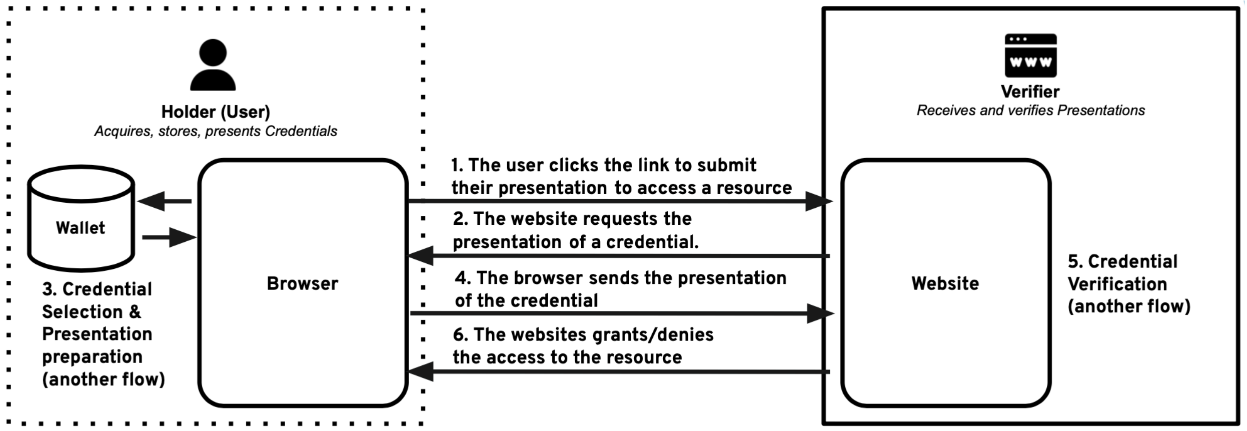
Presenting Credentials on the Web

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?



Holder (User)
Acquires, stores, presents Credentials
Wallet
Browser
3. Credential Selection & Presentation preparation (another flow)
1. The user clicks the link to submit their presentation to access a resource
2. The website requests the presentation of a credential.
4. The browser sends the presentation of the credential
6. The websites grants/denies the access to the resource
www
Verifier
Receives and verifies Presentations
Website
5. Credential Verification (another flow)

The three main approaches described in EUDI ARF: **QR code scan**, a **custom URL scheme/app link**, or a **Browser API (also known as Digital Credentials API)**.

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.

Concerning trust, the holder generally trusts their software (OS, Wallet, Browser) and their hardware (Desktop, Mobile).

Approaches

Before considering the model based on Wallet, it is worth describing how the base case works. With the user entering his or her data via input type text, files (e.g., ' scanned documents) to be verified are normally analyzed by other people or - in the specific case of KYC/AML - by third-party companies that may also use AI systems.

In the specific context of identifying a user in a governmental setting, it is often used as a means of centralized or federated authentication.

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.*

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.

---

[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/
[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/

QR Code Scan/app link

The **website displays a QR code** (on the same or a separate screen); the user opens their wallet and scans it.

In general, QR Codes suffer from a structural problem: they are Machine Readable and not Human Readable, therefore it is difficult for the user to understand if it is a malicious request.

Threats/Attacks[8] [9]

- **Fake/Tampered QR**: Attackers can replace or alter QR codes to point to malicious servers ("quishing").
- **Man-in-the-Middle (QRLjacking)**: Attackers might scan or clone the code before the rightful user, and hijack the session.
- **Data Leakage**: Private details could be overexposed if the protocol or wallet app shares full ID info or fails to limit data.

Browser API

A **standardized web API** that lets the website request Verifiable Digital Credentials directly through the browser. The browser mediates this request, requiring user consent, and coordinates with the OS/wallet IPC to only share approved data.

One of the main threats is related to the fact that the browser can observe the credential request. Given that this is similar to the Threat Model of what we enter as payment information, a password or an email as input type text or password (which the browser can easily understand what it is), as well as WebAuthn or a Secure Payments Confirmation. It can be controlled and balanced by defining, on the one hand, the data visible to the browser by the API (e.g., by encrypting it) and, on the other hand, using a side channel.

Threats/Attacks

- **Phishing/Harvesting**: Malicious websites could request credentials. Mitigations: The browser shows the requesting domain, requires explicit user approval, and ensures safe browsing.
- **Reply Attack**: Where valid digital credential messages are maliciously captured and retransmitted to authenticate or authorize actions fraudulently.
- **API Abuse**: Potential spamming or misuse of the credential prompt. Mitigations: The browser enforces user interaction and can limit repeated prompts.

---

[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

- **Session Hijacking**: Attackers might try to replay or intercept credential responses. Mitigations: Use nonces, one-time tokens, and TLS to prevent reuse or interception.
- **Man-in-the-Middle**: Credential data could be spied on in transit. Mitigations: Use end-to-end encryption (TLS/SSL) and cryptographic binding to the relying party.
- **Information Leakage**: The Browser API scheme can expose wallet's metadata to the OS and browsers. Even if the Browser already have access to the information even if shared using QR Codes and Schemes.

Comparison

Method	Security	Usability	Implementation
Custom URL Scheme	**Low**: It is Easy to hijack, has no built-in domain check, and is prone to oversharing. Its origin is also unclear, and it can easily leak data.	**Moderate** Potentially quick app switch, but can be confusing or fail silently.	**Easy** to implement but full of pitfalls (scheme conflicts, no standard).
QR Code	**Moderate**. It depends on robust cryptographic protocols, and code tampering is risky. It is user-initiated, but domain transparency depends on wallet UI, and there is a risk of scanning fake codes.	**Moderate** Great for cross-device; slightly more friction on single-device	**Moderate** Uses standard libraries for QR & OID flows must ensure short-lived tokens
Browser API	**High** Browser + OS mediation via IPC/other means prevents phishing & interception, consent screens, and origin-binding. Relatively increases the attack surface.	**High** Seamless 1-device flow, clear user prompts	**Moderate** Requires modern browser support & wallet integration.

Considerations

Each method involves trade-offs between security and convenience. The Digital Credentials API (Browser API) method offers a compelling balance – high security with a reasonable UX; QR codes provide a device-agnostic, fairly secure alternative that is usable today at the cost of a bit more user effort; Custom schemes should be phased out due to their security weaknesses[10] [11], though they remain an important compatibility option for now.

Concerning the Digital Credential API, which is the newest solution and is still under study, it is possible to create a trust boundary between the browser and the wallet (therefore exposing a subset of information related to the request for presentation to the browser), guaranteeing the privacy of the holder and still taking advantage of the protections offered by the browser. However, the user agent could be considered a security element (e.g., blocking malicious

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

verifiers before the wallet, possibly alerting the user in case of non-privacy-preserving credentials, limiting the telemetry information collected for the credential).

Two elements must be considered regarding interoperability and freedom: the Default Wallet must be chosen by the user (avoiding Wallet War™), and the user must be free to choose their wallet type (i.e. Web, Mobile, Desktop).

Further readings

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

Appendices

Appendix A - Requirements

A key aspect when understanding "what we are building" is related, to understanding what are the assets to be protected (in this case, the user's data) and what the requirements needs to be followed and properties needs to be assured

This can be described both by some principles, in particular while designing a web feature but in general also on presenting credentials on the web, as well as by regulations (as the Digital Credentials API will be used in eIDAS 2.0) use case are and thus, in this specific case user data and credentials.

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.

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.

[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

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

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.

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

- **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.