# Verification

Verify a Verifiable Credential Suggest Edits

This tutorial will walk you through consuming and verifying verifiable credentials using the verite sdks. For additional context about the verification and credential exchange flow, seeConsuming Verifiable Credentials .

## Step 1: Expose Verification Requirements

The verifier first needs to inform credential holders of their input requirements. Verite uses thePresentation Exchange (PEx) approach for this purpose, in which verifiers express these requirements as a Presentation Definition.

Verite exposes library helpers that create a few predefined types of presentation definitions, for example a KYC verification offer:

JavaScript import { buildKycVerificationOffer } from "verite"

const offer = buildKycVerificationOffer( uuidv4(), verifierDidKey.subject, "https://test.host/verify" )

## Step 2: Receive Presentation Submission

The verification offer wraps a presentation definition, which contains instructions for the credential holder to determine what sort of inputs the verifier requires.

The verification offer also informs the credential holder where to submit the response.

The credential holder uses the presentation definition to gather the required inputs, form a Presentation Submission, submit proof, and send it back to the verifier. The credential holder can take their verifiable presentation, extract the credential they'd like verified, and submit it to the verifier.

Verite exposes libraries to help with this. First, an example of extracting the verifiable credential from a verifiable presentation and decoding it:

JavaScript import { verifyVerifiablePresentation, verifyVerifiableCredential } from "verite"

const decoded = await verifyVerifiablePresentation(presentation) const vc = decoded.verifiableCredential[0] const decodedVc = await verifyVerifiableCredential(vc.proof.jwt) Then, an example of building the presentation submission using the decoded verifiable credential:

JavaScript import { composePresentationSubmission } from "verite"

const submission = await composePresentationSubmission( clientDidKey, offer.body.presentation_definition, decodedVc )

## Step 3: Verify Submission

The verifier verifies the submission and proceeds with the corresponding workflow, returning a status to the credential holder about whether the verification succeeded.

ts await validateVerificationSubmission( submission, offer.body.presentation_definition )

## Summary

Putting this together, we can do the following:

JavaScript // 1. VERIFIER: Discovery of verification requirements const offer = buildKycVerificationOffer( uuidv4(), verifierDidKey.controller, "https://test.host/verify" )

// 2. CLIENT: Decode the verifiable credential from the presentation and create verification submission (wraps a presentation submission) const decoded = await verifyVerifiablePresentation(presentation) const vc = decoded.verifiableCredential[0] const decodedVc = await verifyVerifiableCredential(vc.proof.jwt)

const submission = await composePresentationSubmission( clientDidKey, offer.body.presentation_definition, decodedVc )

// 3. VERIFIER: Verifies submission await validateVerificationSubmission( submission, offer.body.presentation_definition ) You can view this demo as a full working example in ourdemo-issuer demo. Updated5 months ago *Table of Contents * * Step 1: Expose Verification Requirements * * Step 2: Receive Presentation Submission * * Step 3: Verify Submission * * Summary