

Identity Verification

CROSS REFERENCE TO RELATED APPLICATION

This application claims right of priority to and the benefit of British patent application no. 2402637.9 dated 23rd February 2024.

BACKGROUND

The verification of the true identity of a natural person is becoming essential in an increasing range of contexts. Anti-money laundering regulations require that large transactions, privileged operations and the opening of financial services accounts be attributed to persons whose identities have been adequately established. Security concerns require that persons traveling or entering sensitive installations should not conceal their true identities, so that their past records may first be scrutinized.

The most generally accepted source of evidence of the true identity of a person is that issued by governments. In order to issue passports or national identity cards, governments require trusted documentation, adequate corroboration and, if necessary, investigation. In countries where national identity cards do not exist, such as the United States or United Kingdom, reliance is placed on the driving license, and a substantial obligation of due diligence is placed on the Motor Vehicle or Driver administrative bodies that issue driving licenses.

Thus, the physical possession of a government-issued identity document (ID) such as a passport, identity card or high-trust driving license is generally considered a trustworthy corroboration of the identity asserted by an individual.

In a number of countries, digital identities are being created with the aspiration of being used instead of physical documents. In the European Union, the eIDAS Regulations permit the use of

compliant digital identities throughout the member states. Normally, to establish an eIDAS digital identity in the first place, a person must present themselves together with their government issued identity document.

The trust vested by the relying party in such a document depends on the document itself being genuine and un-falsified. For this reason, governments have for centuries invested in increasingly sophisticated methods of printing and laminating such documents, in a race to keep ahead of the increasing sophistication of forgers. In the last decade electronic means have been added, with the insertion of microchips into documents complying with the ICAO 9303 standard. Such microchips contain the information printed in the document, plus a digital certificate signed by the issuing authority. Under current cryptographic practice, such certificates are considered impossible to forge and hence represent the best possible guarantee of the authenticity of the information contained in the chip, which is readable contactlessly using NFC technology.

However, there are many circumstances in which such a chip is absent, or the means to read it are unavailable. For this reason, physical means to confirm authenticity remain important. There are three main measures employed. The first is security printing, which embeds layers of pattern in the printed document. Imperfect copies are exposed by distortions, errors or erasures in such patterns. The second is spectrographic response, which determines the way the document responds to illumination by ultra-violet light. The third method is to embed or emboss a holographic element in the document, whose reflective properties, in terms of color, directionality and intensity and combinations thereof are highly characteristic. Typical examples are holograms embedded in credit cards, and holographic embossing in the surface of driving licenses. Normally, such embossing includes or covers the identity photograph found on practically all identity documents.

There is an increasing requirement to check the identity of an individual remotely and automatically. In-person identity checks require space in an accessible location and consume employee time. Above all, they are inconvenient for the user, who must travel to the checking location and invest appreciable amounts of time in the process. They are increasingly reluctant to

do so, as the process confers no direct value in return for the investment of their time. Organizations able to permit users to prove their identity remotely, preferably on their now-ubiquitous smartphones, gain an immediate competitive advantage.

Early methods of remote identity checking required users to engage in a live video call with an agent, who would interview the person to ensure they were genuinely present humans, then examine the document via the video link, to establish its authenticity. This method, still widely mandated in the financial services regulations of many countries, has a number of drawbacks; it is very costly, requiring an employee to talk with the person for some minutes. It is also ineffective in security terms, as it is difficult for an employee to determine if the document has been forged or modified based just on the imagery available from a video call. The fine details embedded in security printing are not visible, and the correctness of the holograms cannot be adequately assessed.

To facilitate the online identification of users, the Open ID Connect protocol was developed, which has been widely adopted. OpenID Connect is an interoperable authentication protocol based on the OAuth 2.0 framework of specifications (IETF RFC 6749 and 6750), which simplifies the verification of the identity of users based on the authentication performed by an Authorization Server and to obtain user profile information in an interoperable and REST-like manner.

The OpenID Connect protocol, in abstract, follows these steps: The end user navigates to a website or web application (the Relying Party) via a browser and initiates a login. The end user is then redirected to an Identity Provider, trusted by the Relying Party, and verifies the identity to the Identity Provider, typically by signing in using their username and password. The Identity Provider then returns an Access Token to the Relying Party using a redirect URL, and the Relying Party uses the Access Token in a call to the Identity Provider to retrieve the user information, either in the form of an Identity Token or as JSON returned from a call to the user_info endpoint at the Identity Provider.

OpenID Connect enables application and website developers to launch sign-in flows and receive verifiable assertions about users across web-based, mobile, and JavaScript clients and advantageously removes the responsibility of setting, storing, and managing passwords which is frequently associated with credential-based data breaches. However, it still requires the use of databases of identity documents and the more widespread its use becomes, the greater the likelihood of data breaches. The large OpenID user base also has significant investment in their own OpenID infrastructure.

To address these perceived problems a decentralized identity approach is gaining popularity. Decentralized identity is a concept that gives back control of identity to consumers through the use of an identity wallet or digital wallet in which they collect verified information about themselves from certified issuers (such as the Government). By controlling what information is shared from the wallet to requesting third parties (e.g., when registering for a new online service), the user is able to better manage their identity online and their privacy.

EP3629516 discloses an exemplary decentralized identity approach in which a private key is generated in dependence on a start information; subsets of supplied user information are encrypted by means of the secret key such that all user information items is encrypted; the private key is discarded; the encrypted subsets of the user information items are distributed; and the secret key is regenerated for the decrypting of at least one of the encrypted subsets of the supplied user information items.

The known decentralized approaches, however, suffer from the problem that the costs of compliance and infrastructure maintenance fall on the individual wallet owner, who will rarely be able or want to bear them.

SUMMARY

There is therefore still a need to provide means to assess the authenticity of an identity document remotely, using the devices available to members of the public alone, in an automated, reliable and

fast way in order to minimize cost, in particular to the user being identified, minimize the time required, and maximize the reliability of the result.

According to the invention there is provided a method of verifying an identity for a relying party, which identity comprises verifiable credentials held in a digital wallet on a handheld device, wherein a user requests a login page from the relying party, the relying party then requesting information to be verified, the information to be verified being provided to an OIDC bridge, which OIDC bridge creates a signing request and a session identifier, wherein the user requests the signing request and approves the request to send verifiable credentials to the OIDC bridge, which OIDC bridge verifies the signing request and the verifiable credentials and generates an authorisation, the authorisation being provided to the relying party, the relying party then creating an access token request, which access token request is provided to the OIDC bridge, which OIDC bridge then provides user information to relying party to enable verification of the identity for the relying party.

The invention makes use of verifiable credentials, in which the recipient's identity can be authenticated before receiving a verifiable credential. This solution to an essential security feature that has previously limited the adoption due to the difficulties of strong authentication infrastructure. The solution provides the ability to collect and store credentials locally.

The solution of the invention provides an OIDC bridge, which makes a distributed identity verification system accessible to legacy systems which rely upon OIDC. The solution advantageously provides that the user approves the verification request. For example, when a site requests ID verification, the user can choose and for what purpose the information is shared. Therefore not only is the information shared but also the user consent, which facilitates formation of a contract. In certain embodiments it would be possible to post the verified information directly into a web form, which enables the quality and reliability of the information posted to be assured. This is particularly advantageous when engaging with sites for bookings such as airline tickets, where discrepancies in the names of passengers and their ID document can be eliminated.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments of the invention will now be described in greater detail with reference to the drawings in which:

Fig. 1 shows an OIDC flow for a desktop browser and embedded QR Code;

Fig. 2 shows an OIDC flow with desktop browser and QR Code page;

Fig. 3 shows an OIDC flow for a mobile browser with an embedded QR Code;

Fig. 4 shows an OIDC flow for a mobile browser and separate QR;

Fig. 5 shows embodiment with biometric verification

Fig. 6 shows an embodiment in which the Relying Party can generate a QR Code

DETAILED DESCRIPTION

Verifiable credentials are a digital representation of a person's identity and credentials, such as their name, date of birth, educational qualifications, work experience, and more. Unlike traditional identity documents like passports or a driver's licence, verifiable credentials are stored digitally and can be easily accessed and verified online. They are designed to be tamper-proof and secure, making it difficult for fraudsters to falsify or alter them. There are three fundamental roles when it comes to utilizing verifiable credentials: the Issuer, which is a person or organization that issues verifiable credentials such as a government department issuing a national ID or a college issuing a diploma to a person; a holder: which is a person or organization that owns or holds the verifiable credential in their digital wallet and a verifier, who is the person or organization validating or authenticating the credential like a security agent at the airport checking your passport. The digital wallet can be stored on a user's smartphone or other handheld device such as a tablet or, alternatively, it would be possible to hold the information in a web wallet. A digital or web wallet is an app or secured storage which contains data, protected biometrically or by other access features

of the platform, that can enable a user to interact with other services, in which the app is the only app that can access the data within the app.

Figure 1 shows a first exemplary embodiment of an OIDC flow for a desktop browser and embedded QR Code. In a first step, the desktop browser requests a login page with QR code from the Relying Party, which in turn requests a login from the OIDC Bridge and specifies that it wants to receive information from the OIDC bridge so that the Relying Party can build and display a page with an embedded QR code. The requested URL is provided with certain OIDC request parameters.

The OIDC bridge verifies the incoming OIDC parameters and retrieves settings for the given client_id of the Relying Party from a configuration database. The Relying Party provides predefined parameters to the OIDC bridge as part of the configuration data. Each Relying Party may have more than one client_id, which functions as an identifier for each use case of the Relying Party. This information is then used to create a signing request. The signing request can be a JSON web token (JWT), which JWT contains all the information that the Relying Party has requested. The signing request is persisted in a local cache and a random session identifier is created by the OIDC bridge. The OIDC bridge then sends a response with the a) information that is contained in the QR code so that the Relying Party can create the QR code; b) the QR code as data URL that can be directly embedded in the page or c) a URL to the OIDC bridge that creates the QR code (which can be used as the source of an image in the page).

A User then scans the QR code displayed in the browser using an app on a hand held device such as a smart phone or tablet. The desktop browser then gets the signing request from the OIDC Bridge, typically using an HTTP GET request with session identifier as parameter. The app in this case stored on the phone does not require to have intelligence but facilitates communication with the verification system backend enabling a localisation dynamic from the backend. The App can be a form of wallet for the user's personal information. Each piece of information in the wallet such as the name, surname, date of birth is represented by either separate verifiable credentials or verifiable credentials that support selective disclosure (such as those using SD-JWT). The Relying

Party requests a user to share a set of claims. The claims can include information such as the name, surname, date of birth, gender, telephone number, age etc.

The OIDC Bridge then fetches the signing request JWT from the local cache and returns the signing request to the relying party. The signing request contains: 1) The set of claims that the Relying Party is requesting the user to share; 2) Information about the Relying Party (URL, name, logo); 3) the reason that the Relying Party is requesting the information. The reason is requested to ensure that the user is providing informed consent to the request and for GDPR compliance. The reason can then further be traced for future queries about the request.

The Signing Request is then returned to the user's handheld device. A browser on the hand-held device then shows selected disclosure dialogue. Assuming the user then approves the request, the request can then be signed with the App's private key.

The Requested verifiable credentials are then posted by the smartphone app directly to the OIDC bridge either as individual items or as a verifiable presentation and in both cases the response is signed by the private key of the smartphone app and contained in a signed Request. The OIDC Bridge then verifies the signature of the signed Request and then verifies the signatures of the returned credentials (which should be signed by a trusted issuer) before preparing the id_token and userinfo. A trusted issuer is an issuer whose public key is listed within the OIDC bridge as being trusted. This could be either by way of a chain of X.509 certificates or using a trust framework.

The OIDC bridge then sends redirect information with an Authcode to the desktop browser. The redirected pages with the Authcode is sent as an HTTP GET request to the Relying Party. The Relying Party then requests the Access Token using the Authcode and the client_secret Key to the OIDC bridge. The OIDC bridge then provides the Access Token and ID Token to the Relying Party. The Relying Party requests the Userinfo using the Access Token from the OIDC bridge. The OIDC bridge then provides the Userinfo to the Relying Party. The Relying Party then serves a logged in page to the desktop browser.

Figure 2 shows a further embodiment for an OIDC Flow with desktop browser and QR Code page. In a first step, the desktop browser requests a login page with a QR code from the Relying Party, which in turn requests a login from the OIDC Bridge and specifies that it wants the OIDC bridge to display a page with a QR code hosted by the OIDC bridge. The requested URL is provided with certain OIDC request parameters.

The OIDC bridge verifies the incoming OIDC parameters and retrieves settings for the given client_id from a configuration database. This information is then used to create a signing request. The signing request can be a JSON web token (JWT), which JWT contains all the information that the relying party has requested. The signing request is persisted in a local cache and a random session identifier is created by the OIDC bridge. A redirect to the qrauth page is then returned to the desktop browser. The desktop browser then requests the qrauth page from the OIDC bridge. The qrauth page has a URL with the sessionid in the address. A QR code is then constructed which contains the session identifier as well as a bridge id identifier. The returned qrauth html page contains the QR code as a data URL.

To sign in a user then scans the QR code with their smartphone or other handheld device. The App can then send a signing request, eg an http GET request with a session identifier as a parameter, to the OIDC bridge. The OIDC Bridge then fetches the signing request JWT from the local cache and returns the signing request to the relying party. The signing request contains: 1) The set of claims that the Relying Party is requesting the user to share; 2) Information about the Relying Party (URL, name, logo); 3) the reason that the Relying Party is requesting the information. The Signing Request is then returned to the user's handheld device.

The App then shows the selected disclosure dialogue to the user, who can then approve the request. The request is then signed with the App's private key in the App. The requested verifiable credentials are then returned to the OIDC bridge, for example with an HTTP Post message. The

OIDC bridge can then verify the signature of the signing request, verify signatures of returned credentials (which should be signed by a trusted issuer) and then prepare the id_token and userinfo.

The OIDC bridge can then send redirect information with an authcode, eg using a WebSocket push, to the desktop browser. The redirected page with the authcode is then provided to the Relying Party.

The Relying Party then requests the Access Token using the Authcode and the client_secret Key to the OIDC bridge. The OIDC bridge then provides the Access Token and ID Token to the Relying Party. The Relying Party requests the Userinfo using the Access Token from the OIDC bridge. The OIDC bridge then provides the Userinfo to the Relying Party. The Relying Party then serves a logged in page to the browser.

Figure 3 shows an OIDC flow for a mobile browser with an embedded QR Code. In a first step, the desktop browser requests a login page with an embedded QR code and a login button that triggers a deep link to the App from the Relying Party, which in turn requests a login from the OIDC Bridge and specifies that it wants to receive information from the OIDC bridge so that the Relying Party can build and display a page with an embedded QR code and a login button. The requested URL is provided with certain OIDC request parameters.

The OIDC bridge verifies the incoming OIDC parameters and retrieves settings for the given client_id from a cached storage. This information is then used to create a signing request. The signing request can be a JSON web token (JWT), which JWT contains all the information that the relying party has requested. The signing request is persisted in a local cache and a random session identifier is created by the OIDC bridge.

A response is then sent by the OIDC bridge to the Relying Party with either a) the information that is contained in the QR Code so that the Relying Party can create the QR Code; b) the QR Code as

a Data URL that can directly be embedded in the page or c) a URL to the OIDC bridge that creates the QR Code, which can be used as the source of an image in the page.

The login page is then served to the mobile browser. The user can then click on the login page, which will open an App with a deep link. The App can then send an HTTP GET request with a session identifier as a parameter of the request to request the signing request.

The OIDC Bridge fetches the signing request JWT from the local cache and returns it to the relying party in the App. The signing request contains: 1) the set of claims that the Relying Party is requesting the user to share; 2) Information about the Relying Party (url/name/logo); 3) The reason that the Relying Party is requesting the information.

The App can then show selected disclosure dialogue to the user, who can approve the request. The request can then be signed with a private key stored in the App or on the device. The requested verifiable credentials can then be returned to the OIDC bridge. The OIDC bridge can then verify the signature of the signing request, verify signatures of returned credentials (which should be signed by a trusted issuer) and then prepare the id_token and userinfo.

After the app has returned the signed request, control needs to be returned to the smartphone browser, which presents multiple challenges on iOS. To go back to the browser app that invoked the app, the browser can be identified, for example, from the UserAgent string. A redirect URL is returned to the app, which then opens the qrauth page on the smartphone browser. The qrauth page offers a specific URL known by the OIDC bridge that can then be opened. The page can then ping the QR bridge, eg using javascript, so that it can receive the correct URL using WebSocket.

For an android App to ensure that the WebSocket notification works after the browser has become visible, a watcher is added to the page that triggers on visibility events to reload the page.

In both iOS and Android, if the page is hidden when the redirected URL is received through a WebSocket push, the page will not reload. An event listener can be added to the page when the WebSocket push has been received. The event listener then triggers on visibility changes and then sets the redirect URL, thus reloading the page, when the page becomes visible.

The OIDC bridge then sends the redirect information with Authcode by means of a WebSocket push to the smartphone browser. The redirected page with the Authcode in smartphone browser then sends an HTTP Get request to the Relying Party.

The Relying Party then requests the Access Token using the Authcode and the client_secret Key to the OIDC bridge. The OIDC bridge then provides the Access Token and ID Token to the Relying Party. The Relying Party requests the Userinfo using the Access Token from the OIDC bridge. The OIDC bridge then provides the Userinfo to the Relying Party. The Relying Party then serves a logged in page to the browser.

Figure 4 shows an OIDC flow for a mobile browser and separate QR Code implementation, in which the mobile browser sends a request to the Relying Party for a login page and the Relying Party responds with the login page which contains both a QR code and a button with a deep link to open the App.

The OIDC bridge verifies the incoming OIDC parameters and retrieves settings for the given client_id from a cached storage. This information is then used to create a signing request. The signing request can be a JSON web token (JWT), which JWT contains all the information that the relying party has requested. The signing request is persisted in a local cache and a random session identifier is created by the OIDC bridge.

The OIDC bridge then provides a redirect response to the qrauth page in the browser instance. The qrauth page has a URL with the sessionid in the address. A QR code is then constructed with contains the session identifier as well as a bridge id identifier. The returned html page contains the QR Code as a data URL.

The user then clicks on the button which opens an app with a deep link. The App then sends a GET signing request to the OIDC bridge. This request can include a session identifier as a parameter.

The OIDC bridge fetches the signing request JWT from the local cache and returns it to the relying party. The signing request contains: the set of claims that the Relying Party is requesting the user to share; information about the Relying Party (url/name/logo); the reason that the Relying Party is requesting the information. The signing request is then returned to the app, which shows the selected disclosure dialogue to the user, who can then approve the request. Once the user has approved the request, the App can sign the request with the App private key.

The requested verifiable credentials are then returned to the OIDC bridge. The OIDC bridge can then verify the signature of the signing request, verify signatures of returned credentials (which should be signed by a trusted issuer) and then prepare the id_token and userinfo.

After the app has returned the signed request, control needs to be returned to the smartphone browser, which presents multiple challenges on iOS. To go back to the browser app that invoked the app, the browser can be identified from the UserAgent string. A redirect URL is returned to the app, which then opens the qrauth page on the smartphone browser. The qrauth page offers a specific URL known by the OIDC bridge that can then be opened. The page can then ping the QR bridge, eg using javascript, so that it can receive the correct URL using WebSocket.

For an android App to ensure that the WebSocket notification works after the browser has become visible, a watcher is added to the page that triggers on visibility events to reload the page.

In both iOS and Android, if the page is hidden when the redirected URL is received through a WebSocket push, the page will not reload. An event listener can be added to the page when the WebSocket push has been received. The event listener then triggers on visibility changes and then sets the redirect URL, thus reloading the page, when the page becomes visible.

The OIDC bridge then sends the redirect information with Authcode by means of a WebSocket push to the smartphone browser. The redirected page with the Authcode in smartphone browser then sends an HTTP Get request to the Relying Party.

The Relying Party then requests the Access Token using the Authcode and the client_secret Key to the OIDC bridge. The OIDC bridge then provides the Access Token and ID Token to the Relying Party. The Relying Party requests the Userinfo using the Access Token from the OIDC bridge. The OIDC bridge then provides the Userinfo to the Relying Party. The Relying Party then serves a logged in page to the browser.

Figure 5 shows a further embodiment in which a biometric verification is possible. When a biometric verification is requested by a relying party using face_present as the acr (Authentication Context Class Reference) value in the OIDC request, then this credential is not available in the client or relying party in known systems. The client or relying party can therefore initiate a biometric verification and requests that the Credentials issuer create new credentials that include the state and the nonce value present in the OIDC request. The nonce value is a time-varying value that has at most a negligible chance of repeating, for example, a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these. The nonce value is useful in defeating replay attacks. In an embodiment a sequence of illumination (i.e., the flash illuminator being turned off and on in a given sequence or continuously modulated according to a pattern) is provided by the server or randomly selected by software running on the smartphone to prevent spoofing by replay attacks, as described, for example, in U.S. Patent No. 9,075,975, which wholly incorporated herein by reference.

In a first step the Desktop Browser sends a Request Login pages Ger request to the Relying Party, which serves a login pages with Open ID Request URL as a login button to the Desktop Browser

The User then clicks the login button and the browser sends the request login by means of an HTTP GET request to the OIDC bridge. The URL has the OIDC request parameters.

The OIDC bridge verifies the incoming OIDC parameters and retrieves settings for the given client_id from a cached storage. This information is then used to create a signing request. The signing request can be a JSON web token (JWT), which JWT contains all the information that the relying party has requested. The signing request is persisted in a local cache and a random session identifier is created by the OIDC bridge. A redirect to the qrauth page is the returned to the desktop browser. The desktop browser then requests the qrauth page from the OIDC bridge. The qrauth page has a URL with the sessionid in the address. A QR code is then constructed which contains the session identifier as well as a bridge id identifier. The returned qrauth html page contains the QR code as a data URL.

To sign in a user then scans the QR code with their smartphone or other handheld device. The App can then send a signing request, eg an http GET request with a session identifier as a parameter, to the OIDC bridge. The OIDC Bridge then fetches the signing request JWT from the local cache and returns the signing request to the relying party. The signing request contains: 1) The set of claims that the Relying Party is requesting the user to share; 2) Information about the Relying Party (URL, name, logo); 3) the reason that the Relying Party is requesting the information. The signing request is then returned to the user's handheld device.

The App then checks whether credentials for a biometric presence verification are present or missing and in the case that they are missing initiates a biometric verification. The biometric verification can be made with a previously issued credential for a document photo. This request is sent to a Credential Issuer. The Credential Issuer then provides the request to a biometric verification provider, such as iProov®. An example of such a system is disclosed in GB2584547, which is wholly incorporated herein by reference.

The biometric verification provider then returns a token to the credential issuer. The credential issuer then requests a facial verification from the App, which in turn initiates a facial verification on the users' smartphone or device. The App then uses the token to enable authorisation to the biometric verification provider to authorise the face, which then performs a verification. The status of this verification is then returned to the App, which then returns the face status to the Credential Issuer. The credential issuer then provides the token to the biometric verification provider, which checks the result and provides this to the credential issuer. The credential issuer then creates the biometric credentials and returns these to the App.

The App then shows the selected disclosure dialogue to the user, who can then approve the request. The request is then signed with the App's private key in the App. The requested verifiable credentials are then returned to the OIDC bridge, for example with an HTTP Post message. The OIDC bridge can then verify the signature of the signing request, verify signatures of returned credentials (which should be signed by a trusted issuer) and then prepare the id_token and userinfo.

The OIDC bridge can then send redirect information with an authcode, eg using a WebSocket push, to the desktop browser. The redirected page with the authcode is then provided to the Relying Party.

The Relying Party then requests the Access Token using the Authcode and the client_secret Key to the OIDC bridge. The OIDC bridge then provides the Access Token and ID Token to the Relying Party. The Relying Party requests the Userinfo using the Access Token from the OIDC bridge. The OIDC bridge then provides the Userinfo to the Relying Party. The Relying Party then serves a logged in page to the browser.

Figure 6 shows a further alternative embodiment in which the Relying Party can generate a QR Code without interacting with the OIDC bridge. In this embodiment, the Relying Party pre-registers the OIDC parameters that should be used with a specific client_id. The QR Code can then be created entirely by the relying party and contains the client-id, state and nonce parameters. These can be separated by the dot character. It would be possible to additionally use PKCE (Proof Key for Code Exchange).

In this embodiment the mobile or smartphone browser sends a request for the login page to the Relying Party. The Relying Party then generates unique nonce and state values for the QR code and then generates the QR code locally and returns the login page to the mobile browser. The user then clicks on the login page to open the app with a deep link. The App then sends a Get signing request, eg an HTTP GET request with the session identifier as parameter, to the OIDC bridge.

The OIDC bridge detects that the incoming session_id belongs to a pre-generated QR code since it contains at least two dot characters, which are otherwise not allowed in the session_id. The OIDC Bridge uses the client_id to fetch the previously stored values for the OIDC request, generates the signing request JWT and returns it to the relying party. The signing request contains: 1) The set of claims that the Relying Party is requesting the user to share; 2) Information about the Relying Party (URL, name, logo); 3) the reason that the Relying Party is requesting the information. The Signing Request is then returned to the user's handheld device.

The App then shows the selected disclosure dialogue to the user, who can then approve the request. The request is then signed with the App's private key in the App. The requested verifiable credentials are then returned to the OIDC bridge, for example with an HTTP Post message. The OIDC bridge can then verify the signature of the signing request, verify signatures of returned credentials (which should be signed by a trusted issuer) and then prepare the id_token and userinfo.

After the app has returned the signed request, control needs to be returned to the smartphone browser, which presents multiple challenges on iOS. To go back to the browser app that invoked the app, the browser can be identified from the UserAgent string. A redirect URL is returned to the app, which then opens the qrauth page on the smartphone browser. The qrauth page offers a specific URL known by the OIDC bridge that can then be opened. The page can then ping the QR bridge, eg using javascript, so that it can receive the correct URL using WebSocket.

For an android App to ensure that the WebSocket notification works after the browser has become visible, a watcher is added to the page that triggers on visibility events to reload the page.

In both iOS and Android, if the page is hidden when the redirected URL is received through a WebSocket push, the page will not reload. An event listener can be added to the page when the WebSocket push has been received. The event listener then triggers on visibility changes and then sets the redirect URL, thus reloading the page, when the page becomes visible.

The OIDC bridge then sends the redirect information with Authcode by means of a WebSocket push to the smartphone browser. The redirected page with the Authcode in smartphone browser then sends an HTTP Get request to the Relying Party.

The Relying Party then requests the Access Token using the Authcode and the client_secret Key to the OIDC bridge. The OIDC bridge then provides the Access Token and ID Token to the Relying Party. The Relying Party requests the Userinfo using the Access Token from the OIDC bridge. The OIDC bridge then provides the Userinfo to the Relying Party. The Relying Party then serves a logged in page to the browser.

Although all the described embodiments use QR codes, it would be possible to use other forms of coding such as sound, optical or Bluetooth®, a link or short code, eg six digits. The QR code or an alternative is used in the invention to transfer a session identifier in the OIDC session to the client or Relying Party. The content of the QR code could also be encrypted. The OIDC bridge can also translate information between the Relying Party and the user wallet as the Relying Party can specify the format of the returned information from the OIDC bridge.

The various components of the system described herein may be implemented as a computer program using a general-purpose computer system. Such a computer system typically includes a main unit connected to both an output device that displays information to an operator and an input device that receives input from an operator. The main unit generally includes a processor connected to a memory system via an interconnection mechanism. The input device and output device also are connected to the processor and memory system via the interconnection mechanism.

One or more output devices may be connected to the computer system. Example output devices include, but are not limited to, liquid crystal displays (LCD), plasma displays, OLED displays, various stereoscopic displays including displays requiring viewer glasses and glasses-free displays, cathode ray tubes, video projection systems and other video output devices, loudspeakers, headphones and other audio output devices, printers, devices for communicating over a low or high bandwidth network, including network interface devices, cable modems, and storage devices such as disk, tape, or solid state media including flash memory. One or more input devices may be connected to the computer system. Example input devices include, but are not limited to, a keyboard, keypad, track ball, mouse, pen/stylus and tablet, touchscreen, camera, communication device, and data input devices. The invention is not limited to the particular input or output devices used in combination with the computer system or to those described herein.

The computer system may be a general-purpose computer system, which is programmable using a computer programming language, a scripting language or even assembly language. The computer system may also be specially programmed, special purpose hardware. In a general-purpose computer system, the processor is typically a commercially available processor. The general-purpose computer also typically has an operating system, which controls the execution of other computer programs and provides scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management and memory management, and communication control and related services. The computer system may be connected to a local network and/or to a wide area network, such as the Internet. The connected network may transfer to and from the computer system program instructions for execution on the computer, media data such as video data, still image data, or audio data, metadata, review and approval information for a media composition, media annotations, and other data.

A memory system typically includes a computer readable medium. The medium may be volatile or nonvolatile, writeable or nonwriteable, and/or rewriteable or not rewriteable. A memory system typically stores data in binary form. Such data may define an application program to be executed by the microprocessor, or information stored on the disk to be processed by the application

program. The invention is not limited to a particular memory system. Time-based media may be stored on and input from magnetic, optical, or solid-state drives, which may include an array of local or network attached disks.

A system such as described herein may be implemented in software, hardware, firmware, or a combination of the three. The various elements of the system, either individually or in combination may be implemented as one or more computer program products in which computer program instructions are stored on a non-transitory computer readable medium for execution by a computer or transferred to a computer system via a connected local area or wide area network. Various steps of a process may be performed by a computer executing such computer program instructions. The computer system may be a multiprocessor computer system or may include multiple computers connected over a computer network or may be implemented in the cloud. The components described herein may be separate modules of a computer program, or may be separate computer programs, which may be operable on separate computers. The data produced by these components may be stored in a memory system or transmitted between computer systems by means of various communication media such as carrier signals.

Having now described an example embodiment, it should be apparent to those skilled in the art that the foregoing is merely illustrative and not limiting, having been presented by way of example only. Numerous modifications and other embodiments are within the scope of one of ordinary skill in the art and are contemplated as falling within the scope of the invention.

Claims

1. A method of verifying an identity for a relying party, which identity comprises verifiable credentials held in a digital wallet on a handheld device, wherein a user requests a login page from the relying party, the relying party then requesting information to be verified, the information to be verified being provided to an OIDC bridge, which OIDC bridge creates a signing request and a session identifier, wherein the user requests the signing request and approves the request to send verifiable credentials to the OIDC bridge, which OIDC bridge verifies the signing request and the verifiable credentials and generates an authorisation, the authorisation being provided to the relying party, the relying party then creating an access token request, which access token request is provided to the OIDC bridge, which OIDC bridge then provides user information to relying party to enable verification of the identity for the relying party.
2. A method of verifying an identity according to Claim 1, wherein the OICD bridge creates a code, in particular a QR code, which code represents or contains the session identifier and which code is transferred by the OIDC bridge to be readable by a user device.
3. A method of verifying an identity according to Claim 1 or Claim 2, wherein the OIDC bridge provides information so that the Relying Party can create the code.
4. A method of verifying an identity according to any one of Claims 1 to 3, wherein the code comprises data URL that can be directly embedded in the page presented to the user
5. A method of verifying an identity according to any one of Claims 1 to 4, wherein a URL to the OIDC bridge is provided and the OIDC bridge creates the code.
6. A method of verifying an identity according to any one of Claims 1 to 5, wherein the code is provided to an app stored on a user device, which app holds verifiable credential information about the user, wherein the app displays the information requested in the signing request and requests approval from the user and wherein if the user approves the signing request, the signing request is signed by a private key associated with use of the app and the requested verifiable credential information is returned to the OIDC bridge.

7. A method of verifying an identity according to any one of Claims 1 to 6, wherein when the OIDC bridge has received the verifiable credential information, the Relying Party can then requests an access token from the OIDC bridge which can then provide the access token and an ID Token to the Relying Party, the Relying Party then requests the identity information of the user using the Access Token from the OIDC bridge, the OIDC bridge then provides the identity information to the Relying Party.
8. A method of verifying an identity according to any one of Claims 1 to 7, wherein the signing request includes a set of claims that the Relying Party is requesting the user to share and information about the Relying Party.
9. A method of verifying an identity according to Claim 8, wherein the signing request further comprises the reason that the Relying Party is requesting the information.
10. A method of verifying an identity according to any one of Claims 1 to 9, wherein the App checks whether credentials for a biometric presence verification are present or missing and in the case that they are missing initiates a biometric verification.
11. A method of verifying an identity according to Claim 10, wherein the biometric verification can be made with a previously issued credential for a document photo and the request is sent to a Credential Issuer, which Credential Issuer then provides the request to a biometric verification provider.
12. A method of verifying an identity according to Claim 10 or Claim 11, wherein the credential issuer then requests a facial verification from the App, which in turn initiates a facial verification on the users' smartphone or device.
13. A method of verifying an identity according to any one of Claims 10 to 12, wherein the App uses a token provided by the biometric verification provider to enable the biometric verification provider to authorise the face, which then performs a verification.
14. A method of verifying an identity according to Claim 13, wherein the status of the verification is then returned to the App, which then returns the face status to the Credential Issuer, wherein the credential issuer then provides the token to the biometric verification provider, which checks the result and provides this to the credential issuer, the credential issuer then creates the biometric credentials and returns these to the App.

15. A method of verifying an identity according to any one of Claims 1 to 14, wherein the Relying Party pre-registers the OIDC parameters that should be used with a specific client_id so that the code can then be created entirely by the relying party, which code contains the client-id, state and nonce parameters.
16. A method of verifying an identity according to Claim 15, wherein the code comprises at least two dot characters, which enables the OIDC bridge to recognise that the code has been generated by the relying party, wherein the OIDC Bridge then uses the client_id to fetch the previously stored values for the OIDC request, generates the signing request and returns it to the relying party.