Work in progress

This is an early draft of a chapter of a book on the foundations of cryptographic authentication being coauthored by Francisco Corella, Sukhi Chuhan and Veronica Wojnas. Please send comments to the authors.

12. Credential wallets

Chapter summary

In Section 6.4 of Chapter 6, the Mobile Technology chapter, we discussed *mobile wallets* as a highly specialized kind of mobile native app, exemplified by Apple Wallet and Google Wallet. As we saw in that chapter, those wallets are now beginning to carry credentials. But in this chapter, we are concerned with wallets intended specifically for carrying credentials.

The emergence of credential wallets can be credited to Satoshi Nakamoto's C++ program *bitcoin*, which implemented a payment wallet along with a blockchain. For that reason, the chapter begins with a section on cryptocurrency wallets, even though most credentials used for cryptographic authentication are not payment credentials. That section is followed by a short section explaining how decentralized identifiers also originate from bitcoin and some are implemented on the bitcoin blockchain.

Governments are taking the lead in implementing credential wallets and Section 12.3 describes several implementations or projected implementations of government wallets, including the Bhutan NDI wallet, the British Columbia wallet, the California DMV wallet, and the EUDI Wallet.

Then in Section 12.4 we show how a web browser can serve as a secure wallet to store cryptographic credentials, including traditional public key certificates, selective disclosure public key certificates that bind the public key to a redactable digest of the attributes, and anonymous credentials that provide unlinkability in addition to selective disclosure; and how storing such credentials in a browser supports same-device, cross-device and IoT presentation. As a special case, in Section 12.4.3.6 we show how storing the mobile driver's license (mDL) credential in a WebView provides efficient same-device and cross-device presentation, and proximity presentation with mitigation of an unauthorized disclosure vulnerability.

12.1 Cryptocurrency wallets

12.1.1 The Satoshi wallet

Credential wallets can be traced back to *bitcoin*, the program that was released as open source by Satoshi Nakamoto in 2009, after publishing his white paper “Bitcoin: A Peer-to-Peer Electronic Cash System” [1], as the only authoritative specification of the bitcoin protocol. The program still runs today. It is maintained by a community of volunteer developers and available on GitHub [2].

Although Nakamoto’s program is known today as “the original bitcoin client”, it is not a client in the sense of client-server architecture, as it does not communicate with any server. It is written in C++ and thus runs under any operating system in any kind of device, whether a mobile device, a desktop or laptop, or a server. Bitcoin cryptocurrency lives on a peer-to-peer network of nodes, all of which originally ran the same bitcoin program and today run software with equivalent functionality. The bitcoin program is self-contained, storing the entire blockchain in the file system of the device where it runs, so nodes communicated on the peer-to-peer network only to send each other transactions, as we saw in Chapter 9. The original version of the program also stored in the file system a *wallet file*, containing the wallet owner's transaction history, address book and private keys [3], the private keys being used to sign transactions as described in Chapter 9. This wallet file, that could be called *the Satoshi wallet*, is thus the first credential wallet.

12.1.2 Full vs. thin clients

The size of the Bitcoin blockchain was 2 GB as of May 2012. It reached 600 GB by September 2024. Storing the entire blockchain in the file system of the device where the program is running would have been impossible for the original implementation, and is still problematic today, not only because of the sheer size of the data structure, but also because of the network bandwidth required by having to be aware of every transaction that is created.

This problem is addressed by offloading the storage of the blockchain from the device of the user who owns bitcoin and issues transactions to a server. The server is then a node of the peer-to-peer network, while the user’s device is a client of the node. The user’s device is called a *thin client*, while the node is called a *full client*.

Besides storing the blockchain, the full client may also store the private keys and provide a graphical user interface for issuing transactions that the user accesses through a web browser. As we saw in Chapter 9, bitcoin private keys are used to sign transactions that send currency to a different user, so stealing private keys is equivalent to stealing bank nodes. Entrusting the storage of the private keys to a server can be a good thing or a bad thing. If the server is implemented and run by honest personnel competent in cybersecurity, the private keys may be better protected in the server than in the user’s machine. But hundreds of thousands of bitcoins worth fiat currency equivalent to hundreds of millions of US dollars were stolen from customers of Mt. Gox before it ceased operations in 2014 [4].

12.1.3 A Web3 wallet

It is also possible to offload the user interface for issuing transactions to a full client while keeping the private keys in the user's device. This can be achieved using a native app or a browser extension provided by the full client that runs in the user's device and takes care of generating and storing the private keys. This approach is used by the etaMask wallet [5], a Web3 wallet for Ethereum decentralized applications (Dapps) [6].

12.2 From cryptocurrency wallets to identity wallets

The bitcoin blockchain provided the initial inspiration for decentralized identifiers, as Namecoin, a blockchain derived from and merge-mined with Bitcoin, demonstrated the feasibility of registering human-meaningful, memorizable decentralized identifiers, disproving Zooko's trilemma [7]. Decentralized identifiers are no longer memorizable, but decentralized identifiers are still registered in the Bitcoin or Ethereum blockchains. For example, we shall see in Section 14.2.1 how decentralized identifiers with DID method "btcr" (an acronym for BiTCoin Reference) are registered on the bitcoin blockchain, and the authentication and assertion methods in their DID documents are public keys associated with private keys used to sign bitcoin transactions and stored in bitcoin wallets. It would thus make sense to repurpose a bitcoin cryptocurrency wallet as an identity wallet carrying verifiable credentials, or as a multipurpose wallet carrying both bitcoin key pairs and verifiable credentials.

12.3 Government wallets

Most authoritative physical credentials have traditionally been issued by governments. Therefore, it should not come as a surprise that governments are taking the lead in the deployment of digital identity credentials as part of their digital transformation efforts.

A UL whitepaper [8] observes that governments face the choice between two kinds of digital credentials: verifiable credentials, and the mobile driver's license (mDL) of the ISO/IEC 18013-5 standard. Each kind of credential comes with its kind of wallet, but the UL white paper discusses possible ways of avoiding the choice of credential by carrying both in the same wallet. We shall discuss in detail the mDL in Chapter 13, verifiable credentials in Chapter 14, and methods of combining them in Chapter 15. In this section we review the wallets that are being implemented or deployed by several governments and the kinds of credentials that they are storing in those wallets.

12.3.1 The Bhutan National Digital Identity (NDI) wallet

The Bhutan wallet is provided to participants in Bhutan's national identity system and used store and manage verifiable credentials. As of this writing, Bhutan's identity system, launched in 2023, is the only fully functional and operational deployment of verifiable credential technology in the world. It relies on two registries: the NDI Trust

Registry, and the NDI Verifiable Data Registry. The Trust Registry contains a list of public DIDs of trusted organizations that participate in the NDI ecosystem. The Verifiable Data Registry is built on Hyperledger Indy blockchain and contains the public keys of credential issuers, DID documents, and other metadata.

According to [35], the purpose of the NDI system is to facilitate "access to public services" and "trusted peer-to-peer interactions between individuals, government, and organizations". To participate in the system, citizens go through an onboarding process where they validate personal data and provide a facial scan that is compared to their biometric data in the database of the Department of Civil Registration and Census (DCRC). They then get a Foundational ID credential with a Citizen ID (CID) number attribute that is signed by the DCRD and stored in the NDI wallet. Foreigners get a Foundational ID from the Department of Immigration (DoI) instead.

Other verifiable credentials may be stored in the wallet include a permanent address credential, an academic credential, and credentials for employment details, mobile number, driver's license, etc., as well as credentials for self-asserted data.

A unique feature of the Bhutan wallet is that it can establish a persistent connection with an organization such as a bank and maintain it using session-based authentication. Persistent connections are credited with improving user login experience and facilitating compliance of the organization with know-your-customer (KYC) regulations.

12.3.2 The British Columbia wallet

The British Columbia wallet is a mobile app that carries verifiable credentials. According to the BC Wallet web site [11], it is based on technology used since 2019 in OrgBook BC, a searchable list of organizations in British Columbia. The OrgBook BC web site [12] shows impressive statistics about current usage: 1552699 active legal entities, 4527091 verifiable credentials held, and 7193 credentials added this week, as of December 14, 2024.

The BC Wallet itself, however, only supports a small number of use cases at the moment. It is currently being used by lawyers to gain access to court materials, and to log in remotely to subscription databases.

It is also being used by sole proprietors to carry credentials usable to apply for business licenses and permits with municipalities in the province.

The BC Wallet is available in app stores, but the web site points out that "most people will not have a need for BC Wallet right now", as "it does not support credentials that work in Apple Wallet or Google Wallet".

The code that implements the app is available as open source on GitHub, at [11]. An architecture diagram can be retrieved from [12].

12.3.3 The California DMV wallet

While the Ontario and British Columbia wallets are intended for carrying verifiable credentials, the wallet of the California Department of Motor Vehicles (CA DMV) carries the CA driver's license or state ID credential implemented according to the ISO/IEC mobile Driver's License (mDL) standards 18013-5 and 18013-7 described in the next chapter. The same credential is also available in Google Wallet, and in Apple Wallet on iPhone and Apple Watch.

The CA DMV has a web site [13] that advertises a Pilot and has instructions for enrolling in the pilot. At this time, the wallet can be used for:

- In-person identification at security checkpoints in the San Francisco and Los Angeles airports.
- In-person age verification at participating businesses.
- Online login at the CA DMV web site, by scanning a QR code presented by the relying party.

Projects are underway for other use cases, including identification of banking customers to meet Know Your Customer (KYC) requirements.

12.3.4 The EUDI Wallet Architecture and Reference Framework (ARF)

In June 2021 the European Commission adopted a recommendation calling on member states to work towards the development of a technical Architecture and Reference Framework (ARF) for a European Digital Identity (EUDI) wallet, in the context of the electronic IDentification, Authentication and trust Services (eIDAS) regulation [14]. The eIDAS Expert Group developed and published the ARF, which is available online at [15].

The EUDI wallet is intended to carry three different kinds of data: Person Identification Data (PID), Electronic Attestation of Attributes (EAA) and Qualified Electronic Attestation of Attributes (QEAA) [16]. The ARF defines *PID* as "A set of data enabling the identity of a natural or legal person to be established, and *EAAs* as "An attestation in electronic form that allows the authentication of attributes".

Section 5.1.1.1 of the ARF lists "PID attributes for natural persons", including mandatory eIDAS attributes (Current Family Name and First Names, Date of Birth, and "Unique Identifier"), optional eIDAS attributes (Family Name at Birth, First Names at Birth, Place of Birth, Current Address, Gender), and other optional attributes, including a multi-valued Nationality/Citizenship attribute.

The ARF requires the wallet to support four flows:

1. Proximity supervised flow, where the relying party is controlled by human attendant.
2. Proximity unsupervised flow.
3. Remote cross-device flow.
4. Remote cross-device flow.

and considers four scenarios for each of the proximity flows:

a. The User and the Relying Party are both online.
b. Only the User is online.
c. Only the Relying Party is online.
d. Both the User and the Relying Party are offline.

The wallet may carry Verifiable Credentials and/or ISO/IEC 18013-5 mDL credentials, and must support both in Type 1 (high level of assurance) configurations.

Credential presentation in remote flows uses OpenID4VP for general credentials, or SIOPv2 for self-issued credentials. Credential presentation in proximity flows is as specified in ISO/IEC 18013-5.

12.4 Using a web browser as a credential wallet

Browsers have stored SSL certificates, now called TLS certificates, since the very beginning of the Web. But a technique for storing general purpose credentials did not emerge until a presentation at the 2017 International Cryptographic Module Conference [17] [18], after the technique was made possible by the introduction of the *Service Worker API*, which made it possible to use a service worker for credential presentation as explained below in Section 12.4.1.

This technique is particularly timely and relevant today because it makes it possible to login to a web site from a laptop using an mDL credential carried in a mobile device, something that is not supported by ISO/IEC 18013-5 or ISO/IEC 18013-7. The protocol that makes this possible is described below in Section 12.4.3.4.

The 2017 presentation discussed two security problems with the technique. Those problems have now been solved as explained below in Section 12.4.2.

Using a web browser as a credential wallet has compelling deployability advantages: every computer user has a browser ready to be used as a wallet, and relying parties do not have to figure out how to reach that wallet. It also provides important UX benefits, which are discussed in Chapter 17.

12.4.1 Repurposing a service worker for credential presentation

As we saw in Section 5.2 of Chapter 5, a service worker is a script that is registered with the browser by JavaScript code originating from a particular domain and can be configured to intercept requests that target that domain.

The original purpose of the Service Worker API was to enable Progressive Web Apps, which we described in Section 6.3 of Chapter 6, to be used while the browser is offline. The web origin of the JavaScript frontend of a PWA is the domain of its backend, and thus a service worker registered by the frontend of a PWA can intercept requests that target its backend. If the browser is offline, the frontend can continue to function using data that has been cached by the frontend. To do so, it sends requests for data to the backend that are intercepted by the service worker. When it intercepts a request, the service worker check to see if the backend can be reached. If so, it forwards the request to the backend and provides the data that it receives from the backend in response. If not, it provides instead the data that it has cached.

A service worker can only intercept requests that target its own origin. But the Service Worker API does not impose any restriction on the *origin* of those requests. This makes it possible for a service worker registered by the frontend of a credential issuer to intercept requests for credential presentation that relying parties send to the issuer, and respond to those requests by presenting a credential that it has previously obtained from the backend and stored in *localStorage* as specified by the Web Storage API, or in an indexed database as specified by the IndexedDB API, both described in Chapter 5.

All three web APIs mentioned above, the Service Worker API, the Web Storage API, and the IndexedDB API, are standard web technologies available in all browsers. Thus, any web browser on any device, whether mobile, desktop or laptop, can be used as a credential wallet.

12.4.2 Recent resolution of two security problems

When introduced in 2017, the technique for storing and presenting credentials faced two security problems, discussed in the ICMC presentation:

1. An attacker who physically captured the device where the browser was running could use or exfiltrate the credential.

This problem has been solved because a browser is an app, or can be embedded in a WebView of an app, and apps can be locked when not in use. Locking an app prevents it from being used by an attacker who captures the device and encrypts the app's data. In a Samsung phone, an app can be locked by placing it in a Secure Folder [20]. App locking methods for all Android phones are surveyed in [21]. A similar app locking method has also become available in iOS 18 [22], and third party apps are available for locking apps in MacOS [23] and Windows [24]. Conveniently, there are multiple browsers in a mobile phone, and it is possible to create multiple instances of a given browser. Thus, the user can use different browsers or browser instances for storing different credentials, and lock only those that carry sensitive credentials.

2. Since the web origin of the service worker is the domain of the credential issuer, the credential is stored in browser storage controlled by the issuer. Hence, the issuer has access to the private key of the credential, in cases where the credential comprises a public key certificate and its associated private key. This is discussed in [17] and [18] as if it were a vulnerability, but it is not: the credential issuer does not need to access to the private key to impersonate the subject, since it can just issue a credential to itself. But it makes it impossible to claim that use of the private key is non-repudiable, which is typically a feature of an asymmetric signature.

This problem has been resolved by the realization that, while non-repudiation may be desirable for a credential that is used as a seal on a digital signature applied to a document, it is not desirable for a signature of an authentication challenge. On the contrary, repudiability is now considered a desirable privacy feature for an authentication method. For example, ISO/IEC 18013-5 has introduced the concept of an *ECDH-agreed MAC* [25, §9.1.3.3] for the purpose of providing repudiable authentication using an asymmetric key pair. Thus, the fact that the issuer has access to the private key of credentials stored in a browser is now viewed as a privacy feature.

12.4.3 Protocols and use cases

The ICMC presentation only provided protocols for credential issuance and credential presentation to a relying party accessed from the same browser where the credential is stored. In this section we describe those protocols in more detail and provide protocols for several other use cases, including, in Section 12.4.3.4, the above-mentioned use case of cross-device presentation of an mDL.

12.4.3.1 Credential issuance to a browser

Figures 1a-1e show the steps for issuing a credential to a browser, storing it in browser storage, and registering a service worker, essentially as presented at ICMC 2017. The browser storage shown in the figures may be the *localStorage* facility of the Web Storage API, or an indexed database as provided by the IndexedDB API.

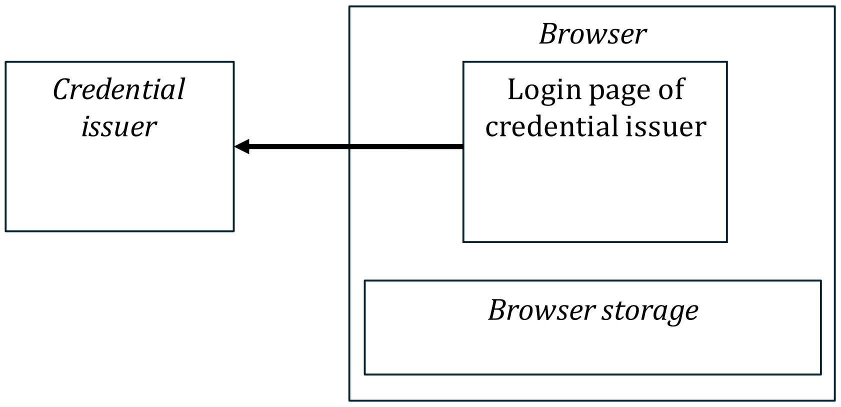

Figure 1a: User logs in to the credential issuer's web site

In Figure 1a the user visits the login page of the web site of the credential issuer and submits a login request. We assume that the user has a preexisting account with the issuer and has provided evidence of being entitled to each of the attributes that will be included in the credential, which the issuer has stored in the account. We assume that the user logs in successfully using a preestablished method for returning-user authentication. As a result of the successful login, the web site of the issuer sets a login session cookie in the user's browser.

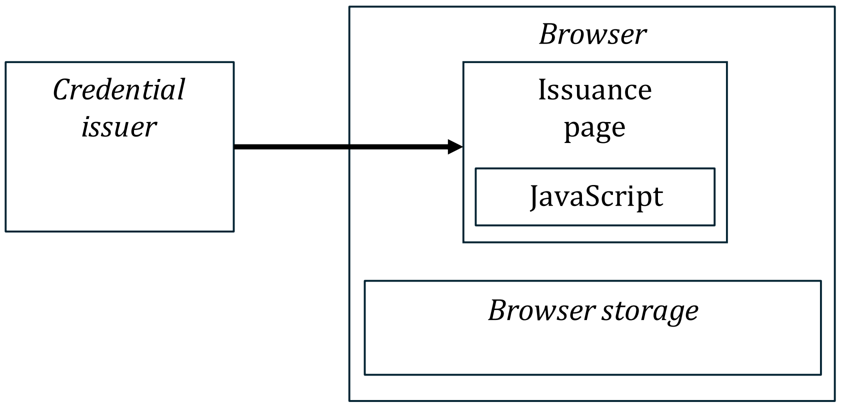

Figure 1b: Credential issuer downloads issuance page with JavaScript

In Figure 1b the user navigates to the issuance page of the credential issuer's site and the credential issuer downloads a web page with embedded JavaScript code.

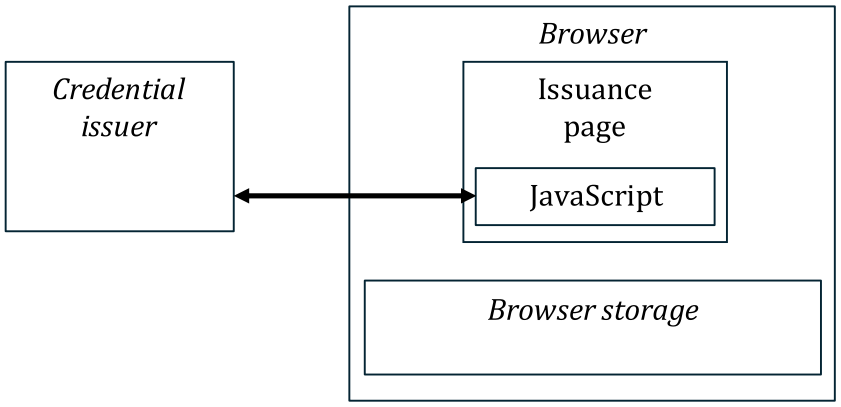

Figure 1c: Issuer's JavaScript frontend executes issuance protocol with backend

In Figure 1c, the user requests issuance of a particular credential with particular attributes, and the JavaScript code embedded in the issuance page, which is part of the frontend of the credential issuer, executes a credential issuance protocol with the backend. The user is authenticated by the login session cookie and the issuer obtains the user's attributes from the account that is referenced by the session cookie.

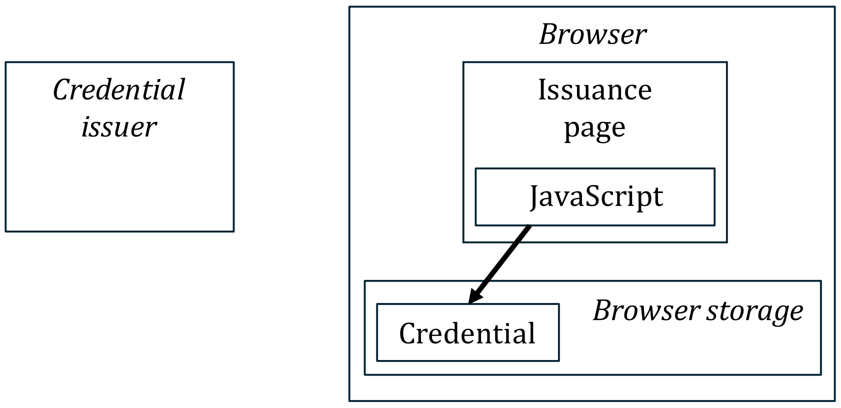

Figure 1d: Issuer's JavaScript frontend stores the credential

In Figure 1d the JavaScript frontend code in the Issuance page stores the credential resulting from the issuance protocol in the browser storage. If the user had requested a public key certificate, the credential stored in browser storage comprises the public key certificate received from the backend and the private key component of the key pair generated by the JavaScript frontend.

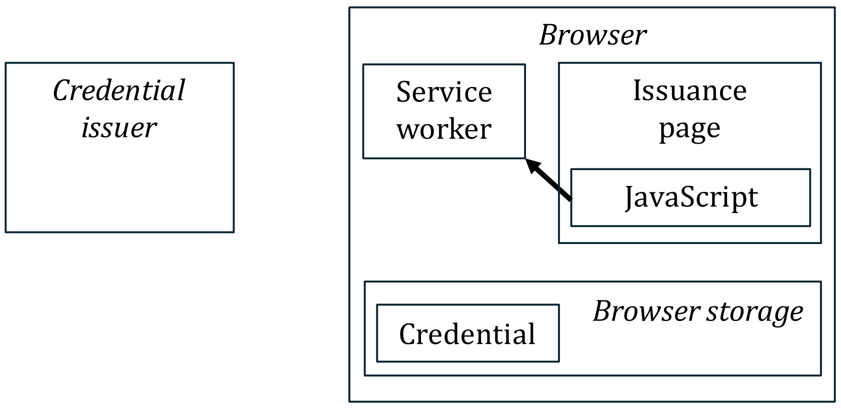

Figure 1e: Issuer's JavaScript frontend registers service worker with browser

In Figure 1e, the JavaScript frontend code in the Issuance page registers a service worker with the browser and configures it intercept credential presentation requests addressed to the credential issuer.

12.4.3.2 Same-browser (a.k.a. same-device) credential presentation

Figures 2a-2f show the steps of the protocol for presenting a credential stored in a browser to a relying party accessed from the same browser. This is essentially the protocol that was described at ICMC 2017.

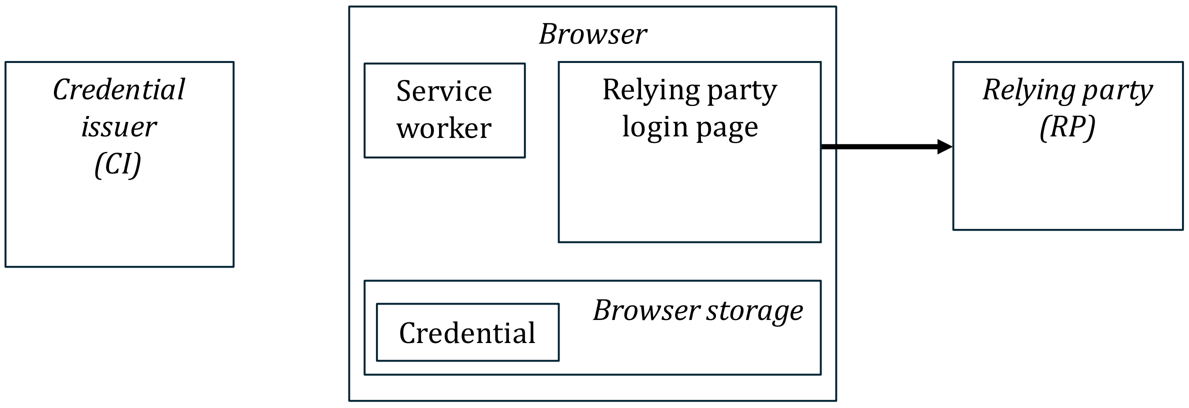

Figure 2a: User visits RP login page and submits login request

In Figure 2a the user visits the login page of the relying party and submits a login request (or performs some other action that requires authentication), which causes an HTTPS request to be sent by the browser to the relying party.

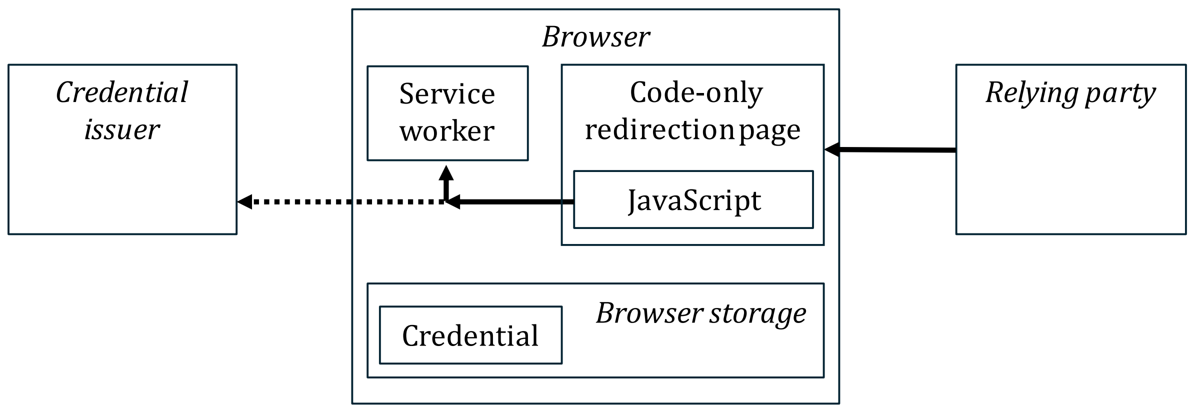

Figure 2b: JavaScript redirection to the issuer is intercepted by the service worker

In Figure 2b, the relying party determines the credential that it needs to authenticate the user and responds to the HTTPS request from the user's browser with an HTTPS response that performs a JavaScript redirection, by downloading a code-only redirection page with JavaScript code that sends an HTTPS request to the issuer, which is intercepted by the service worker. The intercepted request specifies the required credential, and the required attributes if the credential allows selective disclosure, and contains a callback URL and an authentication challenge. (It may also contain values such as a timestamp or validity period.)

In some use cases the relying party has a *relying party designation certificate (RPDC)*, signed by a *relying party designation authority (RPDA)* known to the service worker, that binds the domain name of the relying party to a *user-friendly certified designation (UFCD)* of the party. In those use cases the RP includes its UFCD along with the callback URL.

The figure assumes that there is a service worker in the browser configured to intercept requests that target the credential issuer. If there isn't one because the user has not installed a credential in this browser, or the service worker and the credential have been deleted, the credential request will reach the issuer. The issuer will be prepared to handle this situation and will do so by asking the user to authenticate with the same authentication method that the user used when requesting the credential. When the user does so, the issuer will issue a new credential and install a new service worker on the fly, and the new service worker will present the new credential to the RP. The RP will not be aware that the original credential was not available.

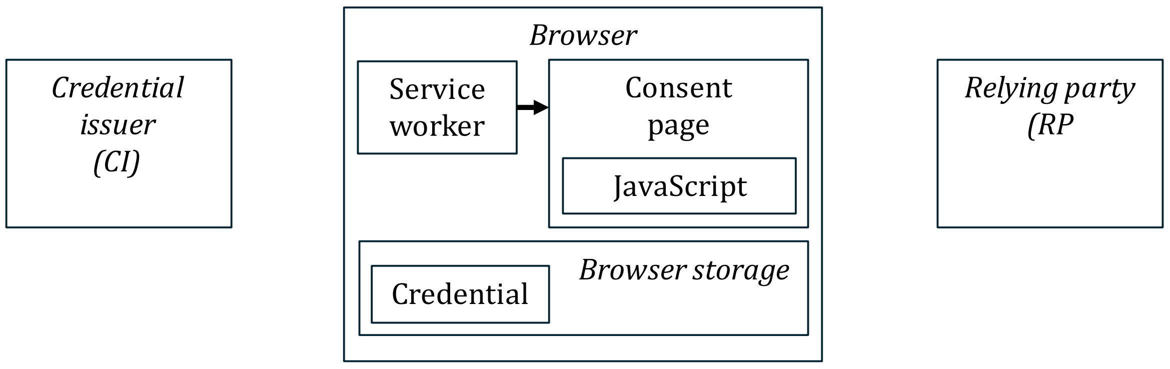

Figure 2c: Service worker generates consent page

In Figure 2c, after intercepting the request, the service worker creates on the fly a web page with embedded JavaScript code that is rendered by the browser as if it came from the issuer and will be used to ask the user for consent to present the credential and the required attributes. Data in the JavaScript code includes the callback URL, the authentication challenge, and a list of requested attributes. If the RP has sent an RPDC along with the callback URL, the service worker uses the public key of the RPDA to verify the binding between the domain name in the callback URL and the UFCD and provides the UFCD to the JavaScript code that will ask consent.

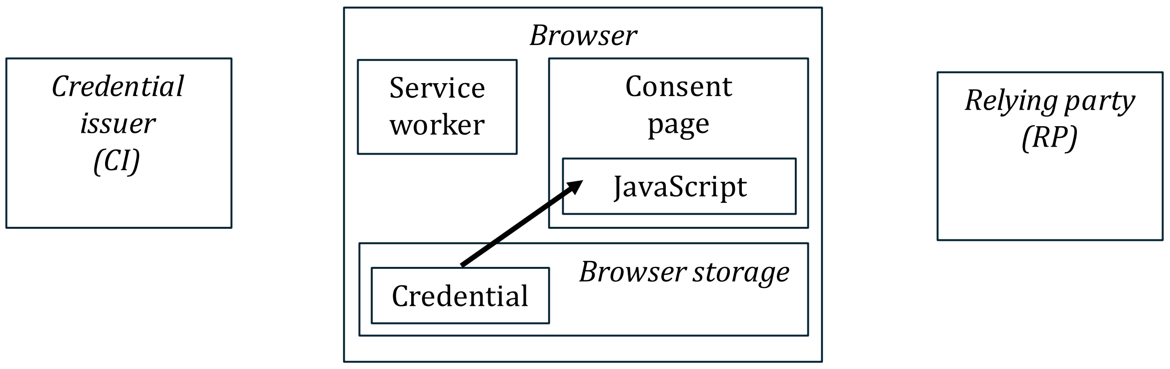

Figure 2d: JavaScript retrieves credential from storage and obtains consent

In Figure 2d, the JavaScript code in the consent page retrieves the credential from browser storage.

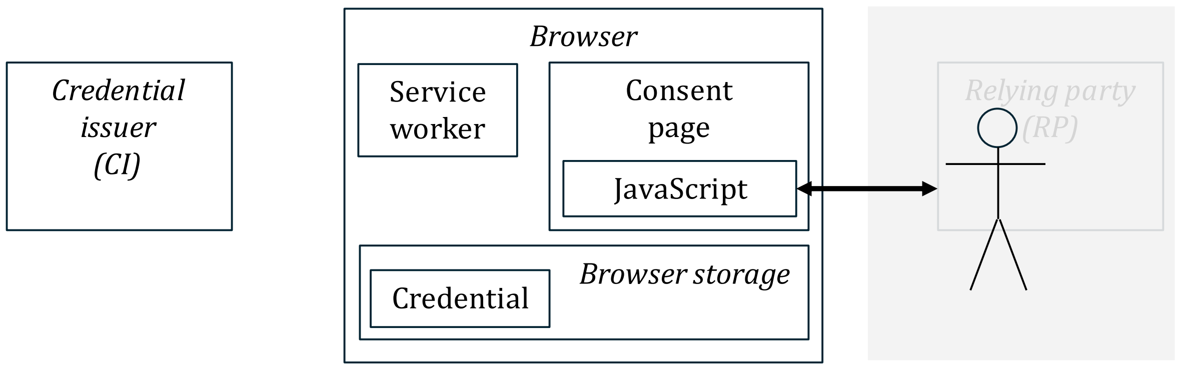

Figure 2e: User is asked for consent to present the credential and required attributes

In Figure 2e, the JavaScript code in the consent page asks the user for consent to provide the credential with the required attributes to the relying party. If the service worker has provided a UFCD, it is used to identify the RP to the user in the consent request. If not, the RP is implicitly or explicitly identified as the web site whose login page has been replaced with the consent page. (The intermediate redirection page is a code-only page that is not rendered by the browser.)

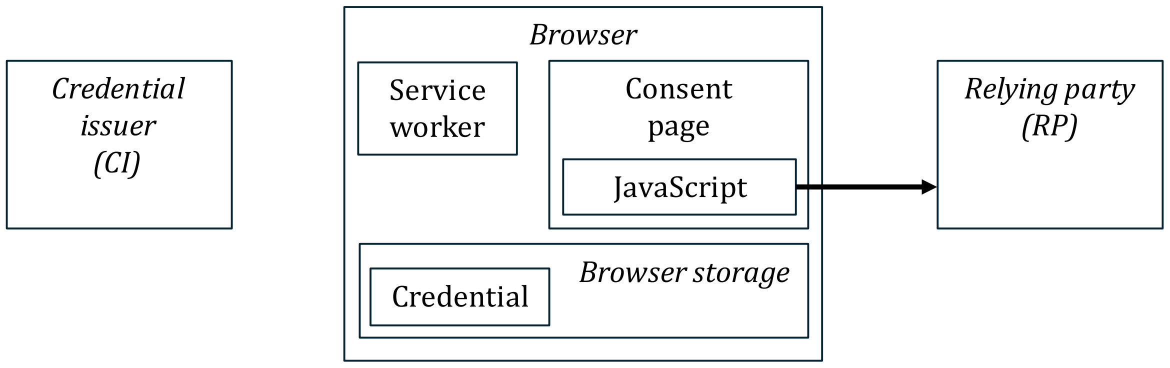

Figure 2f: JavaScript presents credential to relying party

In Figure 2f, the JavaScript code in the consent page presents the credential to the relying party by sending an HTTPS POST request to the callback URL, and the relying party verifies the presentation.

If the credential consists of a public key certificate and its associated private key as described in Section 3.2 of Chapter 3, the body of the POST request contains the certificate and a signature computed with the private key on the authentication challenge, the callback URL, and other data such as a timestamp or validity period if included in the credential request; including the callback URL in the signed data provides protection against man-in-the-middle attacks as explained in Chapter 4. The relying party verifies the issuer's signature in the certificate using the issuer's public key and the signature in the POST request using the public key in the certificate.

If the credential comprises a selective disclosure public key certificate where the issuer's signature is applied to a redactable digest of the subject's attributes as described in Section 3.3 of Chapter 3, the body of the POST request contains the certificate, the disclosed attributes, and a presentation signature computed with the associated private key on the authentication challenge, the disclosed attributes, the callback URL, and any other data included in the request. In addition to verifying the signature in the certificate and the signature in the POST request as in the case of an ordinary public key certificate, the RP verifies that the set of disclosed attributes is an antecedent of the redactable digest. In the particular case where the selective disclosure certificate is an SD-JWT, the presentation signature is included in a Key Binding JWT tied to the SD-JWT [32, §3.3].

If the credential is an anynymous credential such as a BBS signature, the body of the POST request includes the NIZK proof and the disclosed attributes. In this case there is no private key to sign the authentication challenge, the disclosed attributes, the callback URL and any other data included in the request. Instead, those data items are included in the Fiat-Shamir payload as explained in Section 3.5.6.3 of Chapter 3.

12.4.3.3 Cross-device credential presentation

In this section we describe a protocol for presentation of a credential stored in a mobile browser to a relying party accessed from a browser running on a laptop or desktop, where the relying party displays a QR code that encodes a URL of a credential presentation request.

Figures 3a-3g show the steps of the protocol.

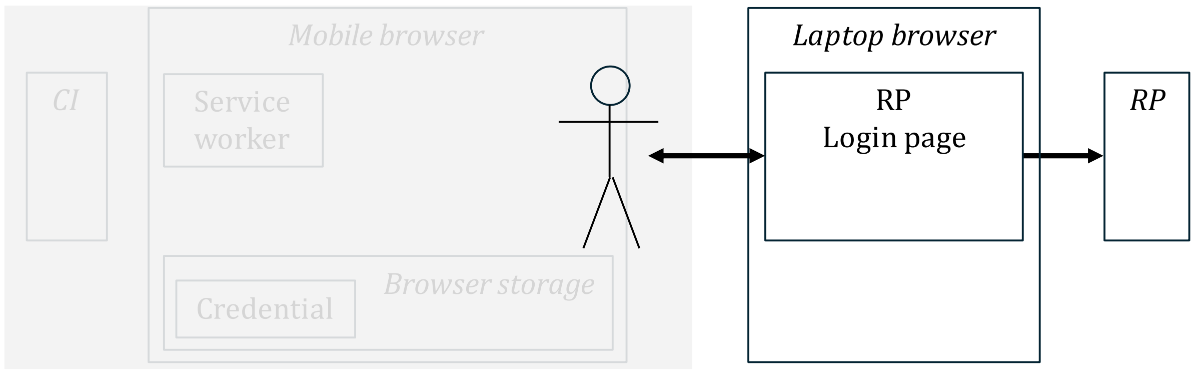

Figure 3a: User chooses option to login with mobile wallet

In Figure 3a, the user visits the login page of the relying party and chooses an option to log in with a mobile wallet.

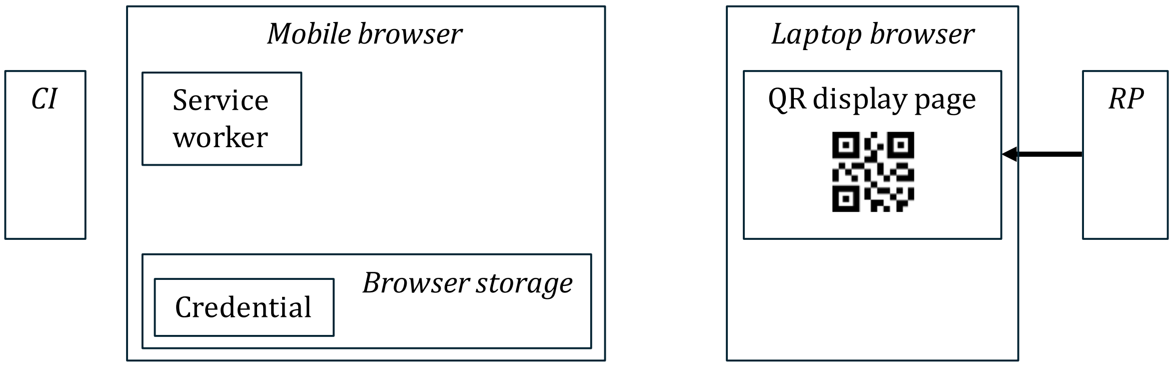

Figure 3b: RP displays QR-encoded credential request addressed to issuer

In Figure 3b, the relying party determines the credential that it needs and displays a QR code on the screen of the laptop that encodes a URL that targets the issuer of that credential. The query string of the URL specifies the credential that is being requested and, if the credential supports selective disclosure, the attributes that the RP is requesting. The query string also includes a callback URL and an authentication challenge.

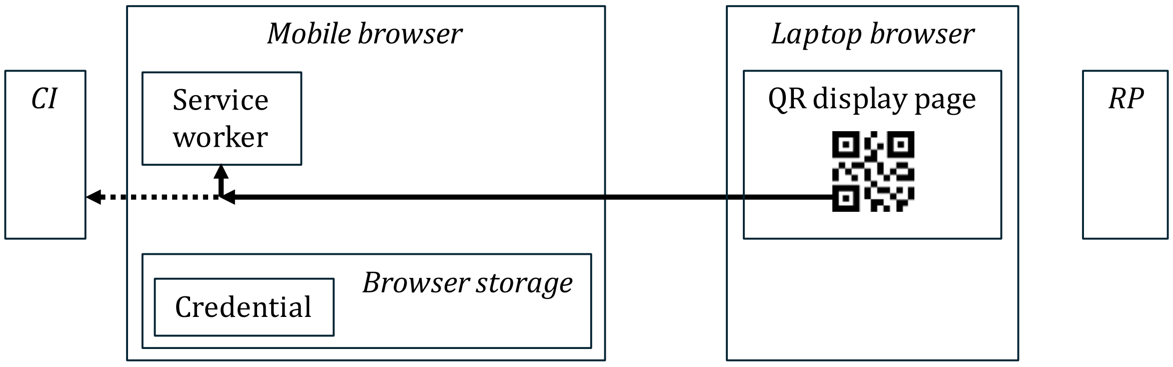

Figure 3c: Credential request to issuer intercepted by service worker

In Figure 3c the user scans the QR code with the mobile device containing the browser used as a credential wallet. The user may have configured that browser as the default browser of the mobile device. In that case the request for credential presentation encoded in the QR code is sent automatically from that browser. Alternatively, the user may have arranged for the device to have no default browser. If so, the user is prompted to choose a browser and chooses the one that contains the credential that the RP needs, so that the request is sent from that browser. In either case, the credential presentation request is sent from the browser containing the credential and intercept by the service worker in that browser.

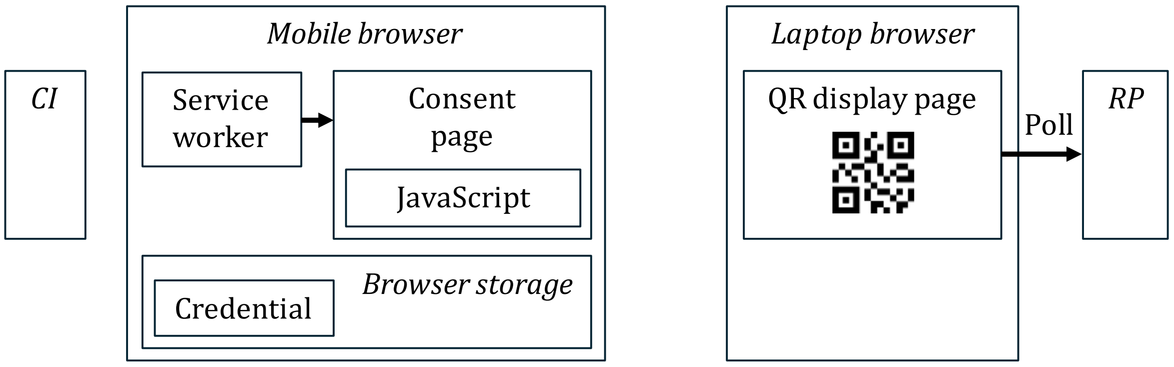

Figure 3d: Service worker generates consent page while laptop polls RP

In Figure 3d, the service worker in the mobile browser generates a consent page with JavaScript code as in Figure 2c, while the QR code is still displayed on the laptop browser and JavaScript code in the page that displays the QR code polls the relying party, waiting for the RP to have received the requested credential.

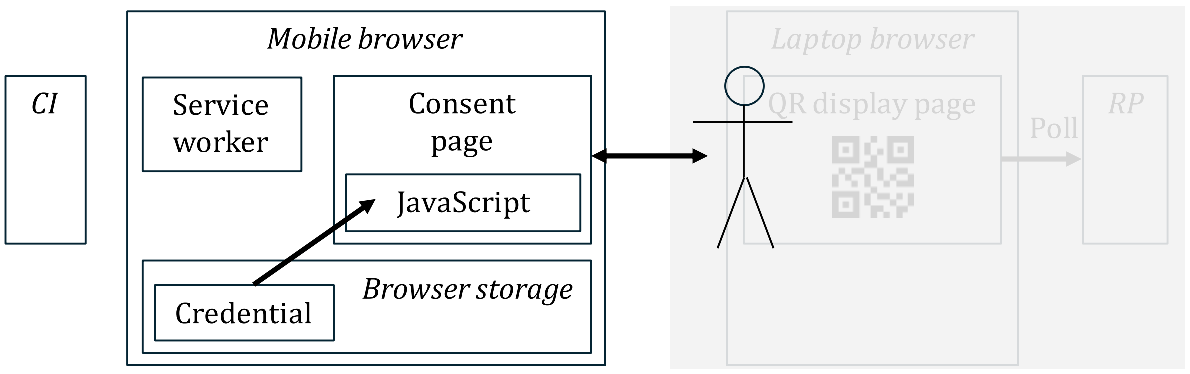

Figure 3e: JavaScript retrieves credential and obtains consent

In Figure 3e the JavaScript code in the consent page retrieves the credential from browser storage and obtains consent to present it.

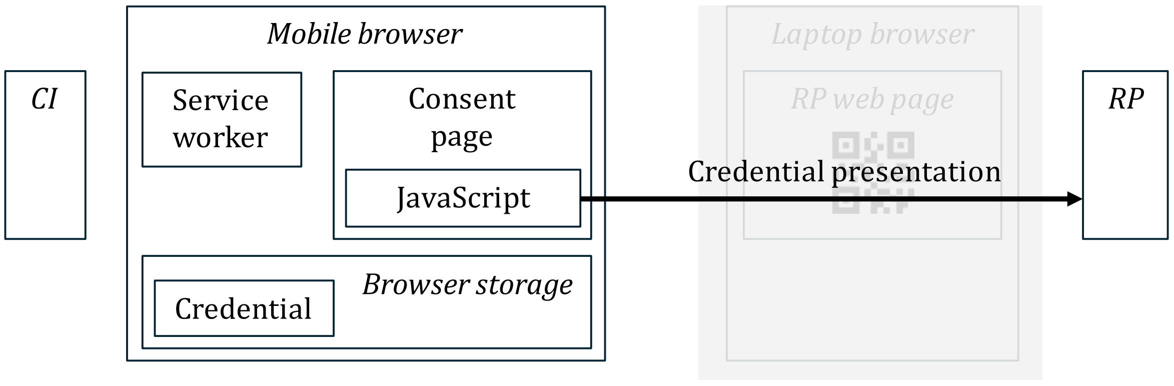

Figure 3f: JavaScript and RP execute credential presentation protocol

In Figure 3f the JavaScript code in the consent page presents the credential to the relying party as in Figure 2f by sending an HTTPS POST request to the callback URL and the relying party verifies the presentation as described in connection with Figure 2f.

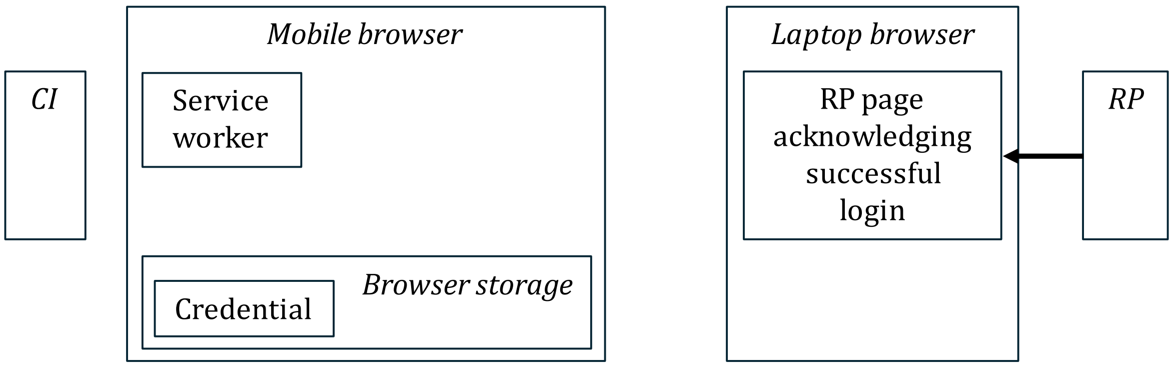

Figure 3g: RP replaces QR code page with login successful page

In Figure 3g the relying party replaces the page that was displaying the QR code with a page acknowledging successful login.

12.4.3.4 Credential presentation to an IoT device controller

In this section we describe a protocol for using a credential stored in a mobile browser to actuate an IoT device by scanning a static QR code displayed on or near the device and presenting the credential to the controller of the device. An important use case is opening a door providing entrance to a facility.

Figures 4a-4e show the steps of the protocol.

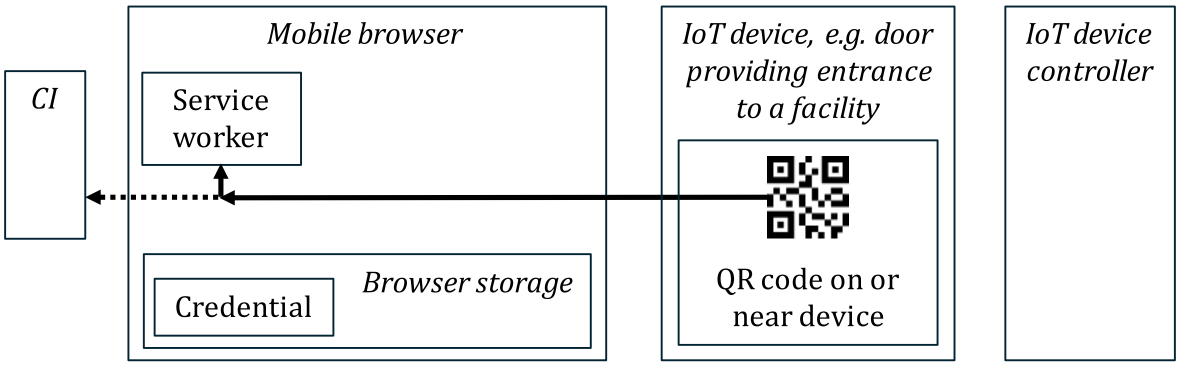
CI
Mobile browser
Service worker
Browser storage
Credential
IoT device, e.g. door providing entrance to a facility
QR code on or near device
IoT device controller

Figure 4a: Mobile device scans QR code of IoT device

In Figure 4a, the mobile device scans a QR code located on of near the IoT device that encodes a URL that targets the issuer of a credential acceptable by the controller of the IoT device. The URL requests presentation of the credential and includes a callback URL that targets the controller of the device and specifies the device to be actuated.

As in Figure 3c, the operating system of the mobile device causes a mobile browser to send a request to the URL, but the request is intercepted by a service worker registered by the credential issuer.

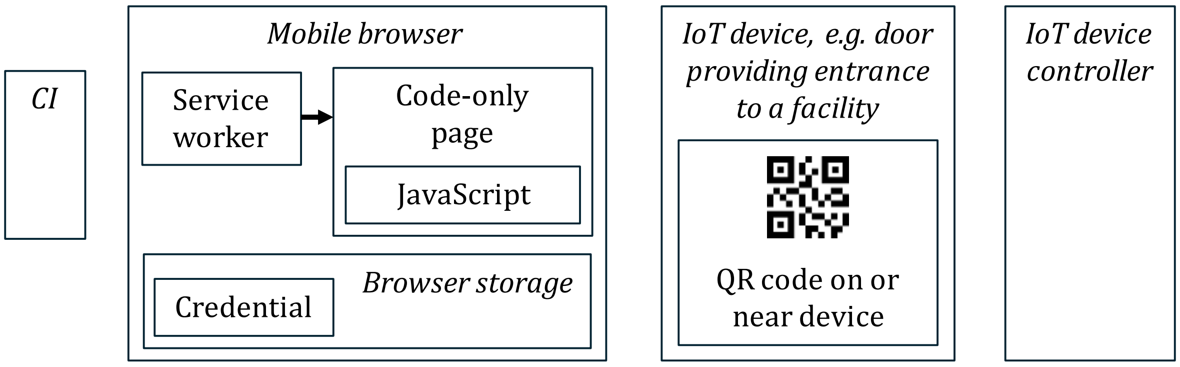

Figure 4b: Service worker generates code-only page with JavaScript code

In Figure 4b the service worker generates a code-only age containing JavaScript code.

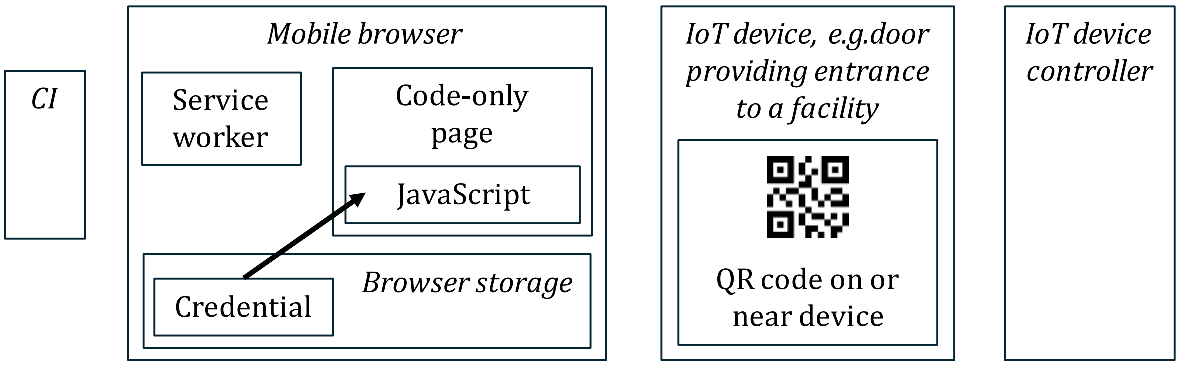

Figure 4c: JavaScript code retrieves credential

In Figure 4c, the JavaScript code in the code-only page retrieves the requested credential from browser storage.

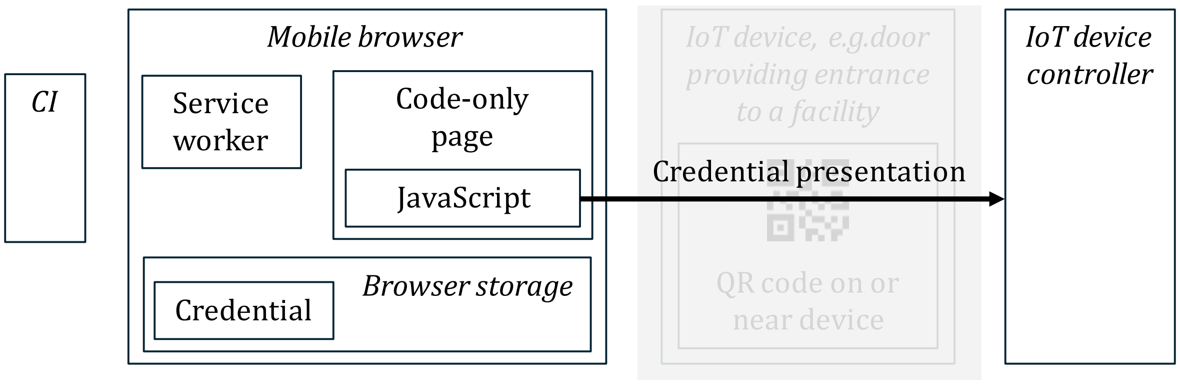

Figure 4d: JavaScript code presents credential to IoT device controller

In Figure 4d, the JavaScript code in the code-only page sends a request to the callback URL. The request targets the IoT device controller, specifies the device to be actuated, and performs a credential presentation protocol with the controller. If the protocol is a challenge-response protocol, the challenge will not have been sent earlier, since the QR code is static. Hence the presentation is interactive, with the controller sending the challenge first, then the JavaScript code sending the response.

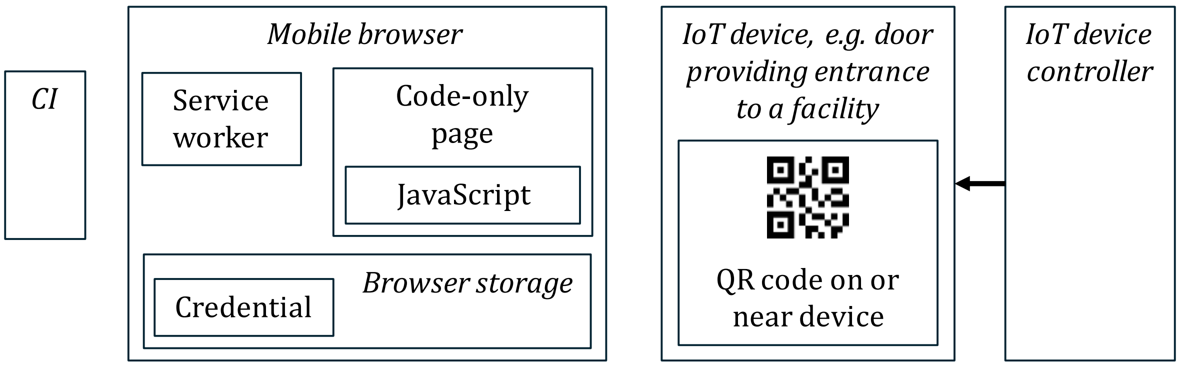

Figure 4e: IoT controller actuates device, e.g. opening the door to the facility

In Figure 4e, the IoT controller actuates the IoT device, e.g. opening the door to the facility.

12.4.3.5 Using a WebView as a wallet for proximity presentation

In the same-device, cross-device and IoT presentation protocols of Sections 12.4.3.2-4, the credential is presented by JavaScript code running in the browser. One thing that JavaScript code running in the browser cannot do, as of this writing, is to present a credential over proximity protocols such as BLE, NFC or Wi-Fi Aware. That requires native code running in a native app.

Proximity presentation, however, can be provided by embedding the browser in a native app as a WebView component and pairing it with a native code component that has the required capability. A WebView is a browser configured for use as a component of a native app. A WebView based on Chromium is available for Android apps [29] and one based on WebKit and known as WKWebView is available for iOS apps [30].

The native code and the JavaScript code can communicate and exchange data, so they can both have access to the same credential, whether the credential is stored in the native component, or the WebView component, or cloned and stored in both components as described below in Section 12.4.3.6.1; and the native code can present it using a proximity protocol specific to the credential while the WebView component uses one of the protocols described above in Sections 12.4.3.2-4 for presentation over the web. We shall refer to a native app having a WebView component for presentations over the web and one or more native components for proximity presentations as a *WebView wallet.*

Once WebView wallets become available, there will be no need for government wallets to present government credentials. We envision that the user will create a WebView wallet by downloading a native app with a WebView component from an app store, then downloading native components as needed with code for proximity presentation of credentials such as an mDL and one or more eIDs, besides credentials providing physical access to government or enterprise facilities. The credentials will not come from the app store, they will be issued by governments over the web to the WebView component. They will then be presented over the web by the WebView component and copied to the native components from proximity presentation.

To make all this more concrete, we look in more detail at the mDL use case in the next section.

12.4.3.6 The mDL use case

The mobile driver's license standard is specified in two documents, ISO/IEC 18013-5 [25], published in September 2021, and ISO/IEC 18013-7 [19], published in October 2024. Both documents are described in Chapter 13.

Part 5 specifies the data model, the mobile driver's license credential, and a protocol for credential presentation by the mobile app using the proximity protocols BLE, NFC or Wi-Fi Aware. There is also an option to download data elements from the mDL issuer over the web, but it is involved in some of the vulnerabilities that we describe in Section 13.1.9.1 of Chapter 13.

Section 3 of Part 5, "Terms and definitions", defines the term "mdoc" in an explicitly ambiguous manner as referring to a "document or application that resides on a mobile device", then defines "mdoc reader" as a "device that can retrieve mdoc (3.2) data for verification purposes". Here we shall avoid the term "mdoc" and use the term *mDL app*, to refer to the native application and the term *mDL reader*, or just *reader,* to refer to the relying party in a proximity presentation. The term "document" is not defined in the glossary but is used sometimes to refer to the cryptographic mDL credential, sometimes to the data elements in the credential, and sometimes to a subset of those data elements, with data elements from multiple "documents" being disclosed to the reader in the same presentation.

Part 7 specifies two methods for remote credential presentation to a relying party over the web. One of the methods, which we shall call the ISO method, is specified in the ISO document itself, while the other one is a forthcoming profile of the OpenID4VP specification that is currently under development. OpenID4VP is discussed in Chapter 5.

Both methods for remote presentation are applicable only to the case where the relying party is accessed from a browser running on the same device where the mDL credential is stored and the mDL app is running. There is no option for "cross-device" presentation, where the user accesses the relying party from a desktop or laptop, and the RP displays a QR code that is scanned by the mobile device where the mDL app is running. Note 1 of Section B.1.1 of Part 7 states that "Cross-device flows are prone to engagement relay attacks which is the reason cross-device flow is not included in this document."

The proximity presentation protocol specified in Part 5 is unusual. The mDL app and the reader establish an encrypted session and use it to exchange a series of request-response messages. The reader asks for specific data elements in each request by their element identifiers and the app sends their element values in the corresponding response. But instead of establishing an authenticated session and using it to transmit data elements with data origin authentication, the app and the reader use ephemeral ECDH key pairs to establish an unauthenticated section and the app signs each response. Furthermore, a response may contain data elements from multiple documents, and the app separately signs the elements of each of those documents.

The performance impact of multiple signatures and signature verifications may not matter much for proximity presentation if the reader only handles one presentation at a time. But this same authentication method is used by the ISO method of credential presentation over the web, where repeated signature verifications may have a performance impact on the RP server.

In sections 12.4.3.6.1-4 we describe protocols for efficient same-device and cross-device presentation and ISO-compliant proximity presentation of the mDL using a WebView wallet.

12.4.3.6.1 WebView wallet setup

Figures 5a-5e show a process for setting up a WebView wallet that will provide same-device, cross-device and proximity presentation of the mDL.

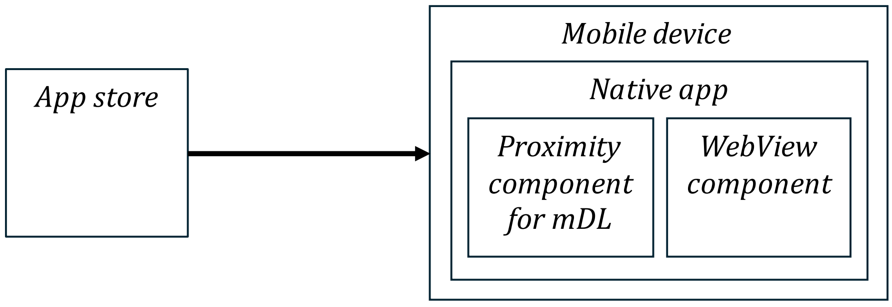

Figure 5a: Installation of app with proximity and WebView components

In Figure 5a, a native app is downloaded from the app store and installed on the user's mobile device. The native app has a WebView component and a proximity component for the mDL, neither of which has any credential. The WebView component has no data. The proximity component has native code for presenting the mDL. The proximity component need not be provided by the mDL issuer; it can be written by any party using a publicly available specification.

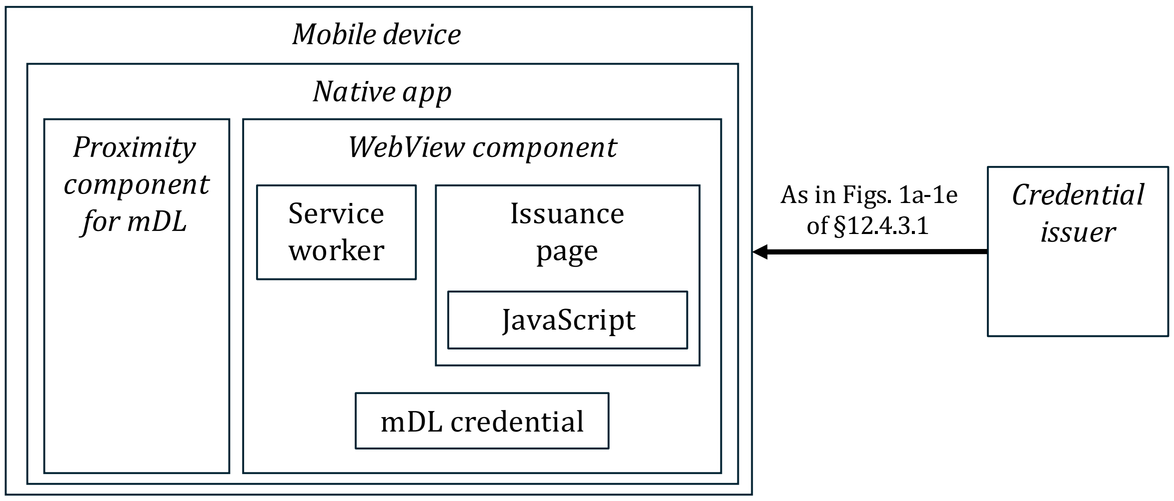

Figure 5b: Issuance of the mDL credential to the WebView component

After the native app is installed with its two components, the credential issuer issues the mDL credential to the WebView component, as described in Section 12.4.3.1. Figure 5b shows the result of the issuance process. The mDL credential is stored in the WebView component, and the JavaScript code in the issuance page has registered the service worker.

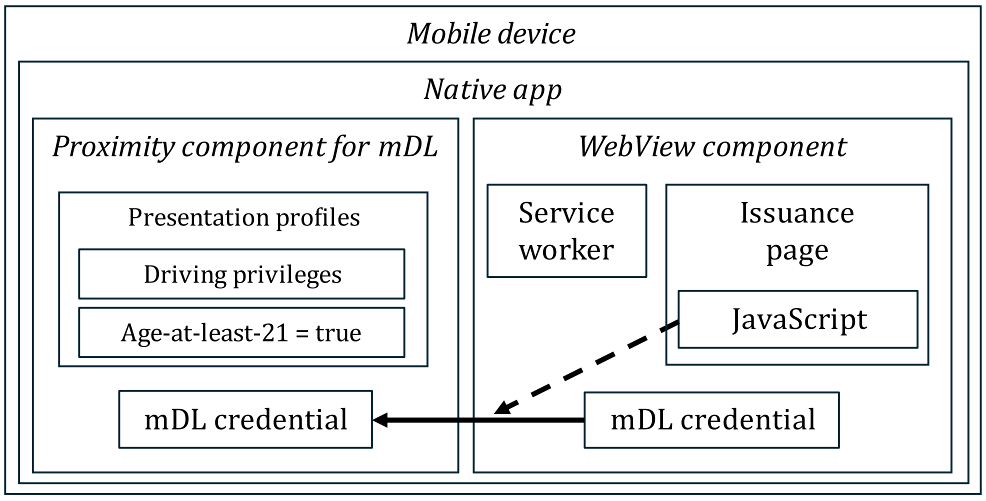

Figure 5c: JavaScript code copies mDL credential to proximity component

After registering the service worker, the JavaScript code in the issuance page copies the mDL credential to the proximity component, as shown in Figure 5c.

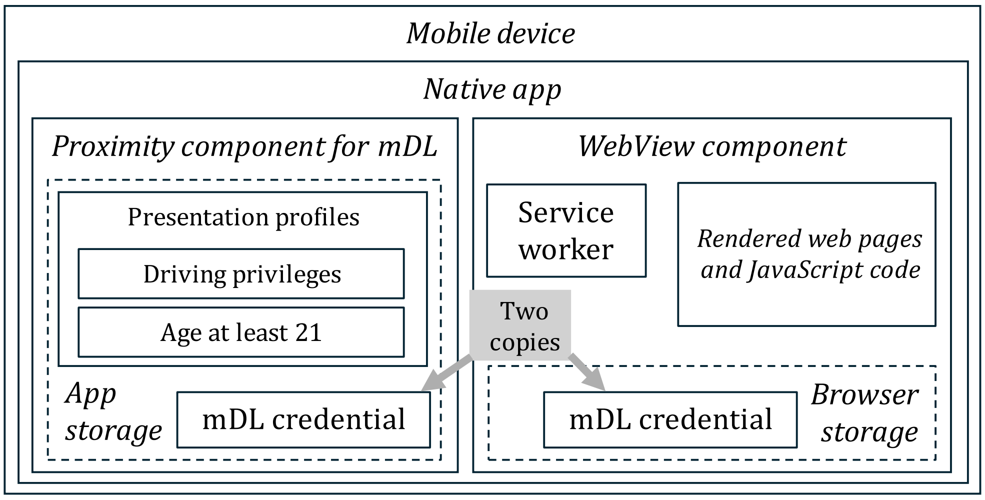

Figure 5d: WebView wallet for multiprotocol presentation of the mDL

Figure 5d shows the state of the WebView wallet at the end of the setup process. There are two copies of the mDL credential, one in each component. The copy in the WebView component is in browser storage, such as the localStorage facility of the Web Storage API. But browser storage is allocated by the WebView component from the native app storage, so there is no essential difference in how the two copies are stored.

An alternative to having a copy in each component would be to keep a copy in one of them and let the code in the component that does not have a copy access the copy in the other component. Accessing the copy in the other component would mean retrieving the public portion of the credential and a proof of possession of the private portion computed by the code of the other component.

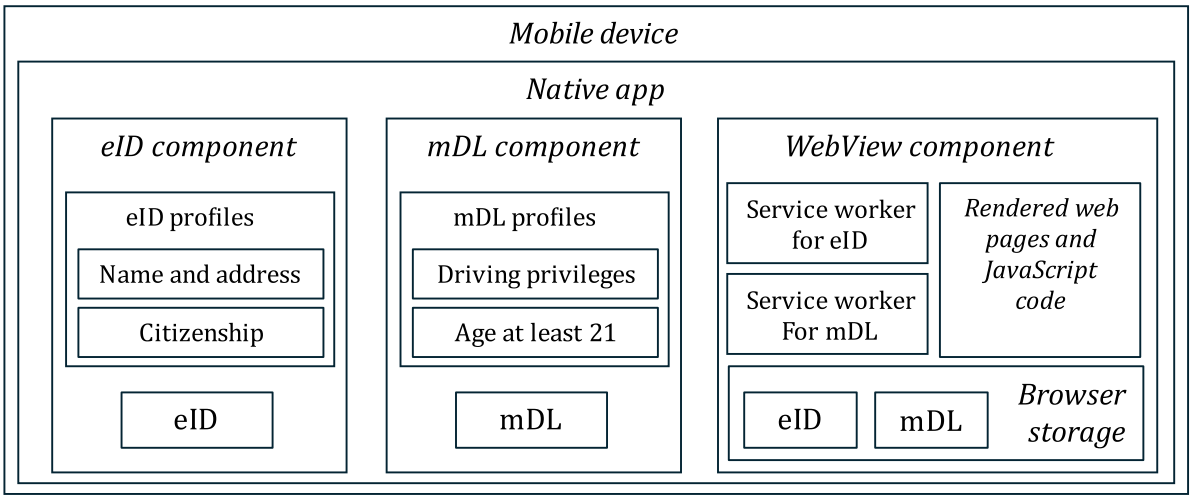

Figure 5e: WebView wallet carrying an mDL and an eID

Figure 5e shows a setup for storing multiple credentials in a WebView wallet and providing same-device, cross-device and proximity presentation for each of them. An eID credential is used as an example. As of this writing, the Wikipedia page entitled

"Electronic identification" [31] tallies 36 countries that use eID's or equivalent credentials. So, a WebView wallet with an mDL credential and one or more eID credentials would provide very useful functionality.

12.4.3.6.2 Efficient same-device presentation

As explained above in Section 12.4.3.6, data retrieval as specified in part 5 of the mDL specification is broken up into a series of request-response exchanges where the mDL reader asks for specific data elements and the mDL provides the values of those elements. In [19, §6.4.3.3], Part 7 specifies the same data retrieval method for credential presentation over the web, which seems inefficient. Appendix B of Part 7 describes a profile of Draft 18 or the OpenID4VP specification as an alternative method.

The idea of retrieving separately several subsets of data elements may have been motivated by the fact that the mDL supports selective disclosure. Obtaining consent for disclosure of specific data elements is out of scope of the mDL specifications, but separate requests for different subsets of elements would allow the subject to select elements by approving some requests but rejecting others. A more practical method of obtaining consent, however, would be to show the subject a list of all the elements that are being requested and ask for a yes-or-no response to an approval request.

Efficient same-device presentation of the mDL over the web with yes-or-no consent can be achieved using the protocol described above in Section 12.4.3.2, which is applicable to credentials comprising an ordinary public key certificate, credentials comprising a selective disclosure certificate, and anonymous credentials. The protocol is applicable to the mDL because an mDL credential comprises a Mobile Security Object (MSO), described in Section 2.2.1.3.4, and the MSO is a selective disclosure certificate according to Section 3.3 of Chapter 3.

Details of the protocol specific to the mDL case are as follows, with reference to the figures of Section12.4.3.2.

In Figure 2b, the required attributes in the intercepted request are specified by element IDs that the RP chooses from the mDL data model but may or may not identify elements included in the mDL credential that has been issued to the user.

In Figure 2d, the credential that the JavaScript code retrieves from browser storage is the entire credential, comprising the private key, the MSO, and the complete set of salted elements whose hashes comprise the array of hashes in the MSO.

In Figure 2e, the JavaScript code in the consent page checks if the element IDs requested by the RP identify elements that can be found in the credential and rejects the request if some of them cannot be found. Then it uses the array of requested element IDs to construct a list of language and culture-specific descriptions of the elements that have been requested and asks the user for permission to disclose them to the relying party.

In Figure 2f, the JavaScript code in the consent page includes the following data in the HTTPS POST request sent to the callback URL:

- The MSO.
- The salted elements found in the credential whose element IDs have been requested.
- A signature computed with the private key on the authentication challenge, the requested salted elements, the callback URL, and any other data included in the request.

The relying party validates the issuer's signature in the MSO using the issuer's public key and the signature in the POST request using the public key in the certificate, and verifies that the set of salted elements is an antecedent of the redactable digest according to Section 2.2.1.3 of Chapter 2.

12.4.3.6.3 Cross-device presentation

Cross-device presentation of the mDL is a special case of the protocol of Section 12.4.3.3. Figures 6a-6g below add details to Figures 3a-3g that are specific to the MSO and to the use of a WebView instead of a standalone browser.

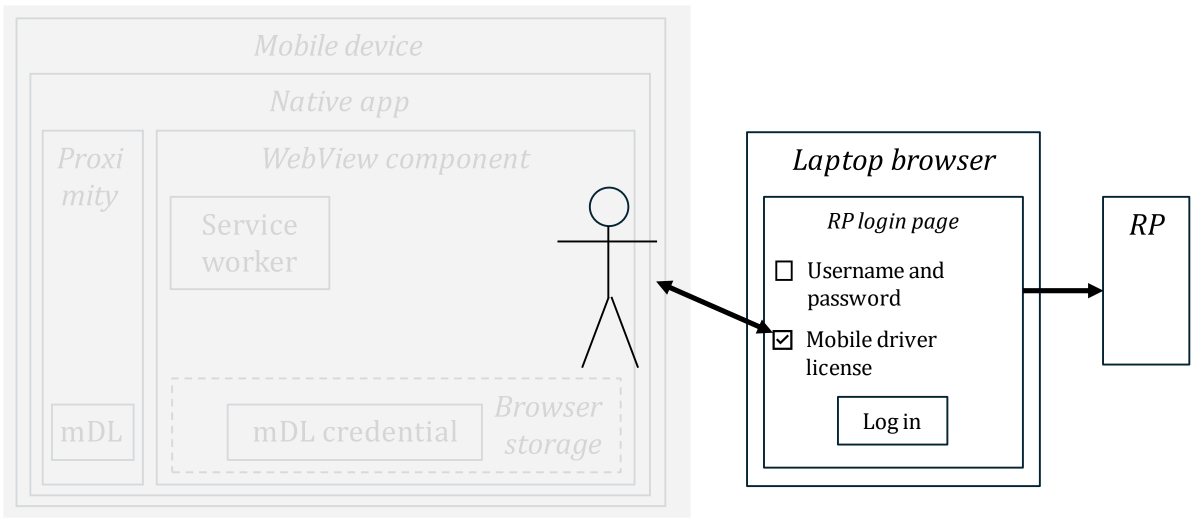

Figure 6a: User chooses option to login with mDL in mobile wallet

In Figure 6a, the user uses a laptop browser to access the login page of the relying party and selects an option to log in using the mobile driver license.

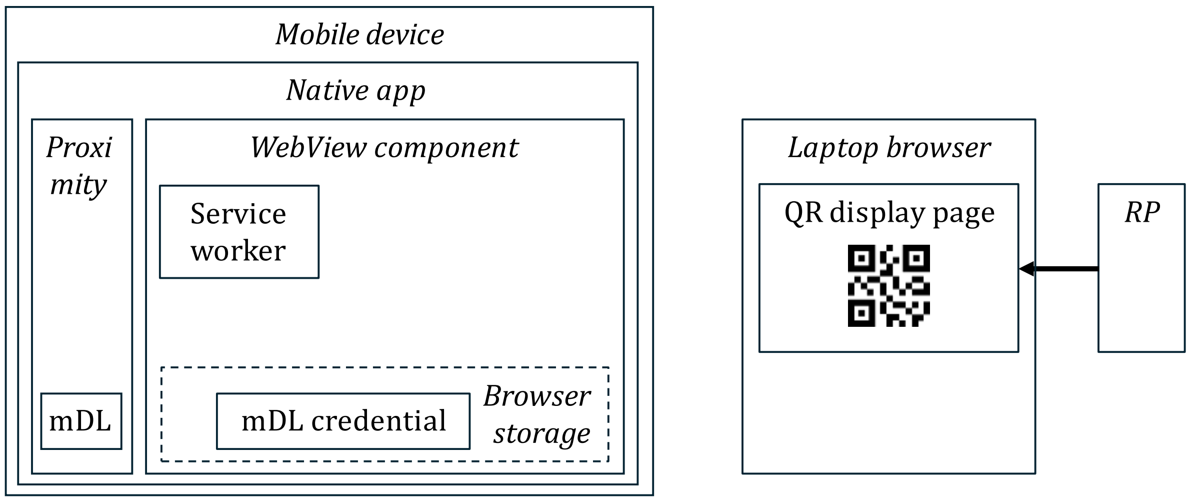

Figure 6b: RP displays QR-encoded credential request addressed to mDL issuer

In Figure 6b, the relying party displays a QR code on the laptop screen encoding a URL that targets the mDL issuer. The query string of the URL specifies the element IDs of the elements of the mDL data model that the RP wants to see, which may or may not be present in the mDL credential that has been issued to the user. The query string also includes an authentication challenge and a callback URL

The user's mobile device includes a native app with a proximity component and a WebView, each having a copy of the mDL credential.

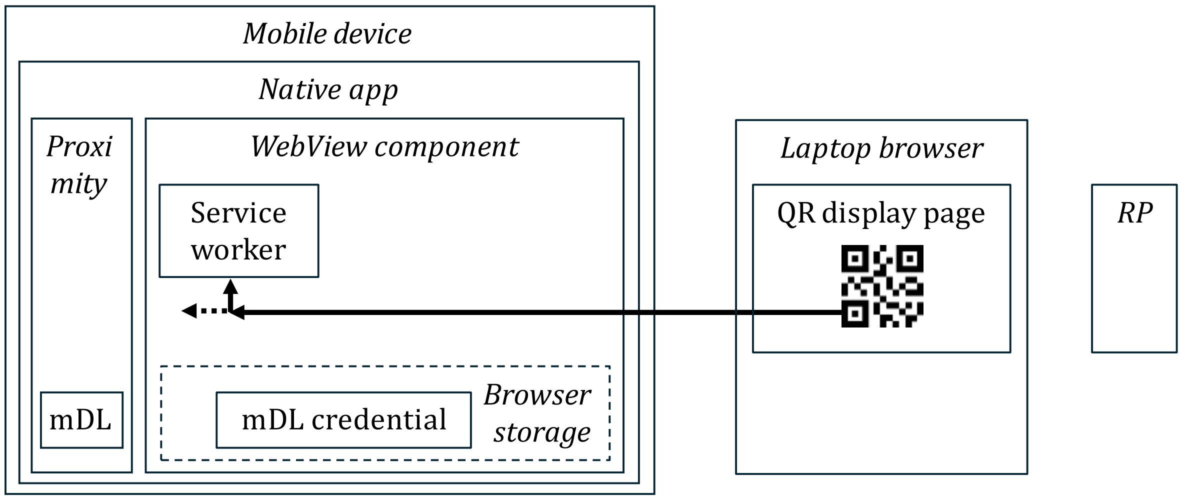

Figure 6c: Credential request to mDL issuer intercepted by service worker

In Figure 6c, the WebView component scans the QR code and sends an HTTPS request to the URL encoded in the QR code. The request targets the mDL issuer but is intercepted by the service worker in the WebView.

A WebView can scan a QR code, as described, for example, in [33].

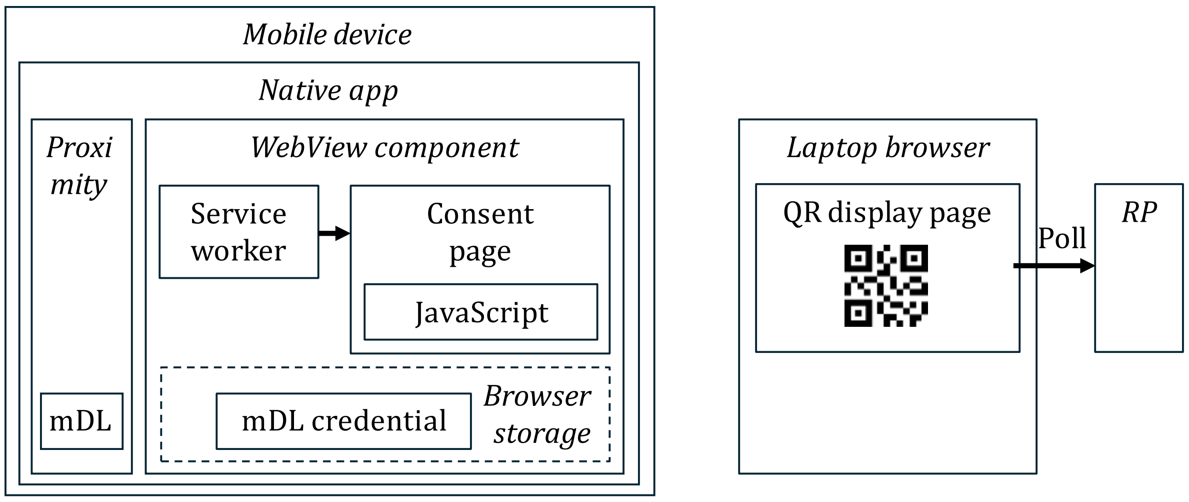

Figure 6d: Service worker generates consent page while laptop polls RP

In Figure 6d, the service worker in the WebView generates a consent page, with data in the JavaScript code of the page that includes the callback URL, the authentication challenge, and the list of element IDs requested by the RP that the service worker has found in the intercepted request.

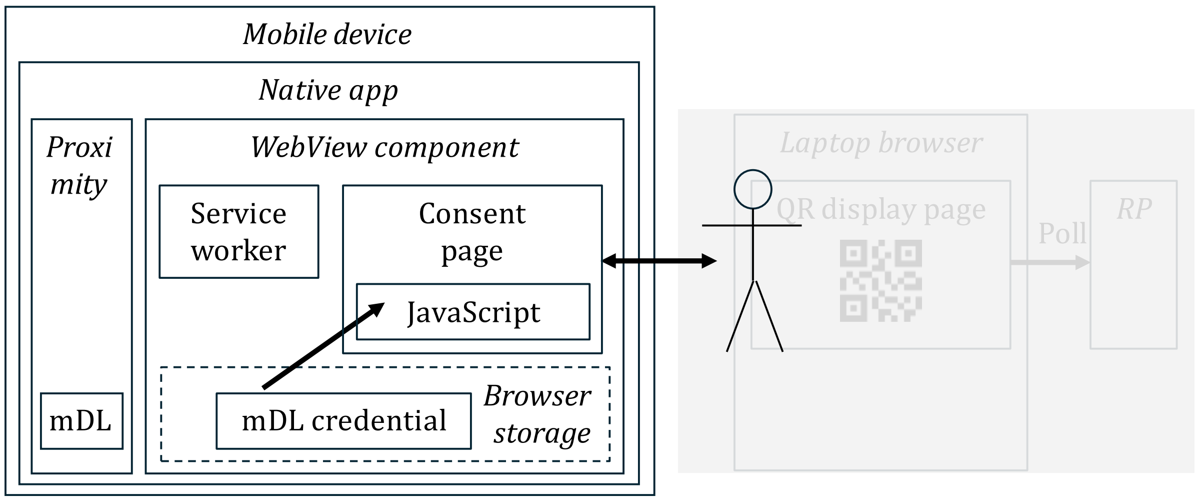

Figure 6e: JavaScript retrieves credential and obtains consent

In Figure 6e the JavaScript code in the consent page does the following:

1. It retrieves from browser storage the entire mDL credential, comprising the private key, the MSO, and the complete set salted elements whose hashes comprise the array of hashes in the MSO. These elements presumably include all the mandatory elements in Table 5 of the Part 5 specification, but probably only a subset of the optional elements.
2. It checks if the element IDs requested by the RP identify elements that can be found in the credential and rejects the request if some of them cannot be found.

3. It uses the array of requested element IDs to construct a list of language and culture-specific descriptions of the elements that have been requested and asks the user for permission to disclose them to the relying party.

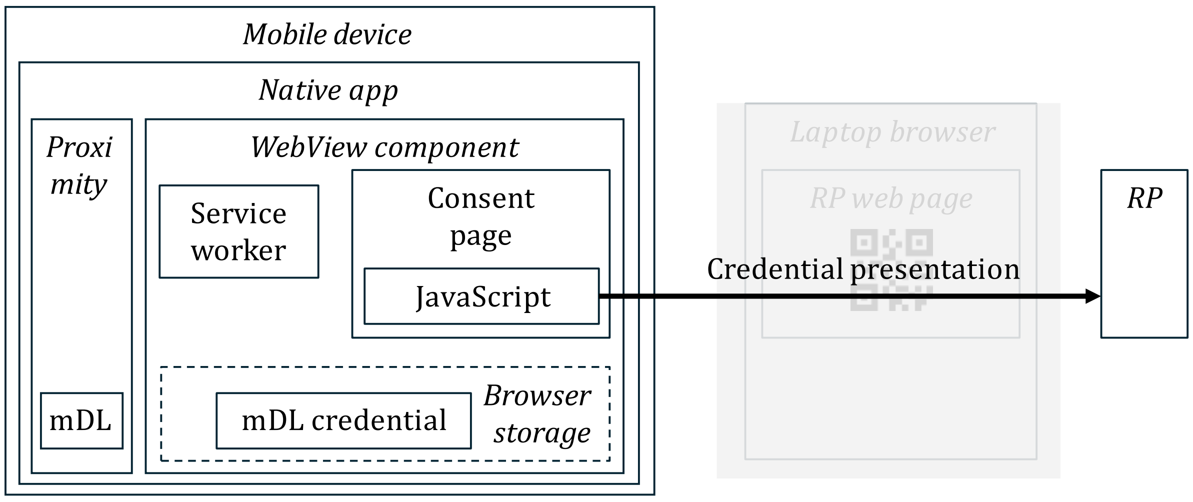

Figure 6f: JavaScript and RP execute credential presentation protocol

In Figure 6f, the JavaScript code in the consent page presents the credential to the relying party by sending an HTTPS POST request to the callback URL, containing:

- The MSO.
- The salted elements found in the credential whose element IDs have been requested.
- A signature computed with the private key on the callback URL, the authentication challenge, and the requested salted elements.

The relying party validates the issuer's signature in the MSO using the issuer's public key and the signature in the POST request using the public key in the certificate, and verifies that the set of salted elements is an antecedent of the redactable digest according to Section 2.2.1.3 of Chapter 2.

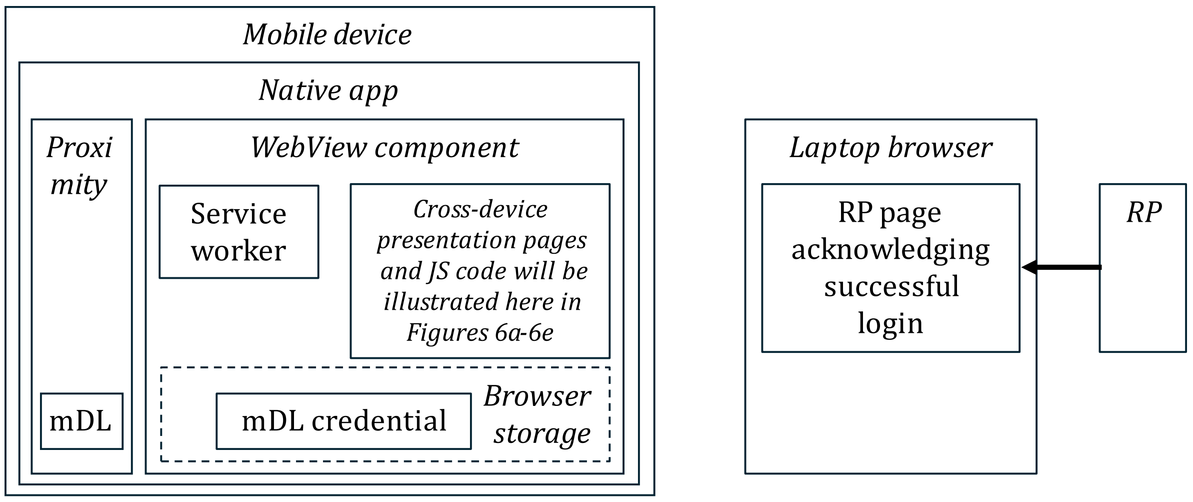
Mobile device
Native app
Proxi mity
mDL
WebView component
Service worker
Cross-device presentation pages and JS code will be illustrated here in Figures 6a-6e
mDL credential
Browser storage
Laptop browser
RP page acknowledging successful login
RP

Figure 6g: RP replaces QR code page with login successful page

In Figure 6g, the relying party replaces the page that is still displaying the QR code in the laptop browser with a page acknowledging that the login has been successful.

12.4.3.6.4 Proximity presentation with mitigation of unauthorized access

As discussed in more detail in Section 13.1.9.2, the mDL is vulnerable to unauthorized access. An important use case of the mDL is to prove that a bar customer is of drinking age. To that purpose, the customer will display a QR code that will be scanned by the bar tender. The QR code will be an encoding of the device engagement structure of the mDL, which will include a server retrieval token. It will be easy for an attacker to surreptitiously take a picture of the QR code, and have it scanned by an ad-hoc attack app that will decode it and extract the token. The attacker will then be able to authenticate to the mDL issuer with the token and retrieve mDL data elements at will, not just a proof of age.

This vulnerability cannot be eliminated by authenticating the mDL reader, because [13, §7.2.1] states that "An mDL shall not require mdoc reader authentication as a precondition for the release of any of the mandatory data elements". It could be eliminated by removing the server retrieval token from the device engagement structure, as the current specification also has a method for transmitting the token to the reader in the encrypted session. But the ability to obtain the token without setting up an encrypted session is a valuable feature of the specification because it allows for a very simple implementation of the reader.

The proximity protocol described below in Figures 7a-7g mitigates this vulnerability instead by requiring the mDL holder to select a presentation profile when activating the app, and restricting the data elements that the mDL reader is able to obtain, not only from the mDL app but also from the mDL issuer, to those that are included in the presentation profile. When activating the app in a bar, for example, the mDL holder will activate a proof of age profile containing a single data element, viz. the age-over-21 attestation. The mDL app will then restrict the information that it provides itself to the mDL by returning an error response if the mDL reader requests any other data element. To restrict the information that the reader obtains from the mDL issuer, the app will include the presentation profile in the server retrieval token that the reader uses the authenticate to the issuer.

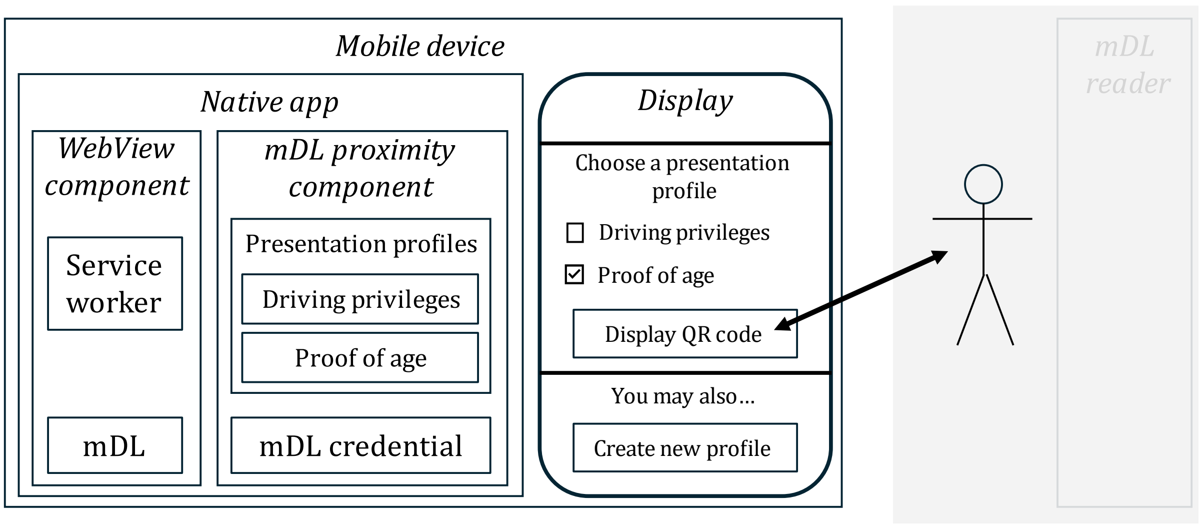

Figure 7a: User chooses presentation profile and displays QR code

In Figure 7a the user activates the mDL app by clicking on a button to display a QR code after choosing a presentation profile. Each profile consists of a set of data elements from the mDL data model. There are two standard profiles, labeled “Driving privileges” and “Proof of age”, and the user is given the option to create a custom profile. The user chooses the “Proof of age” profile, which comprises a single age attestation called “age-over-21”. Age attestations are virtual data elements, discussed in Section 13.1.4 of Chapter 13.

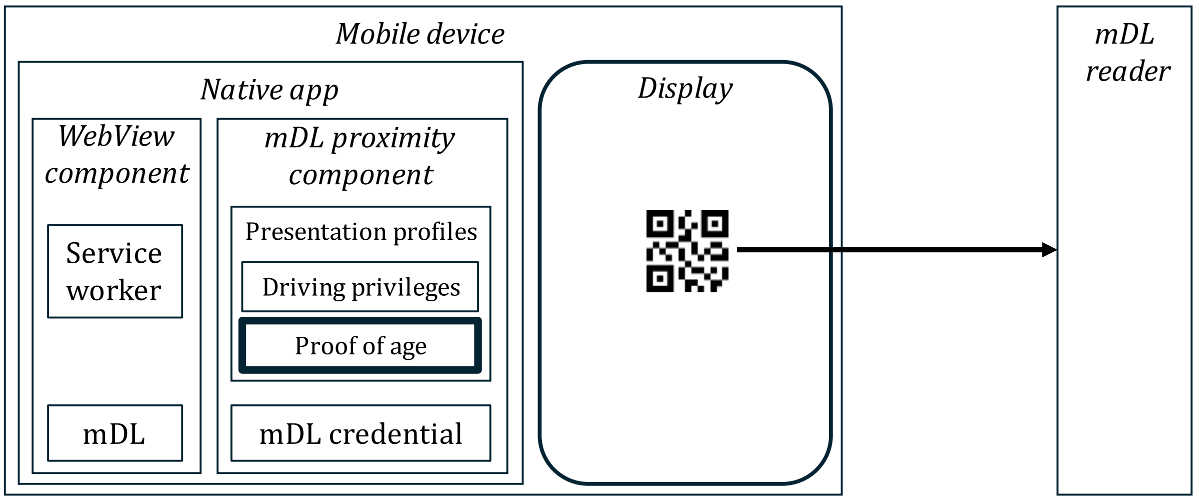

Figure 7b: User shows device engagement QR code to mDL reader

In Figure 7b the mDL proximity component of the native app has recorded the choice of presentation profile made by the user and will reject a request for any data element other than the age-over-21 age attestation. The display of the mobile device shows a QR code that is scanned by the mDL reader. The QR code encodes the device engagement structure of the mDL proximity component, which includes a random ephemeral ECDH public key.

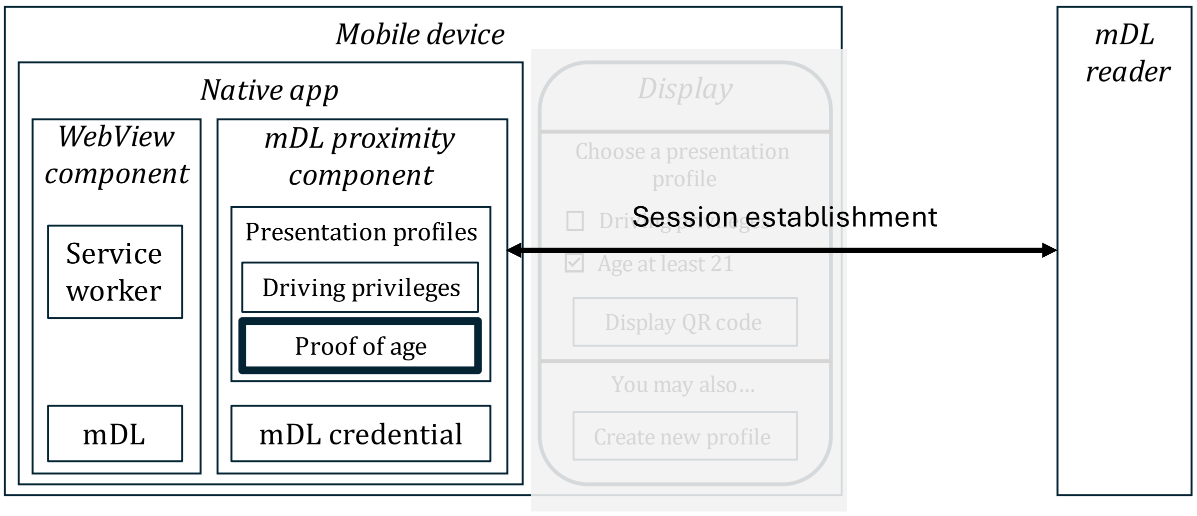

Figure 7c: ECDH handshake for establishment of encrypted session

In Figure 7c the mDL proximity component and the mDL reader establish an encrypted session by performing an ECDH key exchange, using the ECDH public key in the QR code, and the public key component of an ephemeral ECDH key pair generated by the mDL reader for this purpose.

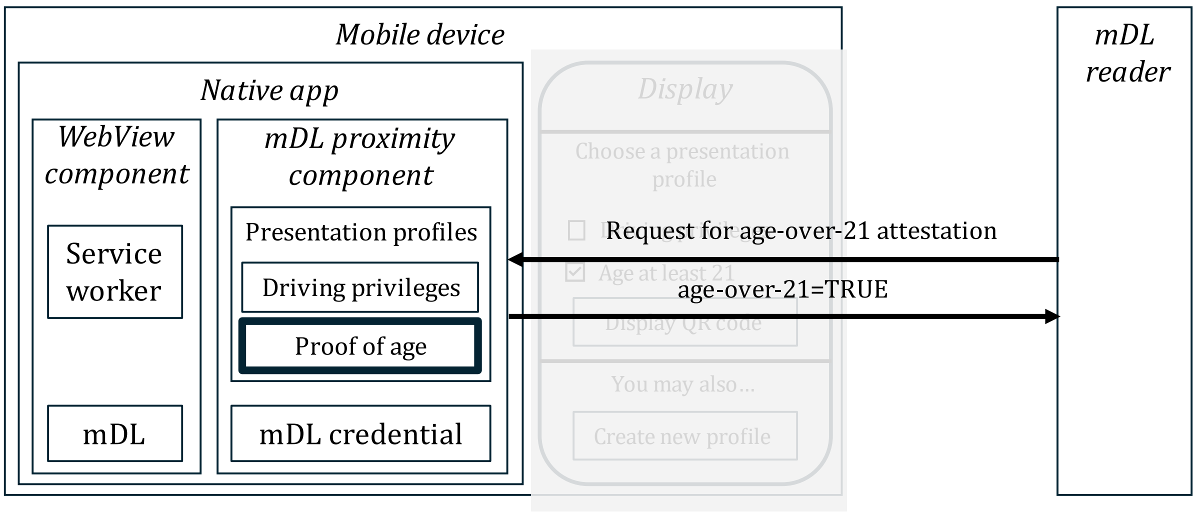

Figure 7d: mDL reader retrieves age attestation from mDL reader

In Figure 7d the mDL reader requests the age-over-21 age attestation. The mDL proximity component verifies that this data element is included in the Proof of age profile and responds to the request by sending the element.

Figures 7e-7g below show how the proof of age presentation profile can also restrict the information that the mDL reader can obtain from the mDL issuer when using server retrieval over the web.

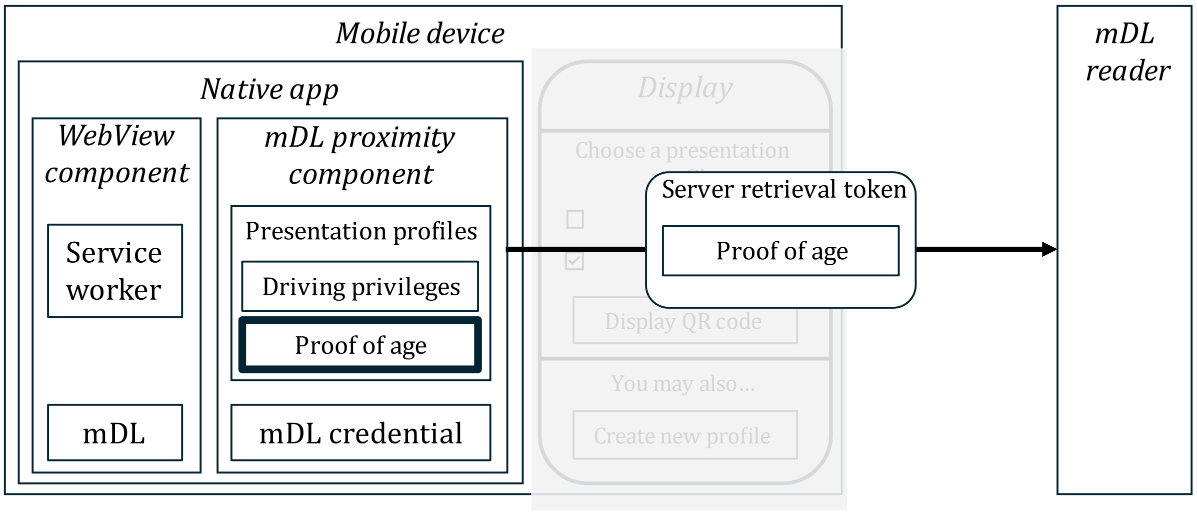

Figure 7e: mDL reader obtains server retrieval token with profile restriction

In Figure 7e, the mDL proximity component has included the selected presentation profile restriction in the server retrieval token, and the mDL reader obtains the token either from the device engagement structure or, after the encrypted session has been set up, in response to an explicit request.

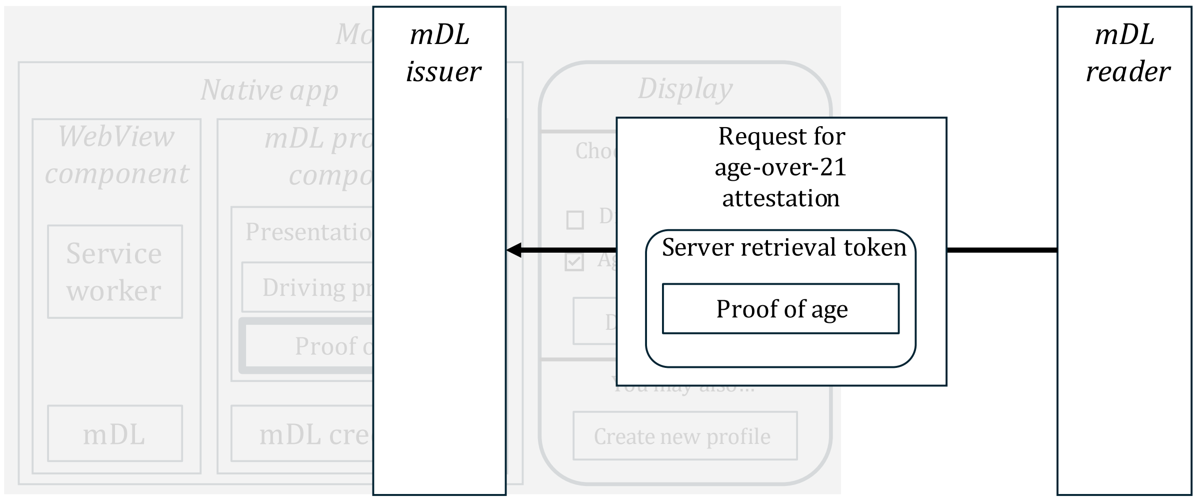

Figure 7f: mDL reader uses restricted service retrieval token for authentication

In Figure 7f the mDL reader sends a request for the age-over-21 to the mDL issuer over the web, with authentication by the server retrieval token with the proof of age restriction.

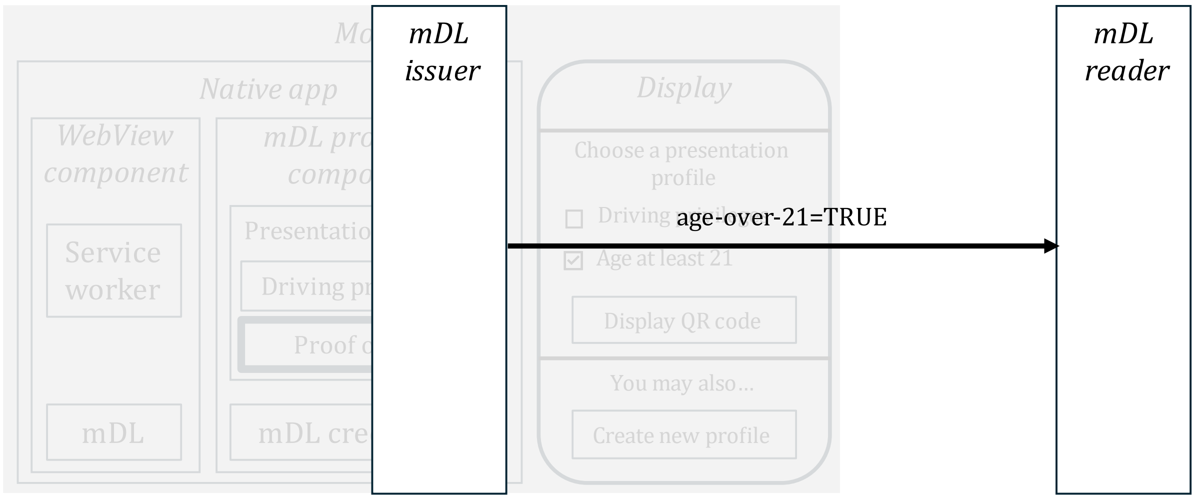

Figure 7g: mDL issuer sends requested attestation to mDL reader

In Figure 7g, the mDL issuer sends the age-over-21 attestation to the mDL reader after verifying that the attestation is included in the presentation profile restriction of the server retrieval token.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoincore.org/bitcoin.pdf.

[2] bitcoincore.org, "Bitcoin Core" [Online]. Available: https://github.com/bitcoin/bitcoin.

[3] S. Rostislav , "Bitcoin Clients," 2012. [Online]. Available: http://publications.theseus.fi/bitstream/handle/10024/47166/Skudnov_Rostislav.pdf?sequence=1.

[4] Wikipedia, "Mt. Gox" [Online]. Available: https://en.wikipedia.org/wiki/Mt._Gox.

[5] MetaMask, "Architecture," [Online]. Available: https://docs.metamask.io/wallet/concepts/architecture/.

[6] MetaMask, "User Guide: Dapps," [Online]. Available: https://support.metamask.io/third-party-platforms-and-dapps/user-guide-dapps/.

[7] Zooko Wilcox-O'Hearn. Names. Secure, Distributed, Human-Readable. Choose Two. https://www.cs.princeton.edu/courses/archive/spr17/cos518/papers/zooko-triangle.pdf

[8] UL white paper, "Verifiable Credentials and ISO/IEC 18013-5 Based Credentials," [Online]. Available: https://collateral-library-production.s3.amazonaws.com/uploads/asset_file/attachment/36416/CS676613_-_Digital_Credentials_promotion_campaign-White_Paper_R3.pdf.

[9] Government of British Columbia, "BC Wallet," [Online]. Available: https://www2.gov.bc.ca/gov/content/governments/government-id/bc-wallet.

[10] Government of British Columnia, "OrgBook BC," [Online]. Available: https://www.orgbook.gov.bc.ca/search.

[11] Government of British Columbia, "BC Wallet to hold Verifiable Credentials," [Online]. Available: https://github.com/bcgov/bc-wallet-mobile.

[12] Government of British Columnbia, "BCWallet_Architecture.png," [Online]. Available: https://github.com/bcgov/bc-wallet-mobile/blob/main/docs/BCWallet_Architecture.png.

[13] California Department of Motor Vehicles, "CA DMV Wallet and mDL Pilot," [Online]. Available: https://www.dmv.ca.gov/portal/ca-dmv-wallet/.

[14] Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity, COM/2021/281 final[

[15] European Commission, "The European Digital Identity Wallet Architecture and Reference Framework," 10 February 2023. [Online]. Available: https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-wallet-architecture-and-reference-framework.

[16] Talao, "EUDI Wallet and eIDAS 2: Understanding the role of EAA, QEAA and PID credentials for individuals and organizations," [Online]. Available: https://talao.io/blog/eudi-wallet-understanding-credentials-in-eidas-2-eaa-qeaa-and-pid/.

[17] F. Corella, "Storing Cryptographic Keys in Persistent Browser Storage," 2 June 2017. Blog post. Available: https://pomcor.com/2017/06/02/keys-in-browser/.

[18] F. Corella and K. Lewison, "Storing Cryptographic Keys in Persistent Browser Storage," in ICMC 2017. Available: https://pomcor.com/documents/KeysInBrowser.pdf.

[19] ISO/IEC TS 18013-7:2024Personal identification — ISO-compliant driving licence — Part 7: Mobile driving licence (mDL) add-on functions. https://www.iso.org/standard/82772.html.

[20] Samsung, "Secure Folder puts private data encryption in your hands," 21 August 2024. [Online]. Available: https://insights.samsung.com/2024/08/21/secure-folder-puts-private-data-encryption-in-your-hands-5/.

[21] B. Snyder and K. Vyas, "How to lock apps on your Android phone to prevent unauthorized access," [Online]. Available: https://www.xda-developers.com/how-to-lock-apps-on-your-android-phone/#how-to-lock-apps-using-the-built-in-app-lock-on-other-versions-of-android.

[22] Apple, "Lock or hide an app on iPhone," [Online]. Available: https://support.apple.com/en-ie/guide/iphone/iph00f208d05/18.0/ios/18.0.

[23] D. Alexandru, "AppLocker," [Online]. Available: https://apps.apple.com/us/app/applocker-passcode-lock-apps/id1132845904.

[24] Cisdem, "Cisdem AppCrypt," [Online]. Available: https://www.cisdem.com/appcrypt.html.

[25] ISO/IEC 18013-5:2021Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application. https://www.iso.org/standard/69084.html.

[26] T. Looker et al, "IRTF CFRG draft: The BBS Signature Scheme," 12 September 2024. [Online]. Available: https://identity.foundation/bbs-signature/draft-irtf-cfrg-bbs-signatures.html.

[27] S. Tessaro and C. Zhu, "Revisiting BBS Signatures," 2023. [Online]. Available: https://eprint.iacr.org/2023/275.

[28] Verifiable Credentials Data Model v1.1. W3C Recommendation 03 March 2022. https://www.w3.org/TR/vc-data-model/.

[29] Google, "WebView overview," 18 December 2024. [Online]. Available: https://developer.chrome.com/docs/webview.

[30] Apple, "WKWebView," [Online]. Available: https://developer.apple.com/documentation/webkit/wkwebview.

[31] Wikipedia, "Electronic identification," [Online]. Available: https://en.wikipedia.org/wiki/Electronic_identification.

[32] D. Fett, K. Yasuda and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)," 15 November 2024. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-oauth-selective-disclosure-jwt-14.html.

[33] Dynamsoft, "How to Use Barcode Scanner in Android WebView," [Online]. Available: https://www.dynamsoft.com/codepool/use-barcode-scanner-in-android-webview.html.

[34] N. Sakimura et al., "OpenID Connect Core 1.0 incorporating errata set 2," [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html.

[35] ToIP Ecosystem Foundry Working Group, "Bhutan NDI (National Digital Identity) & ToIP Digital Trust Ecosystems," 21 May 2024. [Online]. Available: https://trustoverip.org/permalink/Case-Study-Bhutan-NDI-National-Digital-Identity-ToIP-Digital-Trust-Ecosystems-V1.0-2024-05-21.ext.