Implementing a full-featured Verite Identity Wallet

Verite Wallet Integration Guide Suggest Edits

This guide is written for developers seeking to integrate the "wallet-bound" Verite flows and data models into custodial or non-custodial wallets. What follows is best understood as a checklist for adding "identity wallet" capabilities (handling Verifiable Credentials natively and as flexible as built-for-purpose identity wallets do) to an existing mobile wallet, whether non-custodial, custodial, or semi-custodial (MPC, multi-sig, etc).

Minimal Wallet Requirements - Summary

- Generate (or supply) and manage decentralized identifiers for credential subject* did:key (offchain, local-only) is recommended for staging/test-net development
- Generate VC/VP-JWT signatures
- Support credential storage and retrieval
- Support credential issuance and exchange flows described in this document

Architectural Foundations

Verite uses a decentralized identity architecture, in which the individual (subject/holder) directly requests and receives credentials from an issuer, storing them in a wallet that governs a unique and long-lasting DID. In this model, individuals decide when and with whom they want to share their credentials, referred to as verifiers, or more generally, relying parties. In some architectures and use-cases, these counterparties may know the wallet and its useronly by a DID, or by a DID and the contents of credentials managed by the wallet.

The relying party verifies the credential, without necessarily needing to contact the issuer, through an extensible mechanism enabled by the VC specification. The verification process is designed to be maximally privacy-respecting and cryptographically secure.

Concepts Overview

Verite's basis is theW3C Verifiable Credentials (VCs) Data Model . A credential is a claim made by an issuer about an individual (subject), and a verifiable credential is a cryptographically secure wrapper around that credential enabling verification and handling by common, standards-based libraries such as JWT tooling.

Following is a list of concepts used in Verite.

- W3C Verifiable Credentials
- : flexible, tamper-evident way for an issuer to make a claim about a subject in a way that is independently verifiable and privacy-preserving. VCs are the data model used for Verite claims
- Verifiable Presentations
- : also defined in the VC spec, are a way to securely package a set of VCs for transmission to a relying party, in a way that also allows the subject to prove control over the credentials.
- Identifiers, Decentralized Identifiers
- : an "identifier" refers to both a subject and an issuer of a VC. The identifier data type is a URI, one of which is a W3C Decentralized Identifiers (DIDs)
- , which is used in Verite implementations. Verite does not require the use of DIDs, but these are one way to implement identifiers that minimize correlatability and enable proof of control over credentials.
- Credential Manifest
- : standard from the Decentralized Identity Foundation (DIF) for requesting and receiving a credentialCredential Manifest
- allows an issuer to describe (in a machine-readable way) what types (schemas) of credentials they issue, and what their requirements are. It also describes the format for a subject/holder to submit an application for a credential that conforms to those requirements in a machine-readable way.
- Presentation Exchange
- : A DIF standard enabling a verifier to describe what types of credentials they require from a subject/holder, and how the subject/holder can send a submission. SeeDIF Presentation Exchange
- Wallet and Credential Interactions
- : wallet interaction protocols use a lightweight flow loosely based on the work-in-progressDIF Wallet and Credential Interactions
- spec.

Specifications and Libraries Used

See thelist of specifications and spec-conforming libraries used by Verite.

Credential Flows

The credential flows demonstrate use of a non-custodial credential and identifier wallet, but are easily adapted to hosted wallets, combination crypto+identity wallets, and various variations on the described flow. See the wallet-bound variants of theissuance andverification flow sections elsewhere in the docs for context.

Issuance Flow

Overview

This flow enables a subject to send prerequisite information to the issuer before credential issuance. Prerequisites include:

- subject-controlled identifier (e.g. decentralized identifier) the credential should be issued to, along with a proof of control of the identifier
- optionally, an issuer may request input, in verifiable credential form or otherwise (not currently represented in the verite sample implementation or documentation)

The process is initiated by the wallet holder scanning a QR code, which allows the wallet to determine the input requirements, asking for consent before sending (typically including cryptographically signed proof). The issuer validates the request, issues a credential to the identifier specified by the subject, and returns the credential

This flow is based on theDIF wallet and credential interaction (draft) specification , which enables use of links or QR codes for wallet-initiated interactions.

Details

1. Recipient initiates the credential request process by scanning QR code with their mobile wallet
2. The wallet decodes the QR code, which provides an issuer endpoint with more details to continue the interaction:
3.
    1. QR code returns the URL from which to receive information at the
4.
    1. challengeTokenUrl
5.
    1. :
6.
    1. JSON
7.
    1. {
8.
    1. "challengeTokenUrl": "https://verite.id/..."
9.
    1. }
10.
    1. Wallet GETs the payload:
11.
    1. GET {{challengeTokenUrl}}
12.
    1. The return type is a JWM containing acredential manifest
13.
    1. , credential request endpoint, and a challenge to sign.
14.
    1. JSON
15.
    1. {
16.
    1. "id": "4487e7d1-7d10-4075-a923-bae9332266c1",
17.
    1. "type": "CredentialOffer",
18.
    1. "from": "did:key:z6Mkgw8mPijYRa3TkHSYtQ4P7S2HGrcJBwzdgjeurqr9Luqb",
19.
    1. "created_time": "2021-09-14T01:22:05.816Z",
20.
    1. "expires_time": "2021-10-14T01:22:05.816Z",
21.
    1. "reply_url": "https://...",
22.
    1. "body": {

23.
    1. "challenge": "d273da29-74dd-46de-a53c-1677c51cc700",
24.
    1. "manifest": {}
25.
    1. }
26.
    1. }
27. Depending on the wallet implementation, the wallet generally prompts the user for approval to proceed, notifying which data is to be shared (from the credential manifest)
28. On approval, the wallet generates/provides a decentralized identifier, builds the credential request, and signs it to achieveproof of identifier control
29. .
30.
    1. In the Verite demo, the wallet generates a did:key DID, builds acredential application
31.
    1. , and signs the credential application, along with the challenge, with the private key corresponding to the DID.
32.
    1. The Verite library exposes a convenience method
33.
    1. createCredentialApplication
34.
    1. for this purpose -- this is used by demo-wallet in preparing a credential application.
35.
    1. Note: the Verite demo does not require the wallet to submit credentials for issuance, as the credential recipient is already known to the issuer, but the credential manifest and credential application structures allow for this if needed.
36.
    1. Example:Credential Application
37. The wallet sends the credential application to the issuer's endpoint
38. POST {{replyUrl}}
39. The issuer issues the credential and returns it to the wallet
40.
    1. The issuer accepts and validates the input and, on success, issues a Verifiable Credential.
41.
    1. In the Verite demo, the issuer replies with acredential fulfillment
42.
    1. -structured result containing a JWT-encoded Verifiable Presentation.
43.
    1. Examples:1. Credential Fulfillment
44.
    1.
        1. Decoded Credential
45. Wallet stores credential

Verification Flow

Overview

This flow enables a verifier or relying party to request credentials and proof requirements from the subject/holder.

The process is initiated by the wallet holder scanning a QR code, which allows the wallet to determine the credential/proof requirements, asking for consent before sending (typically including cryptographically signed proof).

This flow is based on theDIF wallet and credential interaction (draft) specification , which enables use of links or QR codes for wallet-initiated interactions.

Details

The exchange flow similarly initiates with the credential holder performing a QR scan. See message samples inPresentation Exchange

Appendix: Message Samples

This section contains examples of the messages referenced above.

Credential Issuance

QR Code

Credential subjects initiate credential issuance flows by scanning a QR code, provided by the issuer, with their credential wallet.

Sample:

{ "challengeTokenUrl": "https://verite.id/..." }

Credential Offer

An issuer credential offer tells the wallet how to initiate a credential issuance request. It is a simple JWM wrapper around a DIF Credential Manifest.

Sample:

JSON { "id": "4487e7d1-7d10-4075-a923-bae9332266c1", "type": "CredentialOffer", "from": "did:key:z6Mkgw8mPijYRa3TkHSYtQ4P7S2HGrcJBwzdgjeurqr9Luqb", "created_time": "2021-09-14T01:22:05.816Z", "expires_time": "2021-10-14T01:22:05.816Z", "reply_url": "https://...", "body": { "challenge": "d273da29-74dd-46de-a53c-1677c51cc700", "manifest": {} } } Details:

- from: who the message is from; in this case, the issuer
- reply_url: the URL the wallet should send the credential application to
- body.challenge: a challenge the wallet should sign when proving control, to prevent replays
- body.manifest: follows the DIF Credential Manifest spec

Credential Application

A credential application is sent from the wallet to the issuer before issuance. It contains the recipient identifier and other information required for issuance. This follows the DIF Credential Manifest Spec.

What follows is a JSON object containing the same contents as a Verifiable Presentation in JWT form; there is no proof object, because it would be signed and transmitted as a JWT.

JSON { "@context": ["https://www.w3.org/2018/credentials/v1"], "credential_application": { "id": "2ce196be-fcda-4054-9eeb-8e4c5ef771e5", "manifest_id": "KYCAMLManifest", "format": { "jwt_vp": { "alg": ["EdDSA"] } } }, "presentation_submission": { "id": "b4f43310-1d6b-425d-84c6-f8afac3fe244", "definition_id": "ProofOfControlPresentationDefinition", "descriptor_map": [{ "id": "proofOfIdentifierControlVP", "format": "jwt_vp", "path": ".holder" }] }, "verifiableCredential": [], "holder": "did:key:z6MkjFFeDnzyKL7Q39aNs1piGo27b12upMf1MmSDQcABJmmn", "type": ["VerifiablePresentation", "CredentialApplication"] }

Credential Fulfillment

What follows is a JSON object containing the same contents as a Verifiable Presentation in JWT form; there is no proof object, because it would be signed and transmitted as a JWT.

JSON { "@context": ["https://www.w3.org/2018/credentials/v1"], "type": ["VerifiablePresentation", "CredentialResponse"], "holder": "did:key:z6Mkgw8mPijYRa3TkHSYtQ4P7S2HGrcJBwzdgjeurqr9Luqb", "credential_fulfillment": { "id": "5f22f1ea-0441-4041-916b-2504a2a4075c", "manifest_id": "KYCAMLManifest", "descriptor_map": [{ "id": "KYCAMLCredential", "format": "jwt_vc", "path": ".verifiableCredential[0]" }] }, "verifiableCredential": [] // Credential would be found here, as a JWT, i.e. ["eyJhbG..."] }

Decoded Credential

The following represents the intermediate form of a JWT-encoded verifiable credentialpost-verification and post-decoding to restore the "credential" (i.e., combining fields from both the payload and the protected headers of the JWT token):

JSON { "@context": ["https://www.w3.org/2018/credentials/v1", { "@vocab": "https://verite.id/identity/" }], "type": ["VerifiableCredential", "KYCAMLCredential"], "credentialSubject": { "KYCAMLAttestation": { "type": "KYCAMLAttestation", "process": "https://verite.id/definitions/processes/kycaml/0.0.1/usa", "approvalDate": "2021-09-14T02:00:07.540Z" }, "id": "did:key:z6Mkjo9pGYpv88SCYZW3ZT1dxrKYJrPf6u6hBeGexChJF4EN" }, "issuer": { "id": "did:web:verite.id" }, "credentialStatus": { "id": "http://192.168.1.16:3000/api/revocation/05c74310-4810-4ec4-8402-cee4c28dda91#94372", "type": "StatusList2021Entry", "statusPurpose": "revocation", "statusListIndex": "94372", "statusListCredential": "http://192.168.1.16:3000/api/revocation/05c74310-4810-4ec4-8402-cee4c28dda91" }, "issuanceDate": "2021-09-14T02:00:07.000Z" }

Presentation Exchange

QR Code

Credential holders initiate credential exchange flows by scanning a QR code, provided by the verifier, with their credential wallet.

Sample:

JSON { "challengeTokenUrl": "https://verite.id/..." }

Presentation Request

JSON { "id": "1308e77f-9ab0-4de7-97a8-ad2111b585bf", "type": "VerificationRequest", "from": "did:key:z6MkizuwMHiYpZrBAn64ZnbS2cz5og7iGqAa3nV3EuTj4aaZ", "created_time": "2021-09-14T20:19:32.655Z", "expires_time": "2021-10-14T20:19:32.655Z", "reply_url": "http://192.168.1.16:3000/api/verification/1308e77f-9ab0-4de7-97a8-ad2111b585bf/submission", "body": { "status_url": "http://192.168.1.16:3000/api/verification/1308e77f-9ab0-4de7-97a8-ad2111b585bf/callback", "challenge": "e0e52794-7889-451c-bb05-28d8cff9ed13", "presentation_definition": { "id": "KYCAMLPresentationDefinition", ... } } } Details:

- from: who the message is from; in this case, the issuer
- reply_url: the URL the wallet should send the credential submission to
- body.challenge: a challenge the wallet should sign when proving control, to prevent replays
- body.presentation_definition: this follows the DIF Presentation Definition spec
- body.status_url: url returning verification results when complete

Presentation Submission

Note: In the Presentation Object that follows (a signed VP in JWT form), the verifiableCredential object contains one signed verifiable credentials. For clarity & exemplarity, a non-JWT expression in JSON-LD format is included, but the same credential, expressed as a JWT as it is in the rest of this guide, could be substituted for it in JWT format, i.e. verifiableCredential": ["eyJhbGc..."] . JSON-LD and JWT credentials could even be combined, if the relying party was expecting multiple VCs and processed both kinds, i.e. verifiableCredential": ["eyJhbGc...", {"@Context":..."proof":...}] .

JSON { "@context": ["https://www.w3.org/2018/credentials/v1"], "presentation_submission": { "id": "d885c76f-a908-401a-9e41-abbbeddfe886", "definition_id": "KYCAMLPresentationDefinition", "descriptor_map": [{ "id": "KYCAMLCredential", "format": "jwt_vc", "path": ".presentation.verifiableCredential[0]" }] }, "verifiableCredential": [{ "@context": ["https://www.w3.org/2018/credentials/v1"], "type": ["VerifiableCredential", "KYCAMLCredential"], "credentialSubject": { "id": "did:key:z6Mkjo9pGYpv88SCYZW3ZT1dxrKYJrPf6u6hBeGexChJF4EN", "KYCAMLAttestation": { "type": "KYCAMLAttestation", "process": "https://verite.id/definitions/processes/kycaml/0.0.1/usa", "approvalDate": "2021-09-14T02:00:07.540Z" } }, "issuer": { "id": "did:web:verite.id" }, "proof": {...} }

] }

Response

JSON { "status": "approved" } Updated5 months ago * Table of Contents * * Minimal Wallet Requirements - Summary * * Architectural Foundations * * * Concepts Overview * * * Specifications and Libraries Used * * Credential Flows * * * Issuance Flow * * * Verification Flow * * Appendix: Message Samples * * * Credential Issuance * * * Presentation Exchange