge": { "id": "usdc_gateway_transfer_001", "realm":
"api.example.com", "method": "usdc", "intent": "charge", "request":
"eyJhbW91bnQiOiIyNTAwMDAwMCIsImN1cnJlbmN5IjoidXNkYyIsImRlc2NyaXB0aW9uIjoiQXJjIEdh
dGV3YXkgYmFsYW5jZSB0byBTb2xhbmEgbWVyY2hhbnQiLCJleHRlcm5hbElkIjoiaW52b2ljZS1ndy0wM
DEiLCJtZXRob2REZXRhaWxzIjp7ImdhdGV3YXkiOnsiYWNjZXB0ZWRTb3VyY2VzIjpbImVpcDE1NTo1MD
QyMDAyIiwic29sYW5hOkV0V1RSQUJaYVlxNmlNZmVZS291UnUxNjZWVTJ4cWExIl0sImNyZWRlbnRpYWx
UeXBlcyI6WyJ0cmFuc2ZlciJdLCJkZXN0aW5hdGlvbk5ldHdvcmsiOiJzb2xhbmE6RXRXVFJBQlphWXE2
aU1mZVlLb3VSdTE2NlZVMnhxYTEiLCJtYXhGZWUiOiI1MDAwMDAifSwidHlwZSI6ImdhdGV3YXkifSwic
mVjaXBpZW50IjoiQUtuTDROTmYzREdXWkpTNmNQa25CdUVHblZzVjRBNG01dGdlYkxIYVJTWjkifQ",
"expires": "2026-04-01T12:05:00Z" }, "source":
"eip155:5042002:0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266", "payload": { "type":
"transfer", "sourceNetwork": "eip155:5042002", "destinationNetwork":
"solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1", "maxFee": "500000", "authorization":
{ "format": "circle-gateway-v1", "transfer": { "burnIntent": { "maxBlockHeight":
"39350204", "maxFee": "500000", "spec": { "version": 1, "sourceDomain": 26,
"destinationDomain": 5, "sourceContract":
"0x00000000000000000000000077777d7eba4688bdef3e311b846f25870a19b9",
"destinationContract":
"0x0000000000000000000000000000000000000000000000000000000000000005",
"sourceToken":
"0x0000000000000000000000360000000000000000000000000000000000000000",
"destinationToken":
"0x0000000000000000000000000000000000000000000000000000000000000006",
"sourceDepositor":
"0x000000000000000000000000f39fd6e51aad88f6f4ce6ab8827279cfffb92266",
"destinationRecipient":
"0xedc636e0401e29c099ea2703806591e6e40beec1091507025dbbfe6d63761fe2",
"sourceSigner":
"0x000000000000000000000000f39fd6e51aad88f6f4ce6ab8827279cfffb92266",
"destinationCaller":
"0x0000000000000000000000000000000000000000000000000000000000000000", "value":
"25000000", "salt":
"0xab8500000000000000000000000000000000000000000000000000000000000000" },
```

```
"recipientSetupOptions": { "includeRecipientSetup": true,
"recipientOwnerAddress":
"0x8a88e3dd7409f195fd52db2d3cba5d72ca6709bf1d94121bf3748801b40f6f5c" } },
"signature":
"0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffff" } } } }
```


The payer authorizes up to 0.50 USDC in Gateway fees on top of the 25.00 USDC merchant amount. The server submits the Gateway authorization to Circle Gateway, waits for destination settlement, and returns a receipt that includes the Gateway-returned `transferSpecHash`:

```
http HTTP/1.1 200 OK Payment-Receipt: BASE64URL_JCS_RECEIPT
```

The decoded receipt payload is:


```
json { "method": "usdc", "type": "gateway", "challengeId":
"usdc_gateway_transfer_001", "reference":
"5j7s2KpP4uYc8LmZqEhNwR3vJbXt6yA1DsVfBgCoM9TpHxUeQk", "status": "success",
"timestamp": "2026-04-01T12:02:17Z", "network":
"solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1", "externalId": "invoice-gw-001",
"gateway": { "transferId": "550e8400-e29b-41d4-a716-446655440000",
"sourceNetwork": "eip155:5042002", "destinationNetwork":
"solana:EtWTRABZaYq6iMfeYKouRu166VU2xqa1", "transferSpecHash":
"0xca85000000000000000000000000000000000000000000000000000000000000" } }
```


14. References

14.1. 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>.

[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>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

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

[I-D.payment-intent-charge] Moxey, J., Ryan, B., and T. Meagher, "'charge' Intent for HTTP Payment Authentication", 2026, <https://datatracker.ietf.org/doc/draft-payment-intent-charge/>.

[I-D.evm-charge] "EVM Charge Intent for HTTP Payment Authentication", n.d., <https://paymentauth.org/draft-evm-charge-00.html>.

[I-D.solana-charge] "Solana Charge Intent for HTTP Payment Authentication", n.d., <https://paymentauth.org/draft-solana-charge-00.html>.

[CAIP-2] "Chain Agnostic Improvement Proposal 2", n.d., <https://chainagnostic.org/CAIPs/caip-2>.

[CAIP-10] "Chain Agnostic Improvement Proposal 10", n.d., <https://chainagnostic.org/CAIPs/caip-10>.

[SIP-005] "Stacks Blocks, Transactions, and Accounts", n.d., <https://raw.githubusercontent.com/stacksgov/sips/main/sips/sip-005/sip-005-blocks-and-transactions.md>.

[SIP-010] "Stacks Fungible Token Standard", n.d., <https://raw.githubusercontent.com/stacksgov/sips/main/sips/sip-010/sip-010-fungible-token-standard.md>.

[CIRCLE-GATEWAY-INFO] "Circle Gateway GET /v1/info", n.d., <https://developers.circle.com/api-reference/gateway/all/get-gateway-info>.

[CIRCLE-GATEWAY-ESTIMATE] "Circle Gateway POST /v1/estimate", n.d., <https://developers.circle.com/api-reference/gateway/all/estimate-transfer>.

[CIRCLE-GATEWAY-TRANSFER] "Circle Gateway POST /v1/transfer", n.d., <https://developers.circle.com/api-reference/gateway/all/create-transfer-attestation>.

[CIRCLE-GATEWAY-TRANSFER-STATUS] "Circle Gateway GET /v1/transfer/{id}", n.d., <https://developers.circle.com/api-reference/gateway/all/get-transfer-by-id>.

[CIRCLE-GATEWAY-TRANSFER-SPEC] "Circle Gateway GET /v1/transferSpec/{transferSpecHash}", n.d., <https://developers.circle.com/api-reference/gateway/all/get-transfer-spec>.

14.2. Informative References

[CIRCLE-GATEWAY-FORWARDING] "Circle Gateway Forwarding Service guide", n.d., <https://developers.circle.com/gateway/howtos/forwarding-service>.

[CIRCLE-USDC-ADDRESSES] "Circle USDC Contract Addresses", n.d., <https://developers.circle.com/stablecoins/usdc-contract-addresses>.

[CIRCLE-XRESERVE] "Circle xReserve architecture", n.d., <https://developers.circle.com/xreserve>.

Authors' Addresses

Harshal Bhangale
Circle Internet Group, Inc.
Email: harshal.bhangale@circle.com

Huawei Gu
Circle Internet Group, Inc.
Email: hgu@circle.com

Bhushit Agarwal
Circle Internet Group, Inc.
Email: bhushit.agarwal@circle.com