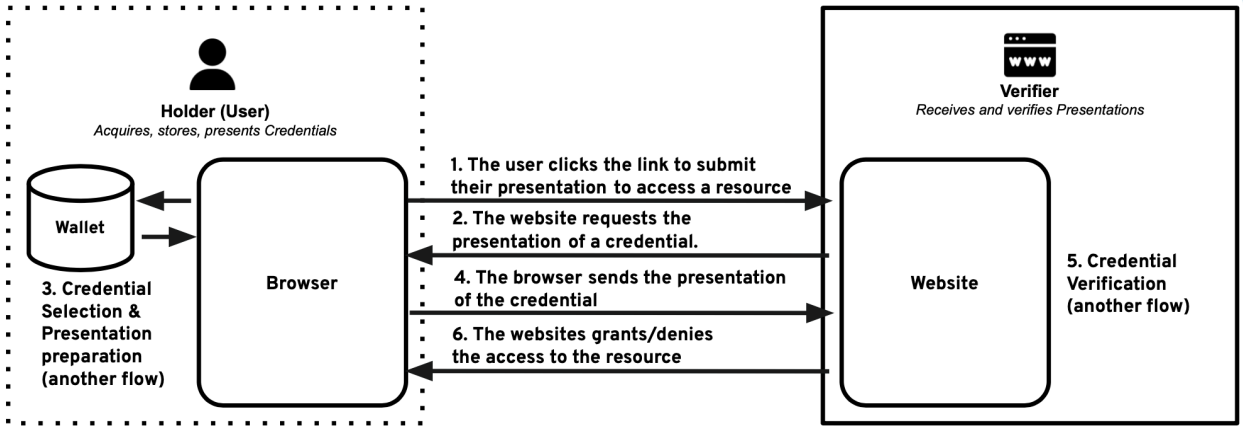
Threat Model for Presenting Verifiable Digital Credentials on the Web

Editor: Simone Onofri
Status: Editor's Draft

Introduction

This document is a threat model for presenting Verifiable Digital Credentials on the Web. It focuses on the case in which a user is already using a browser to visit a verifier's website. The verifier's website starts a presentation flow so that the user can present a credential and the verifier can receive and verify the resulting presentation.

The document compares the techniques used to connect the verifier's website, the browser, and the wallet or credential manager, together with the protocol-specific data carried through those techniques. It builds on the Threat Model for Decentralized Credentials and on A Threat Model for the W3C Digital Credentials API: An Initial Analysis, while focusing on the threats introduced by presenting credentials from a Web context.



The main approaches considered are QR codes, custom URL schemes or app links, and browser mediation through the Digital Credentials API. Other dispatch mechanisms, such as Web Share, are considered as alternatives.

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 is the baseline “do nothing” case for wallet-mediated presentation. As a design recommendation, an appropriate non-wallet alternative should remain available; in EUDI Wallet contexts, access should also remain possible through other existing identification and authentication means.

Custom URL schemes and verified app links[1]

A private-use URI such as ageverification://authorize?client_id can ask the operating system to open a registered wallet application. A verified HTTPS link, such as an Apple Universal Link or Android App Link, additionally associates a web domain with an application. Neither dispatch mechanism, by itself, authenticates the credential verifier or binds a presentation response to the intended browser session.

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]: *Discussion Topic F identifies cross-browser and cross-platform variability, operational inefficiencies, and potential security risks when remote presentation relies on custom URI schemes or universal links. This is discussion material, not a fixed normative requirement of the Architecture and Reference Framework.*

From Historical implementation studies by Liu et al. *and* Xing et al[4]. *document scheme and deep-link hijacking and cross-app URL-scheme attacks. They support the attack class, but not its current prevalence or a finding against correctly verified HTTPS app links.*

---

[1] https://github.com/WICG/digital-credentials/blob/main/custom-schemes.md
[2] https://developer.apple.com/documentation/xcde/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.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/liu; https://doi.org/10.1145/2810103.2813609

Threats/Attacks[5] [6] [7] [8]

- **Callback interception and scheme collision**: Private-use scheme ownership is not exclusive, so another installed application may receive the URI and its parameters. The impact depends on whether the URI carries bearer or sensitive material and whether the presentation protocol cryptographically binds the response to the intended verifier and session.
- **Origin confusion**: Custom-scheme dispatch does not itself authenticate the invoking web origin. A presentation protocol can authenticate the verifier separately, but the wallet needs enough authenticated context to distinguish the verifier from the page or application that initiated the dispatch.
- **Sensitive-data exposure**: Data placed in a private-use URI can be exposed to a competing handler or to components that receive or record that URI. Failure behavior and exposure depend on the operating system, browser, application, and protocol design.

QR code scan

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 instead of carrying the full presentation request.

One QR-specific issue is user inspectability. The code is machine-readable, but users usually cannot tell which verifier, request object, wallet invocation, or protocol context it represents before a wallet or scanner interprets it. A QR-only flow also creates an accessibility barrier; the current Digital Credentials API Working Draft calls for an equivalent accessible means when cross-device presentation uses a visual artifact. In three experiments with 1,876 participants, Kowalewski et al.[9] found that only 13% recognized fraudulent QR payment requests, compared with 46% when participants entered payment details manually.

A barcode whose contents are cryptographically verifiable provides stronger authenticity and integrity properties than an unsigned barcode. Verifiable Credential Barcodes is one W3C example: it protects optical barcode contents using Verifiable Credentials. This property applies to the authenticated barcode data, not to the context in which an image is displayed. A page can

---

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

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

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

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

[9] https://www.usenix.org/conference/usenixsecurity25/presentation/kowalewski

still replace an unsigned barcode with another value, or replace a cryptographically valid barcode with a different valid barcode. Detecting that substitution depends on binding the authenticated data to the intended Web origin, user interaction, browser session, or transaction. Barcode protection, presentation-protocol security, and browser mediation are separate layers. The Digital Credentials API can mediate the interaction and make browser-derived context available to the mediated flow, but it is designed to be agnostic to credential formats and to keep the act of requesting a credential separate from the presentation protocol.

Threats/Attacks[10] [11]

- **Signed barcode and display substitution**: A cryptographically verifiable barcode provides stronger authenticity and integrity properties than an unsigned barcode. Verification still does not show that the displayed code is the one intended for the current page, origin, user interaction, session, or transaction. A compromised relying-party page can replace the displayed code with a different cryptographically valid barcode. Detecting or preventing that substitution requires a separate binding between the barcode and the relevant context, supplied by authenticated data, the presentation protocol, browser mediation, or a combination of these mechanisms.
- **Same-device capture provenance**: A wallet or scanner that opens a screenshot or other capture of a code rendered inside the relying party's page cannot infer the code's provenance from the image alone. Page content or scripts can replace the displayed code; this includes scripts injected through a cross-site scripting (XSS) vulnerability in the relying party. Signed fields reduce undetected alteration, but not wrong-context substitution unless the signed data is bound to the current origin, user interaction, and session. The flow should also provide an accessible equivalent to the visual artifact.
- **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. Wallet and platform UI should expose enough authenticated verifier context and requested-data detail to support informed authorization; the exact mandatory elements depend on the presentation protocol and profile.
- **Cross-device consent phishing and session fixation**: An attacker can obtain or relay a verifier's QR code and present it to a holder in a misleading context. With an out-of-band response such as OpenID4VP direct_post, the presentation may complete a session controlled by the attacker unless the request is authenticated, fresh, single-use, and bound to the intended verifier, relying party, and browser or device context. Clear verifier and action display, nonce and audience binding, and platform-enforced proximity or an equivalent session-binding control reduce this risk. RFC 10027 documents an illicit-onboarding scenario in which an attacker presents a victim with a wallet-invocation QR code and receives the resulting credential-based onboarding session. Adjacent

---

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

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

evidence from Zhang et al.[12] found authorization hijacking, double login, brute-force login, universal account takeover, and privacy abuse in QR-login deployments. Those deployments are not OpenID4VP, so the study supports fresh, single-use, session-bound identifiers by analogy rather than proving a Digital Credentials API vulnerability.

- **Scanner-to-handler confusion and parameter abuse**: A crafted QR code can cause a wallet or scanner to invoke an unintended handler when decoded content is dispatched without strict parsing, allow-listing, origin restrictions, and user confirmation. Treat QR payloads as untrusted input and isolate presentation parsing from generic deep-link or WebView actions. Han et al.[13] examined 800 popular Android and iOS apps, found 123 apps with 2,872 vulnerable remotely accessible handlers, and confirmed 46 apps with critical or high-severity vulnerabilities. This is direct evidence that malicious QR input can reach sensitive app functions when scanning components dispatch data without validation; it is not evidence of an OpenID4VP or Digital Credentials API flaw.

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.

By design, the browser can observe the presentation request 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 API is designed around opaque, encrypted responses; the selected presentation protocol and profile provide encryption, audience binding, and session binding.

Threats/Attacks

- **Phishing and harvesting**: A malicious website can request credentials or frame a request in a misleading way. Browser mediation helps by requiring user activation, providing context, 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 request unnecessary credentials or try to trigger misleading prompts. The API requires transient activation and mediation for each operation; user agents may apply additional anti-abuse policy.
- **Session or context binding failure**: A credential response could be associated with the wrong verifier session or browser context. In a cross-device flow, the credential manager

[12] https://www.rfc-editor.org/rfc/rfc10027.html; https://openid.net/specs/openid-4-verifiable-presentations-1_0.html#name-session-fixation; https://www.usenix.org/conference/usenixsecurity25/presentation/zhang-xin
[13] https://www.usenix.org/conference/usenixsecurity23/presentation/han-xing

cannot inherently trust an origin string forwarded by the primary device. Browser-provided origin context, cryptographic verifier authentication, request signing, proximity, and session binding are separate controls and should not be conflated.

- **Request tampering and malicious payloads**: HTTPS does not prevent scripts already executing in the verifier page, including compromised dependencies or malicious extensions, from modifying an unsigned request or replacing response-encryption parameters. A malicious website can also submit crafted protocol data intended to exploit parsing or validation weaknesses in the operating system or credential manager. Request signing helps only when the wallet can authenticate the signing key as belonging to the intended verifier. The API requires safe JSON serialization; protocol-specific parsing and validation remain responsibilities of the platform and credential manager.
- **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.

Remediation

- **Same-device origin-bound response**: The Digital Credentials API supplies the top-level origin as request context. In OpenID4VP[14] over the API, the response audience is the origin prefixed with origin:, and replay protection uses the origin instead of the client identifier. This binds the presentation response to the browser-supplied origin; it does not, by itself, sign the request or authenticate the verifier.
- **Same-device direct_post session fixation**: OpenID4VP[15] can detect and stop this attack when the response endpoint returns a redirect_uri containing a fresh response code and requires the original frontend to present that code before it retrieves the authorization response. The protection depends on the attacker not obtaining the code.
- **Cross-device origin and session binding**: The same redirect-based technique does not apply when the wallet-side browser does not share the original verifier session. A credential manager also cannot inherently trust an origin string forwarded by the primary device. Use an authenticated or signed request whose key is recognized as the verifier's, fresh nonce and state values, response audience and session binding, a short validity window, and platform-enforced proximity[16] or an equivalent channel-binding control.

[14] https://www.w3.org/TR/digital-credentials/; https://openid.net/specs/openid-4-verifiable-presentations-1_0.html
[15] https://openid.net/specs/openid-4-verifiable-presentations-1_0.html#name-session-fixation
[16] https://www.w3.org/TR/digital-credentials/#cross-device-requests; https://www.rfc-editor.org/rfc/rfc10027.html#section-6.1.1

Comparison

The following ratings are qualitative assessments under the stated assumptions. Actual risk depends on protocol binding, verifier authentication, platform routing, wallet UI, and deployment.

Method	Security	Usability	Implementation
Custom scheme / app link	🔴 **Low**: private-use schemes can be hijacked or misrouted. Verified HTTPS app links improve handler routing, but do not authenticate the verifier or bind the presentation to the originating browser session.	🟡 **Conditional**: app switching can be quick, but behavior depends on installed handlers, platform configuration, and fallback paths.	🟡 **Moderate**: initial integration is simple, but schemes, association files, strict parsing, redirects, return paths, and fallbacks remain application- and platform-specific.
Standalone/ custom QR code	🔴 **Low**: a QR handoff provides no verifier authentication, origin or session binding, freshness, single use, or relay protection by itself. The surrounding protocol must supply these controls.	🟡 **Mixed**: familiar for cross-device handoff, but opaque before scanning and dependent on vision, a camera, and a second device. Platforms SHOULD provide an equivalent accessible path.	🟡 **Moderate**: encoding is simple; secure request retrieval, expiry, anti-replay, proximity, error recovery, and scanner input validation are not.
Digital Credentials API	🟢 **Stronger baseline**: browser and operating-system mediation add secure-context restriction, transient activation, required mediation, Permissions Policy, and platform-supplied origin context. Cross-device assurance still requires verifier-authenticated or signed requests, proximity, protocol binding, encryption, and session binding.	🟢 **Strong where supported**: platform wallet selection replaces custom handoffs and can manage same- and cross-device presentation. Accessibility and availability still depend on the operating system, wallet, credential, and allowed protocol.	🟡 **Moderate and improving**: presentation is enabled by default in Chrome 141+ and supported in Safari 26+.[17] Deployments still need feature and protocol detection, compatible wallets, server-side verification, interoperability testing, and fallback.

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 gives the user agent a mediated entry point and several baseline controls, but end-to-end security still depends on the selected presentation protocol, authenticated request context, response encryption, session binding, wallet behavior, and platform implementation. QR codes are carriers or bootstrap mechanisms, not security protocols. Private-use custom schemes are operating-system dispatch mechanisms; verified HTTPS links additionally bind a web domain to an application.

[17] https://developer.chrome.com/blog/digital-credentials-api-shipped; https://webkit.org/blog/17431/online-identity-verification-with-the-digital-credentials-api/

None of these mechanisms, by itself, authenticates the verifier or binds a credential response to the intended browser session[18] [19].

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 by supplying origin context, requiring user activation, and constraining 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.

Two elements are important 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[20]
- Digital Credentials[21]
- Threat Model for Decentralized Credentials[22]
- OpenID for Verifiable Presentations 1.0[23]
- RFC 10027[24]
- RFC 8252[25]
- Medusa Attack: Exploring Security Hazards of In-App QR Code Scanning[26]
- Scanned and Scammed: Insecurity by ObsQRity?[27]
- Demystifying the (In)Security of QR Code-based Login in Real-world Deployments[28]
- Measuring the Insecurity of Mobile Deep Links of Android[29]
- Cracking App Isolation on Apple[30].

---

[18] https://blog.timcappalli.me/p/preso-nistmdl24-dcapi/nistmdl24-dcapi.pdf
[19] https://github.com/WICG/digital-credentials/blob/main/explainer.md
[20] https://github.com/w3c/credential-considerations/blob/main/credentials-considerations.md
[21] https://www.w3.org/TR/digital-credentials/
[22] https://www.w3.org/TR/threat-model-decentralized-credentials/
[23] https://openid.net/specs/openid-4-verifiable-presentations-1_0.html
[24] https://www.rfc-editor.org/rfc/rfc10027.html
[25] https://www.rfc-editor.org/rfc/rfc8252.html
[26] https://www.usenix.org/conference/usenixsecurity23/presentation/han-xing
[27] https://www.usenix.org/conference/usenixsecurity25/presentation/kowalewski
[28] https://www.usenix.org/conference/usenixsecurity25/presentation/zhang-xin
[29] https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/liu
[30] https://doi.org/10.1145/2810103.2813609

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[31]:

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.

W3C Principles on Identity and Privacy

Relevant TAG Web Platform Design Principles[32] include section 1.4, "Design for user intent", especially section 1.4.1, "Help users make good decisions", and section 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).*

[31] https://www.behavioraleconomics.com/resources/mini-encyclopedia-of-be/nudge
[32] https://www.w3.org/TR/design-principles/

*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[33] 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

Regulation (EU) 2024/1183[34] provides a regulatory basis for several privacy properties. The following summary distinguishes the regulation's requirements from additional technical interpretations[35][36], with a focus on privacy loss:

- **Selective Disclosure**: users can decide to limit the information in a presentation.
- **Unlinkability**: the wallet framework must prevent post-issuance tracking or correlation unless explicitly authorized, and support unlinkability where an attestation does not require identification.
- **Pseudonymous Authentication**: don't use unique identifiers unless necessary.
- **Holder binding and unauthorized use**: protection against unauthorized credential use is a security objective whose exact requirements depend on the credential format, presentation protocol, and applicable implementing rules.

[33] https://www.w3.org/TR/privacy-principles/#support-choosing-info
[34] https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation
[35] https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/issues/200
[36] https://docs.google.com/presentation/d/1OX1xRGUKRS3O5x7ajTaLMT0sn0-fBaZCWctHi_AQYrs/edit#slide=id.g305652a2a73_0_0