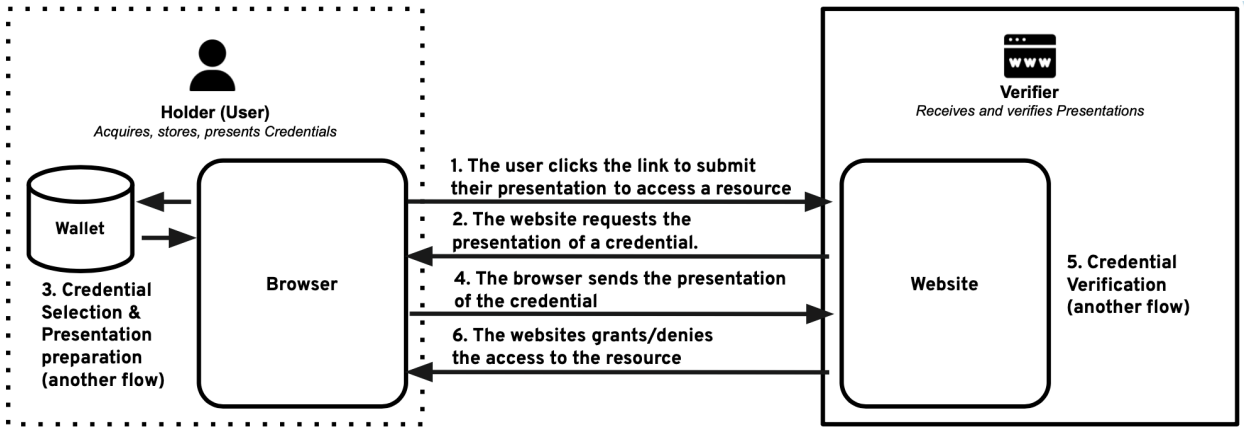
Presenting Credentials on the Web

Editor: Simone Onofri
Status: Editor's Draft

Introduction

Verifiable Digital Credentials can be presented to a mobile wallet or website using different methods. As an example, it is possible to use what happens in the browser.

The user interacts with a wallet to consent to share information with the Verifier. How does the browser hand-control the wallet for credential selection?



This comparison separates three layers that are easy to mix: the invocation mechanism, the presentation protocol, and the component that mediates user choice. A QR code can encode a protocol request or a URI. A custom scheme or app link invokes an installed application through OS dispatch. The Digital Credentials API lets the browser mediate the request and coordinate with platform or wallet components. The security properties depend on how these layers are combined, not only on the visible user gesture.

The visible mechanisms discussed here are QR code-based presentation, custom URL schemes or app links, and browser mediation through the Digital Credentials API.

Other dispatch APIs, such as Web Share, can pass text, files, or URLs to a user-selected target. They are useful at the invocation layer, but they do not by themselves define wallet selection, presentation request validation, or credential response handling.

Each method has trade-offs in security & privacy, usability, and implementation. Below is a comparison of these methods across dimensions such as Security and Privacy, Usability, and Implementation.

This model treats the holder's device, operating system, browser, and wallet as separate trust assumptions. The analysis should make explicit which component mediates the request, which component stores or selects credentials, and where the presentation response is exposed.

Approaches

Before considering wallet-based presentation, it is useful to keep the base case separate. A verifier can ask the user to type data into a form or upload files, such as scanned identity documents. Those documents may be reviewed by the verifier or by a third-party identity-verification provider. For example, Veriff describes AI-supported document and identity verification with human-in-the-loop validation, and Jumio describes automated identity verification for KYC/AML flows. The exact checks, automation level, and data-sharing model are deployment-specific.

In Italy, SPID is a concrete example of this pattern: AgID describes it as the public digital identity system that gives citizens and businesses access to public-administration and participating private services through a single digital identity. Operationally, SPID separates accredited identity providers from service providers, so this base case is closer to federated authentication than to wallet-mediated credential presentation.

This corresponds to the base case, so to "do nothing", and also the default option that ***should*** be present as an alternative to presenting credentials.

Custom URLs Scheme[1]

A **website link or redirect** (like `ageverification://authorize?client_id)` that opens the wallet app on the user's phone. The wallet processes any request parameters and may return the user to the browser upon completion or using a side channel.

From the Apple website [2]: *URL schemes offer a potential attack vector into your app, so make sure to validate all URL parameters and discard any malformed URLs. In addition, limit the available actions to those that don't risk the user's data. For example, don't allow other apps to directly delete content or access sensitive information about the user.*

From the EUDI ARF[3]: *relying on custom URI schemes or universal links introduces variability in User experiences across different browsers and operating systems, resulting in operational inefficiencies and potential security risks.*

---

[1] https://github.com/WICG/digital-credentials/blob/main/custom-schemes.md
[2] https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app
[3] https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/latest/architecture-and-reference-framework-main/

Threats/Attacks[4] [5] [6] [7]

- **Scheme Hijacking**: Another app can register the same scheme, intercepting calls or stealing data.
- **Phishing / Origin Confusion**: The wallet often cannot confirm which site triggered it; malicious or untrusted sites can initiate requests.
- **Data Interception**: If sensitive info is passed back via a URL, other apps could log or intercept it.

QR Code Scan/app link

The website displays a QR code on the same or a separate screen. The user may scan it with a wallet or with a generic camera/OS scanner. A QR code is not itself a presentation protocol: depending on the flow, it can encode a protocol-specific request, a request URI, an OpenID4VP authorization request, or a URI intended for OS dispatch. In proximity flows, QR or NFC can also bootstrap a local secure channel rather than carry the full presentation request.

The main QR-specific issue is user inspectability. The code is machine readable, but the user cannot usually tell which verifier, request object, wallet invocation, or protocol context it represents before a wallet or scanner interprets it.

Threats/Attacks[8] [9]

- **Request substitution**: An attacker can replace a QR code, URI, or request object so that the holder is sent to a different verifier or wallet invocation. The wallet needs to show the verifier identity, requested attributes, and protocol context before approval.
- **QR relay or pre-scan**: An attacker may scan or relay the QR code before the intended holder. Responses need to be bound to a fresh request, verifier session, and intended relying party.

---

[4] https://www.vaadata.com/blog/what-are-deep-links-vulnerabilities-attacks-and-security-best-practices/

[5] https://securityaffairs.com/88469/hacking/ios-url-scheme-flaw.html

[6] https://docs.ostorlab.co/kb/IPA_URL_SCHEME_HIJACKING/index.html

[7] https://developers.googleblog.com/en/improving-user-safety-in-oauth-flows-through-new-oauth-custom-uri-scheme-restrictions/

[8] https://www.yubico.com/blog/qr-codes-within-enterprise-security-key-considerations-and-best-practices/

[9] https://www.cyber.gc.ca/en/guidance/security-considerations-qr-codes-itsap00141

- **Dispatch confusion**: If the QR code is scanned by a generic camera and routed through a custom scheme or app link, the flow can inherit custom-scheme risks such as unclear origin, app selection ambiguity, and request tampering.

Browser API

A web API under standardization that lets a website request Verifiable Digital Credentials through the browser. The browser mediates the request, presents a user-controlled chooser or prompt, and coordinates with platform or wallet components. The browser is a mediator and policy point, not the wallet or credential store.

The browser can observe the presentation request by design, because the user agent needs enough information to mediate the flow and perform risk analysis. The sensitive part is the credential response, plus any request metadata that may reveal too much before the user has a meaningful choice. The response should remain opaque to the browser, with encryption handled by the presentation protocol where supported.

Threats/Attacks

- **Phishing/harvesting**: A malicious website can request credentials or frame a request in a misleading way. Browser mediation can help by showing the requesting origin, requiring user activation, and giving the user a chance to refuse.
- **Replay attack**: A valid credential response or presentation message could be captured and retransmitted if the protocol does not bind it to a fresh request, verifier, and session context.
- **Prompt abuse**: A verifier or malicious website can repeatedly trigger credential prompts or ask for unnecessary credentials. The user agent can require transient activation and rate-limit or block abusive prompts.
- **Session or context binding failure**: A credential response could be associated with the wrong verifier session or browser context. Responses should be bound to a fresh request, verifier session, and intended relying party.
- **Man-in-the-middle**: Presentation messages could be observed or modified if transport security, protocol-level encryption, verifier binding, or session binding is missing. Mitigations include HTTPS/TLS for transport, encrypted credential responses where supported, nonces, and cryptographic binding to the relying party.
- **Information leakage**: Browser/OS mediation can reveal or depend on protocol and credential-manager availability unless constrained. The API should avoid exposing installed wallets, credential availability, hardware properties, or user preferences before user mediation.

Comparison

Method	Security	Usability	Implementation
Custom URL Scheme	**Low/conditional: custom schemes can be hijacked or misrouted when scheme ownership, origin display, and parameter validation are weak. Sensitive data should not be returned in URLs.**	**Conditional: quick app switch when configured; confusing when several apps or handlers can respond, or when the flow fails silently.**	**Low initial effort, higher integration risk: handlers, redirects, return paths, and parameter validation are app- and platform-specific.**
QR Code	**Conditional: security depends on the presentation protocol, freshness, verifier/session binding, and wallet UI. QR tampering is mainly an inspectability and request-substitution problem.**	**Good for cross-device flows; extra friction on single-device flows unless the wallet/OS handoff is clear.**	**Moderate: QR generation is simple, but the encoded request, request URI, or proximity bootstrap has to follow the presentation protocol and use short-lived, bound requests.**
Browser API	**Conditional: browser/OS mediation can reduce phishing, prompt abuse, and origin-confusion when origin display, user activation, and credential-manager selection are implemented. It still depends on protocol binding and response confidentiality.**	**Potentially strong for one-device flows when prompts are clear and wallet choice remains under user control.**	**Moderate/high: depends on user-agent support, OS or credential-manager integration, and compatible presentation protocols.**

Considerations

Each method involves trade-offs between security, privacy, usability, and deployability. Over-disclosure is a general threat across all methods: any presentation can leak more data than necessary if the protocol, verifier request, wallet UI, or credential format does not support selective disclosure and clear user review. The Digital Credentials API is promising because it gives the user agent a mediation point, but its security properties still depend on the presentation protocol, wallet behavior, and platform integration. QR-based flows remain useful, especially cross-device, but their security depends on request binding and wallet UI. Custom schemes remain a compatibility option, but their security and interoperability weaknesses should be documented[10] [11].

The Digital Credentials API gives the user agent a place to contextualize a credential request before it reaches the wallet. Browser mediation can reduce some phishing, prompt-abuse, and origin-confusion risks when the user agent can show origin, require user activation, and constrain credential-manager selection. It does not remove the need for protocol-level binding, response confidentiality, and wallet-side review. The privacy claim should stay narrow: the API can support privacy-preserving presentation, but it cannot by itself guarantee unlinkability or prevent all issuer involvement once the request moves into the credential manager and presentation protocol.

[10] https://blog.timcappalli.me/p/preso-nistmdl24-dcapi/nistmdl24-dcapi.pdf
[11] https://github.com/WICG/digital-credentials/blob/main/explainer.md

Two elements matter for interoperability and user choice: wallet selection should remain under user control, and the architecture should not assume only one wallet form factor. Mobile, desktop, and Web wallets can all be in scope, but each needs a defined invocation path and a compatible presentation protocol binding for the request and response. A Web wallet therefore needs more than the JavaScript API entry point; the protocol still has to define how the verifier, wallet, and browser exchange presentation messages.

Further readings

- Rick Byer's Credentials consideration[12]
- High Level Threat model[13]

Appendices

Appendix A - Requirements

A useful way to understand what is being built is to name the assets and properties at stake. In this case, the assets include the holder's data, credentials, wallet choices, verifier session, and presentation response. The properties include selective disclosure, verifier binding, user consent, unlinkability where supported, and clear responsibility boundaries between the browser, OS, wallet, verifier, issuer, and presentation protocol.

These requirements can be described through Web design principles, identity and privacy principles, and the regulatory context around eIDAS 2.0 and the EUDI Wallet. The EUDI ARF discusses use of the Digital Credentials API for some remote presentation flows, but the API is still under development and should not be described as a fixed legal requirement.

These requirements need to be considered, at the API level, particularly to define limits for the release of information, some bad patterns that must not be implemented, as in the default options to be presented to the user to avoid dark patterns, as they are nudges[14]:

Any aspect of the choice architecture that alters people's behavior in a predictable way without forbidding any options or significantly changing their economic incentives.

[12] https://github.com/w3c/credential-considerations/blob/main/credentials-considerations.md
[13] https://github.com/w3c-cg/threat-modeling/blob/main/models/decentralized-identities.md
[14] https://www.behavioraleconomics.com/resources/mini-encyclopedia-of-be/nudge

These nudges can also be used in a positive sense, for example by alerting the user to a request from a Verifier for a full credential (and not selective disclosure), as well as if the credential being used uses technologies that call home or release information to the issuer.

W3C Principles on Identity and Privacy

Two main W3C Design principles[15] are "*1.4. Ask users for meaningful consent*" and "*1.5. Use identity appropriately in context*". Quoting the relevant parts:

Features that use or depend on identifiers and the attachment of data about a person to that identifier carry privacy risks *which often reach beyond a single API or system. This includes data that has been passively generated (for example, about their behaviour on the web) as well as that which* ***has been actively collected*** *(for example, they have filled in a form).*

For such features, you should ***understand the context in which it will be used, including how it will be used alongside other features of the web****. Make sure the user can give* ***appropriate consent****. Design APIs to* ***collect the smallest amount of data necessary****. [...]*

In the context of fulfilling a user need, a web page may want to make use of a feature that has the potential to cause harm. Features that have this potential for harm should be designed such that people can ***give meaningful consent for that feature to be used, and that they can refuse consent effectively****.*

In order to give meaningful consent, the user must: ***understand what permission they may choose whether to grant the web page; be able to choose to give or refuse that permission effectively****.*

Relevant W3C Privacy principles[16] are Principle 2.1 "A **user agent should help its user present the identity they want** in each context they are in, and **should prevent** or support recognition as appropriate", and Principle 2.15.1 "User agents should **support people in choosing which information they provide to actors that request it, up to and including allowing users to provide arbitrary information**"

eIDAS 2.0 on privacy requirements

[15] https://www.w3.org/TR/design-principles/
[16] https://www.w3.org/TR/privacy-principles/#support-choosing-info

eIDAS 2.0 regulation[17] from which is possible to derive some technical requirements, as described by cryptographer's feedback[18] summarized here[19] and , focused on Privacy Loss:

- **Selective Disclosure**: users can decide to limit the information in a presentation.
- **Unlinkability**: two verifiers should not recognize the same holder, issuer should not know when a credential is presented or verified, even with the collusion of verifier and issuer.
- **Pseudonymous Authentication**: don't use unique identifiers unless necessary.
- **Non-Transferability**: must not use fraudulent or shared credentials.

[17] https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation
[18] https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/issues/200
[19] https://docs.google.com/presentation/d/1OX1xRGUKRS3O5x7ajTaLMT0sn0-fBaZCWctHi_AQYrs/edit#slide=id.g305652a2a73_0_0