yubico

Universal Second Factor and OpenID Connect

White Paper

August 11, 2015

Justin Richer
justin@bspk.io
for Yubico

yubico

Copyright

Trademarks

Disclaimer

Contact Information

Yubico Inc.
420 Florence Street, Suite 200
Palo Alto, CA 94301
USA
yubi.co/contact

Justin Richer
Bespoke Engineering
justin@bspk.io
https://bspk.io/

yubico

Contents

Introduction

Digital identity and authentication technologies are currently undergoing some major changes. Once the realm of centralized authorities like large enterprises and governments, new technologies are putting more power in the hands of end users in ways never before possible. Today's technology stack is built on the lessons of the past, with an understanding of what works and what does not in the real world with real users.

Among these technologies are Universal Second Factor (U2F) and OpenID Connect (OIDC), both relatively new open standards. On the surface they may seem to provide similar capabilities for user authentication, but in reality they are quite different. Some of these differences are fundamental, while others are more subtle and complex. In this whitepaper, we will examine these two identity technologies, comparing and contrasting their features, functions, and design goals. We will then explore several means by which they could be used together to provide flexible and powerful authentication capabilities in a dynamic and user-centered world.

Universal Second Factor

Universal Second Factor, or U2F, is a specification published by an industry consortium called the FIDO Alliance. U2F allows end users to register specialized devices (connected over protocols such as USB, NFC, or Bluetooth) to be used as a second-factor security component with U2F-enabled sites and applications. These devices are designed to be simple, inexpensive, and interoperable. They provide key-based security that can be deployed and used easily alongside other identity infrastructures.

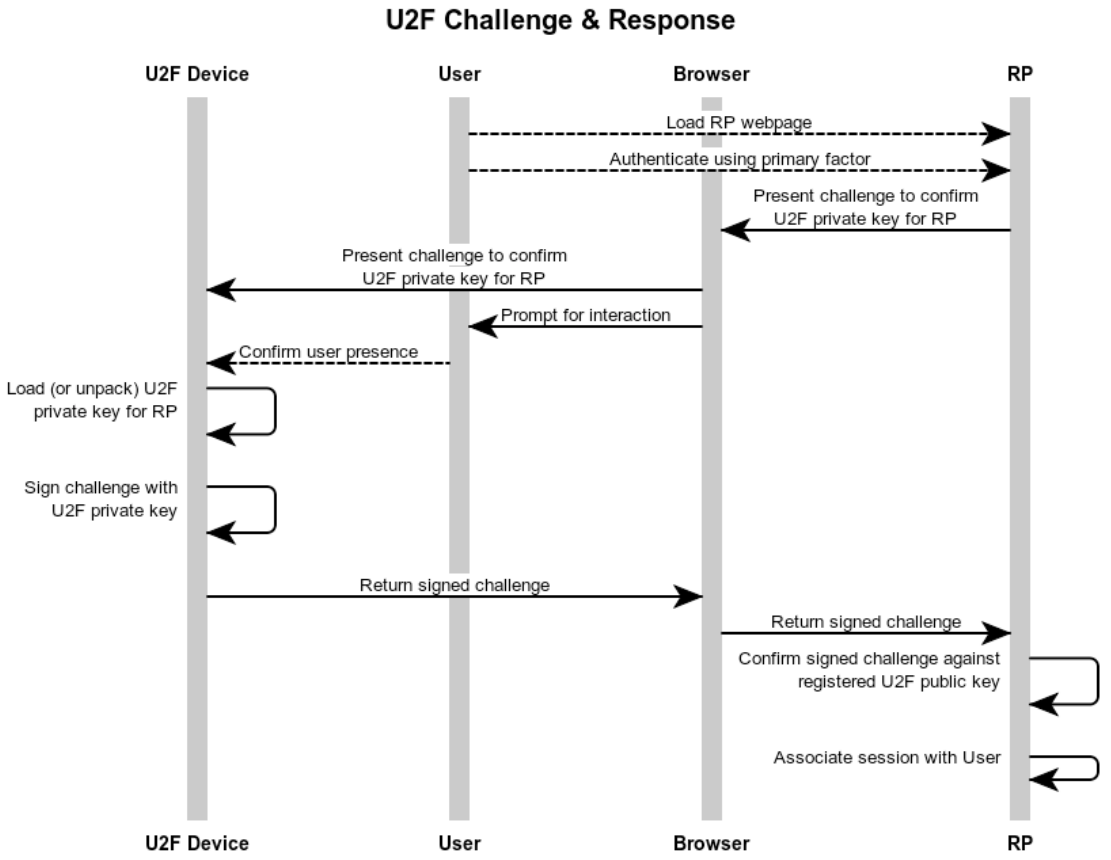
A U2F device can be thought of as a personal digital cryptographic key generator and verifier: something that creates and confirms cryptographic keys for use with compatible applications. In this way, the protocol allows a U2F-enabled site or application to determine whether a specific U2F device is present during a transaction by checking for the presence of a key. Some classes of device can determine if a user is present (not necessarily a particular user, but some human), but as far as the protocol is concerned, it can only show proof of access to the correct private key. While some types of U2F devices store the generated keys in a stateful manner, others issue stateless keys recognizable by the device. This latter technique of embedding the encrypted private key into the public key allows the device to function without writeable local storage, enabling it to scale to a practically unlimited number of sites.

Regardless of how it's implemented, the U2F device provides two functions: key generation and signature verification. A new key can be generated for every site the device is used with, and the presentation of a challenge to the device along with its public key allows the device to verify that it can produce the associated private key for that site. This all happens with a simple user experience, where the user usually needs to simply press a button on their device when prompted.

With all this, the U2F device fits the "something you have" component of a good authentication system design. It is, however, only a single factor, and one that can be physically used and shared by different people. By design, whoever has the device has access to the keys, and therefore the ability to cryptographically prove access to the keys.

What U2F can do

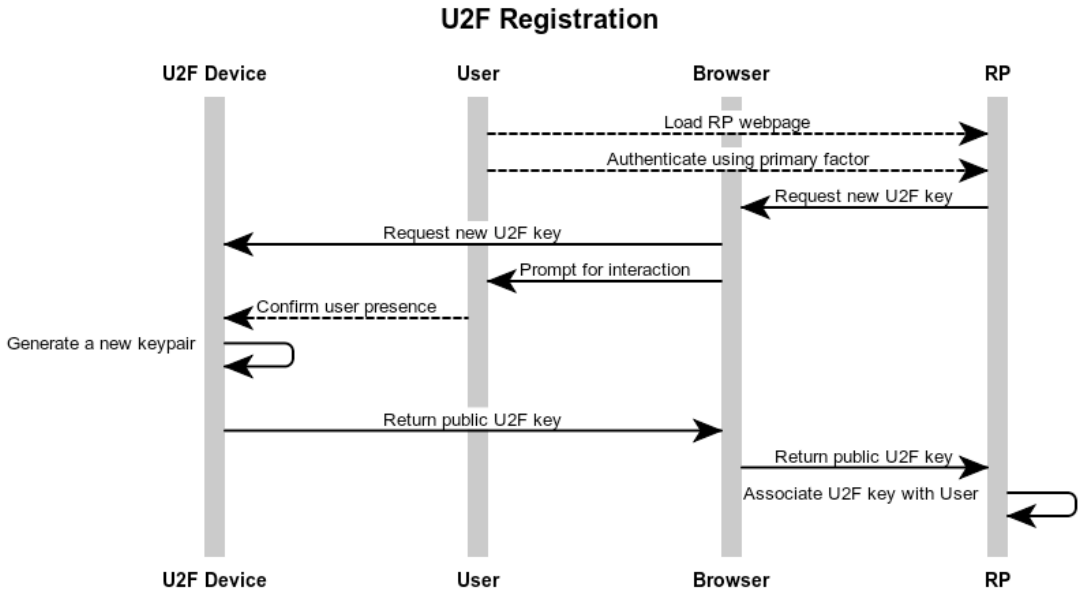
U2F is a technology that allows different manufacturers to produce devices in different form factors, which can be used for second-factor authentication with a variety of platforms using the same technical API. Each device is able to generate a set of keys that can identify the device to U2F-enabled software. After one of the device's public keys has been registered, a U2F relying party (RP) can send a challenge to the U2F device, asking it to sign something to prove possession of the corresponding private key.



This signature is verified against the previously registered public key. If the signatures match, this is taken as proof that the U2F device was present during the transaction.

The U2F protocol also allows the RP to request user interaction, such as a physical button press, during the transaction. If this request is made, the U2F device can refuse to sign the challenge until user interaction takes place. The user interaction has been explicitly designed to be trivial to operate and therefore not cause undue consternation on the part of end users. While this interaction can show that a person is likely present during the transaction (as opposed to a scripted piece of malware), it cannot identify that person.

How does the relying party know which key to verify against? In U2F, the device creates a new key pair upon request and sends the public key to the relying party during an explicit registration process.



This approach allows the U2F device to generate and store a separate key pair for each RP that requests one. The public keys themselves cannot be correlated with each other by colluding RPs, offering enhanced privacy to users. This registration also takes place in a context where the user's identity has already been authenticated to some level, thereby binding the newly generated public key to that user's identity.

Comparison with X.509-style PKI

In many ways, the U2F process is similar to how public key infrastructure (PKI) certificates such as X.509 have been used in login systems for a very long time: a challenge is presented, the challenge is signed, and the signature is verified from a registered key. However, there are a few very important differences.

First, in traditional PKI, trust of the certificate's keys is predicated upon trust of the certificate authority that signed those keys. The certificate authority's keys are often signed in turn by another, higher certificate authority, all the way to a set of root authorities, which must be pre-configured in clients as trusted. In contrast with U2F, the explicit registration step obviates the need for a white-listed trust root. This allows for new keys to be generated as often as they are needed, in contrast to a signed certificate, which is a heavyweight cryptographic device that is comparatively expensive to produce, protect, and re-issue. As a consequence, a U2F device is able to generate a different key pair for each RP instead of reusing the same keys with every site. This is in contrast to certificate-based PKIs, in which the same certificate and key pair is used to represent the user at all RPs.

U2F's key mechanism effectively uses the public key portion of PKI without the attendant infrastructure. The assurances and functionality granted by the centralized authority of a PKI system can now be distributed at enormous scale. This distributed approach also enhances

privacy by removing the possibility of two colluding RPs linking a user across sessions using their public key alone, since the public key will be different at each RP with no means of tying keys together.

U2F does have a trust chain similar to the certificate authorities found in traditional PKI, but this is not tied directly to the key pairs issued by the U2F device. Instead, this trust chain is tied the device's identifier certificates. These device certificates are used alongside the ephemeral keys to identify the device itself (or a batch of devices), allowing knowledgeable RPs to make informed decisions about which device manufacturers they are willing to accept.

What U2F cannot do

U2F is explicitly unable to provide information about the user's identity. In fact, a U2F device has no concept of a user at all. All that an RP can ever know from a U2F transaction is that something with access to the appropriate private key (i.e., the U2F device) exists on the far side of the transaction. This design provides strong privacy and security, including the ability to seamlessly share a device among multiple accounts, but it has some major drawbacks when used with many kinds of real-world applications. Particularly, most RPs need to know attributes about the user, such as what to call them or where to email them. Additionally, many RPs are interested in these identity attributes being strongly proofed before accepting them. Since there is no inherent binding of the user to the U2F device, the device's presence alone is not enough to satisfy this kind of requirement.

This disconnect is a second way in which U2F differs from traditional PKI authentication. In X.509, user attributes are usually included in the payload of the certificate and presented as part of the authentication transaction. On the surface, this solves the problem of identity and authentication in one go, but it introduces several new problems. First, by having the attributes inside the certificate, they cannot be altered or extended without re-issuing the certificate as a whole. The attribute bundle also can only be sent as a single immutable set: there is no way to limit the disclosure of some attributes in a particular transaction without issuing a special certificate with just those attributes. For instance, the US Government's Common Access Card is issued with two certificates, one with an email address and one without. This essentially couples key rotation and attribute updates to each other in the same expensive process. Furthermore, since the certificate authority signs the attributes themselves, there's an implied attestation by that authority in the presentation of the certificate. This has perhaps unintentionally lead to certificate authorities being unduly focused on identity proofing, raising the bar of access unnecessarily and unintentionally limiting the classes of users who can benefit from this kind of security. Finally, since the user presents the same certificate at every site, colluding RPs can track a single user across the internet in ways that are not immediately apparent to the user.

By avoiding user identification entirely, the U2F specification avoided many of the problems associated with X.509 certificate attributes. However, the desire to use strong key-based credentials alongside bundled identity attributes remains. How then can we solve this? Fortunately, U2F can fit nicely alongside of other technologies that solve these other problems, which we will discuss in a later section of this paper.

Identifying the device, not the person

There are real use cases for sharing a U2F device, since the technology offers a very good set of security properties. Since there are no user-identifying components in the key sets or in the ceremony of using the key, a single device can suffice for multiple users, such as workers rotating through a single computer terminal or multiple members of a family. So long as this factor is used in concert with other components to provide a full authentication stack, this sharing isn't problematic since access to the keys alone is never enough to completely access an account.

Traditional PKI systems generally rejected the premise of shared keys and overcame this perceived limitation by requiring PINs or other knowledge-based authentication mechanism to unlock the private keys in a certificate store. This approach adds a "something you know" component to the "something you have" of the certificate file itself. However, the added security benefit is only realized if the PIN was entered interactively for the transaction. The credential caching mechanisms found nearly universally in X.509 certificate managers do away with this ceremony and therefore one of the authentication factors.

Why would such caching systems be widely used when they clearly subvert a fundamental aspect of the security components? A system that constantly prompts a user for the same PIN again and again is likely to be ignored or rejected by users annoyed at the constant prompting. The use of a credential cache is often considered a reasonable tradeoff. However, the U2F design avoids having to make this tradeoff decision in the first place by explicitly declaring that the ephemeral keys are used to identify the device alone.

OpenID Connect

If U2F doesn't tell you anything about the user, their attributes, or their affiliations, is there a protocol that does? After all, many RPs will want to know this information in order to provide a good user experience.

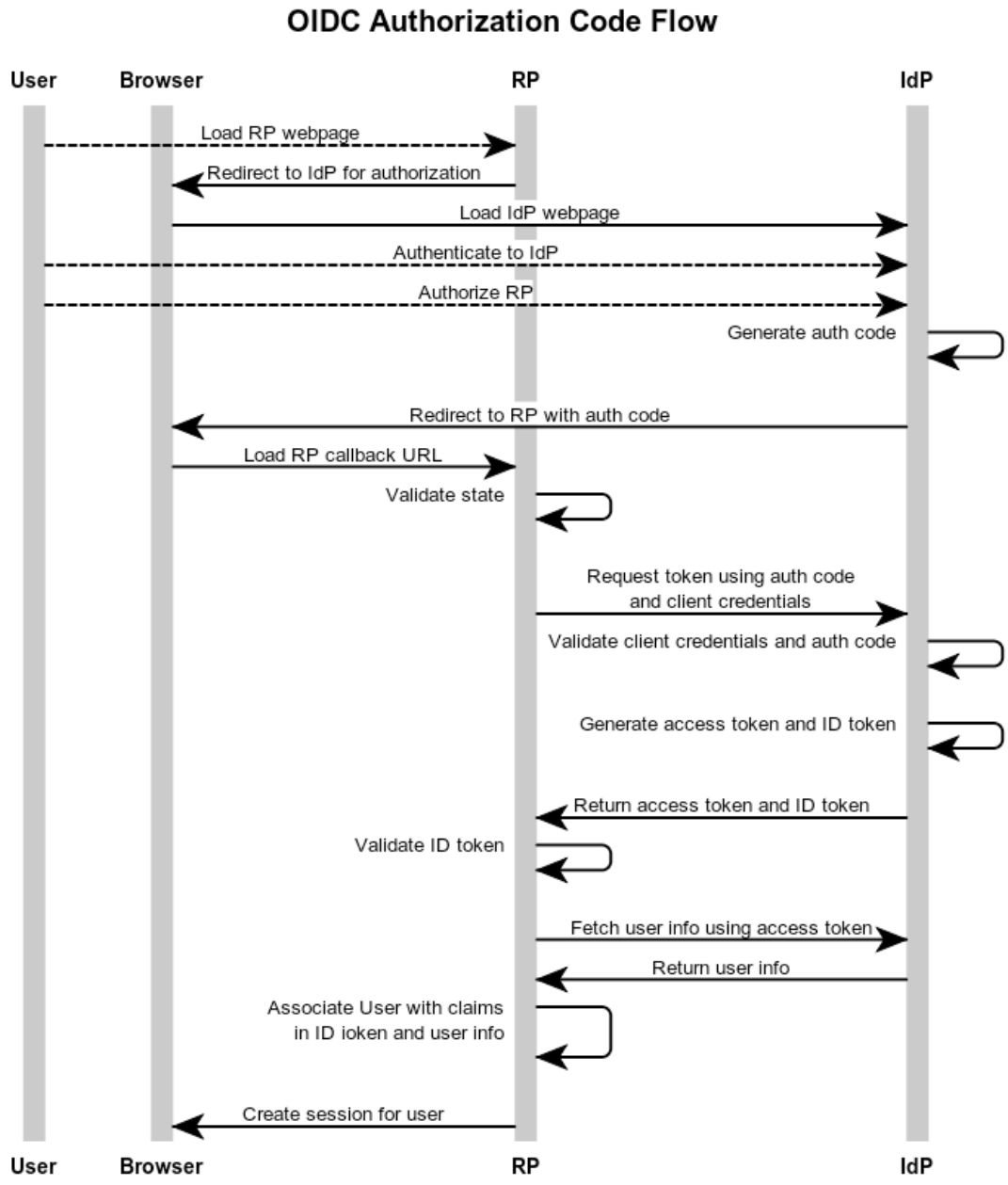
OpenID Connect (OIDC) is an identity federation protocol published by the OpenID Foundation. OIDC is built on top of the OAuth 2.0 delegation protocol and provides a standard identity API that is distributable at internet scale. By making use of OAuth as well as the JSON Object Signing and Encryption (JOSE) suite of specifications, OIDC provides a relatively easy-to-implement identity protocol that fits well alongside other web APIs.

OIDC provides information about the current user, from how they have authenticated at the IdP to attributes bound to them. OIDC can function in a distributed fashion at internet scale, and it is as at home in the enterprise as it is on the social web.

What OIDC can do

In an OIDC transaction, the RP requests authorization from a user in order to access authentication and identity information from the user's identity provider (IdP). This process is based on the OAuth 2.0 authorization protocol, which is in wide use on the internet today.

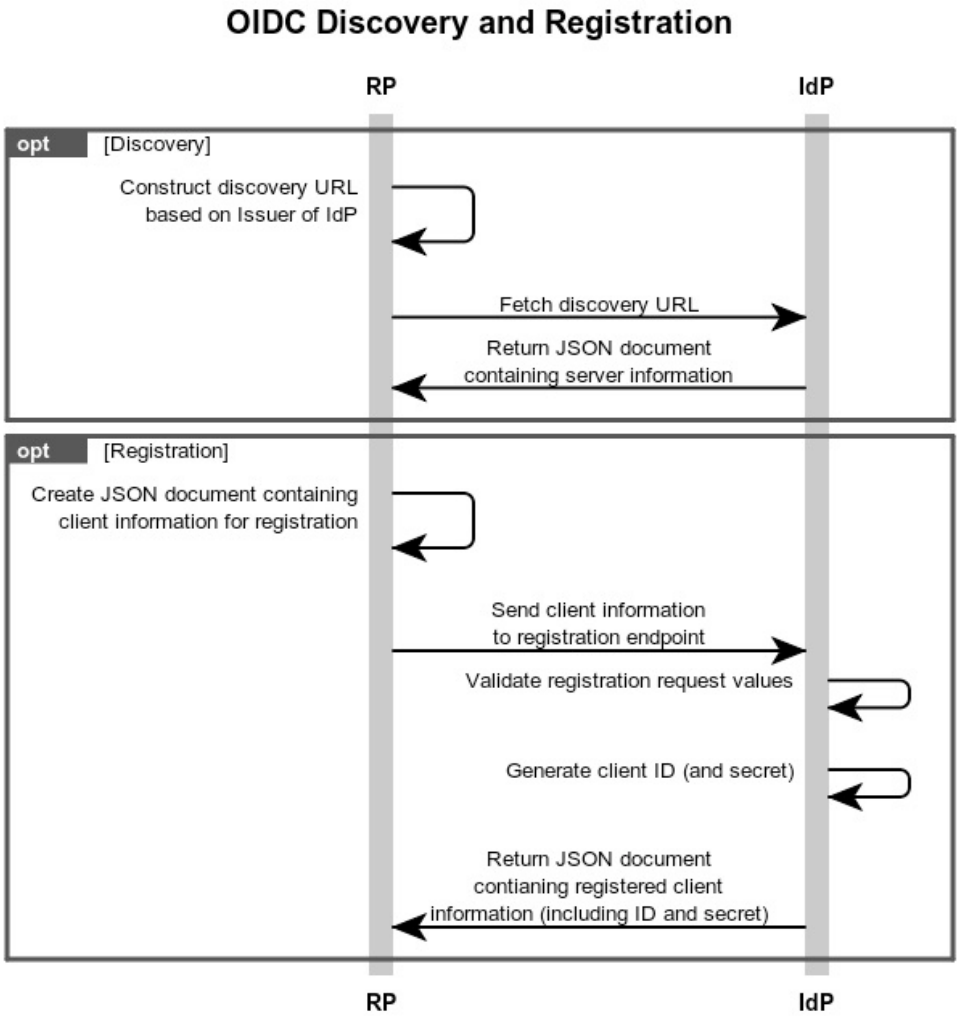
In the most common flow, the authorization code flow, the RP redirects the user to the IdP with a specialized request. The user then logs into the IdP directly, using whatever credentials the IdP requires, and authorizes the RP to access identity information. The user is then redirected back to the RP with an authorization code. This code is then presented in the back end to the IdP along with the RP's own credentials to get a set of tokens which can be used to identify the user. The interaction between the parties looks like this:



The RP is given an ID Token that contains a set of information about the user and the surrounding authentication event. This token is given to the RP directly by the IdP, and its signature can be cryptographically verified using the IdP's published public key. All of the communication between systems is protected by TLS in addition to the signatures (and optional message encryption) of various components, but the keys used in the identity transactions (such as signing the ID Token) are separate entities.

In an OIDC transaction, each party knows only the minimum amount of information required for it to process its part of the transaction. This ensures that the possible damage from an attacker compromising any part of the transaction is minimized and cascading failures are prevented.

How do the relying party and identity provider learn about each other? In older federated identity systems like those based on the Security Assertion Markup Language (SAML) and most PKI systems, the relationship between the IdP and the RP is set up manually with an out-of-band configuration. While this manual process can also be done with OIDC, it provides a dynamic server discovery and client registration protocol set to allow it to scale to the internet. The user needs to remember only an identifier like their email address to start the introduction process. These steps take place the first time two services are introduced to each other:



Once the two components have been introduced, subsequent transactions can be simplified using the IdP and RP configuration information gathered in these steps. If this cached information ever fails, the RP is able to repeat the registration and discovery process at any point.

OIDC is built from simple components that can be combined and extended to solve a wide range of use cases. A guiding principle of the OIDC working group was “make the simple things simple, and the complex things possible,” This goal is apparent in the resulting OIDC protocol. For instance, the base OAuth protocol uses simple HTTP redirects with components in the query parameters, and the OIDC protocol can use this as well. However, OIDC also provides a mechanism for RPs to send requests to the IdP using a JOSE container and taking advantage of the resulting signing and encryption capabilities.

Since an OIDC server is also by definition an OAuth 2.0 server, it can provide access delegation to an unlimited set of protected user APIs. Differentiation between different APIs is determined by the existing OAuth 2.0 “scope” mechanism. OIDC defines a set of five scopes that map to the identity API it provides, but other scopes can be mixed with these at runtime to provide access to other APIs.

Using OIDC, the user is able to control which sites receive their identity information, often with the ability to decide which attributes are sent to which RPs. This focus on user choice is in stark contrast to SAML, where attribute bundles are decided at configuration time between system administrators, and PKI systems, where all attributes are bundled into the certificate. In contrast, OIDC is flexible enough in its deployments to allow for previously-vetted sets of attributes to be released to trusted parties through the use of simple whitelists or for end users to have full control over which attributes are released to whom, all within the same protocol and deployment.

Since OIDC is a web-friendly and network-based protocol, the primary authentication by the user at the IdP is extraordinarily flexible. This means that the user is able to make use of a variety of different authentication mechanisms when presenting to the IdP, and that these authentication mechanisms can be swapped out over time. This flexibility allows new technologies to be adopted and their benefits enjoyed seamlessly by the entire network. The IdP can provide a number of additional security services for end users, such as monitoring account usage and applying security heuristics. For instance, if an account is used from a new geographic location at a new site, the IdP is in a position to notice this change of behavior and require higher authentication from the end user.

What OIDC cannot do

One of the biggest perceived downsides to any federation protocol, OIDC included, is that the relying party is at the mercy of the identity provider. From the perspective of the RP, the IdP’s own authentication mechanisms are opaque. While this can often frighten RP operators, the IdP is generally in a much better position to identify the current user. Still, the RP can trust this only as much as it trusts the IdP. There is no standard way for an RP to verify an account locally in addition to receiving the assertion from the IdP.

While the flexibility of federated identity allows any form of primary authentication to be used, many RPs desire to enforce and verify that a user’s primary credential meets a certain set of minimal criteria. OIDC provides mechanisms for requesting certain authentication contexts,

authentication times, and other authentication criteria. These are all communicated back to the RP using the ID Token's claims mechanism. However, the RP is once again at the mercy of the IdP's claims.

OIDC on its own does not provide a means to determine the trustworthiness of any IdP or RP in a given system. However, OIDC's assertion mechanisms can be anchored to policies and contracts that provide this necessary information. The standards to define these anchors are currently being established and will provide a more robust framework for trusted systems in the future.

Like many identity technologies, OIDC provides attestation of an authentication event. However, a user's interaction with an RP will generally continue over many actions across time. At the end of the authentication transaction, the local session management of the RP generally takes over, binding the remotely authenticated federated user to a particular local account in the RP. The session management extensions of OIDC provide ways for an RP to determine if a user still has an active session at an IdP, though at the time of this writing these specifications are not final and their deployment and availability is limited.

Internet scale and user choice

An important aspect of the OIDC protocol is its ability to work across security domains by design. Unlike the single-provider identity solutions that have traditionally been deployed in large-scale enterprises, the open protocol stack of OIDC encourages multiple independent, interoperable implementations. Its flexibility allows the same software stack to be used with lightweight "social" identities as well as heavyweight high-assurance identities.

In some IdP implementations, the user can select which attributes they wish to be provided to a given RP, often with minimal effort. This flexibility allows users to have full control over what is shared while still providing an easy-to-use, low-friction path for the most common use cases. These decisions are generally cached after the user makes the initial introduction, a method known as "Trust on First Use" or TOFU. In this way, the user experience is streamlined by anticipating that a user will make the same authorization decision as last time. The user can also audit and revoke previously made decisions, causing them to be prompted again next time. The IdP has the opportunity to use other security mechanisms, such as heuristic analysis of the user's session at the time of the authorization request, to determine whether or not the user should be re-prompted even though a decision has previously been made.

Accepting a remote federated identity gets RPs out of the business of managing credentials themselves. By outsourcing identity and primary credentialing to an IdP, an RP can limit the kinds of attacks to which it is susceptible. For instance, many sites have been compromised by passwords stolen at one site being replayed at another. With a federated identity, however, such as provided by OIDC, there is no password that can be replayed. A credential stolen from one compromised RP cannot be used at another RP, since the OIDC ID Token is audience-restricted to the RP itself.

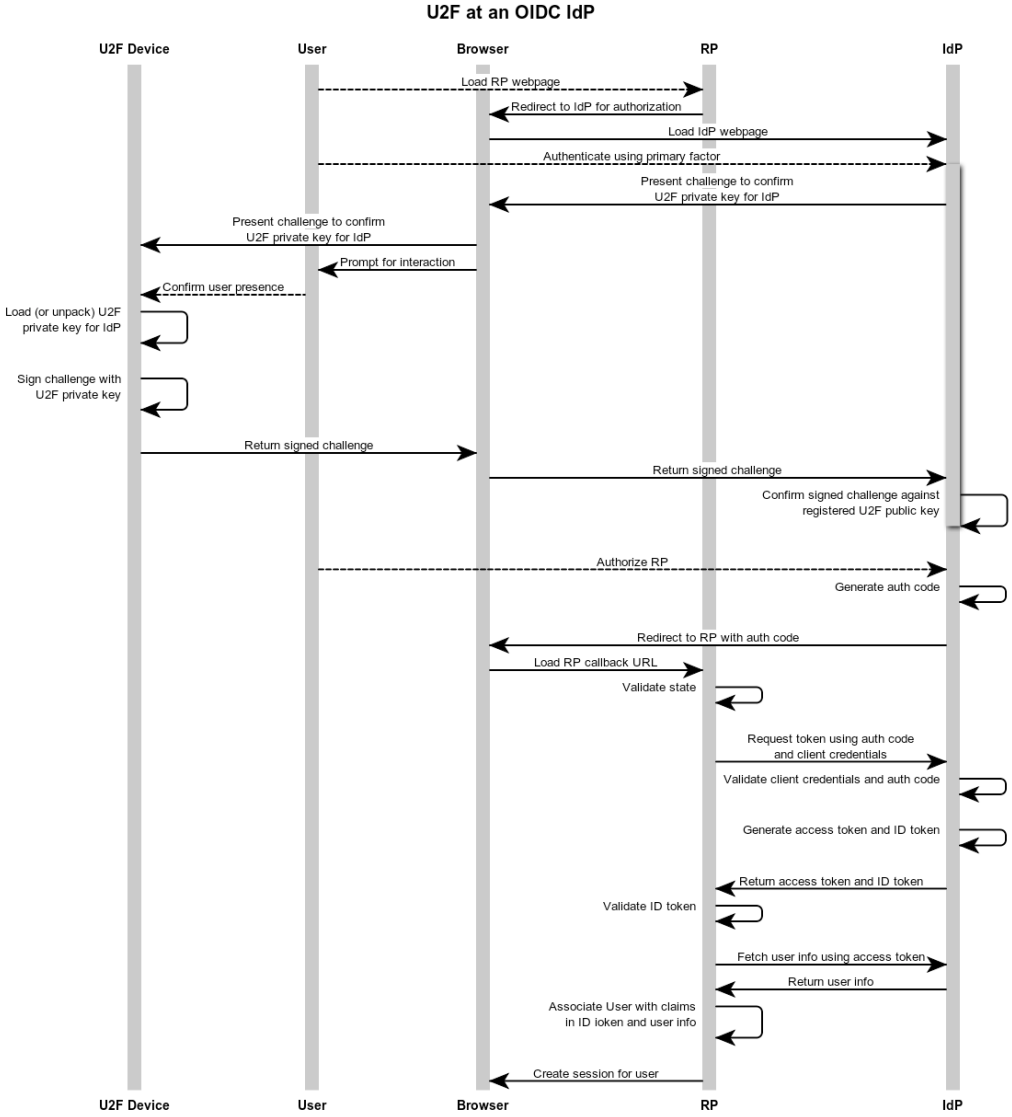
Users are also given the ability to keep their attributes, both self-asserted and proofed by a trusted authority, at a trusted identity provider. This setup allows a user to set an attribute once (or have it proofed once) and later have that attribute attested to a number of different relying parties.

Combining U2F and OIDC

U2F and OIDC are two very different protocols that solve complementary problems. These problems are often found together in the wild. Fortunately, there are a number of potential methods to combine the two technologies.

Using multiple factors at the IdP

A simple means of combining these two technologies is to apply the U2F factor as part of the login process at the IdP. Since the OIDC protocol leaves the primary credentialing mechanism deliberately undefined, it can make use of a large number of different technologies. A user's account at an IdP can be backed by several factors including a U2F device. These factors can even be applied dynamically, where the IdP can decide which authentication transactions require the presence of a U2F device for authentication and which do not, perhaps at the behest of the end user's configuration or the RP's request.

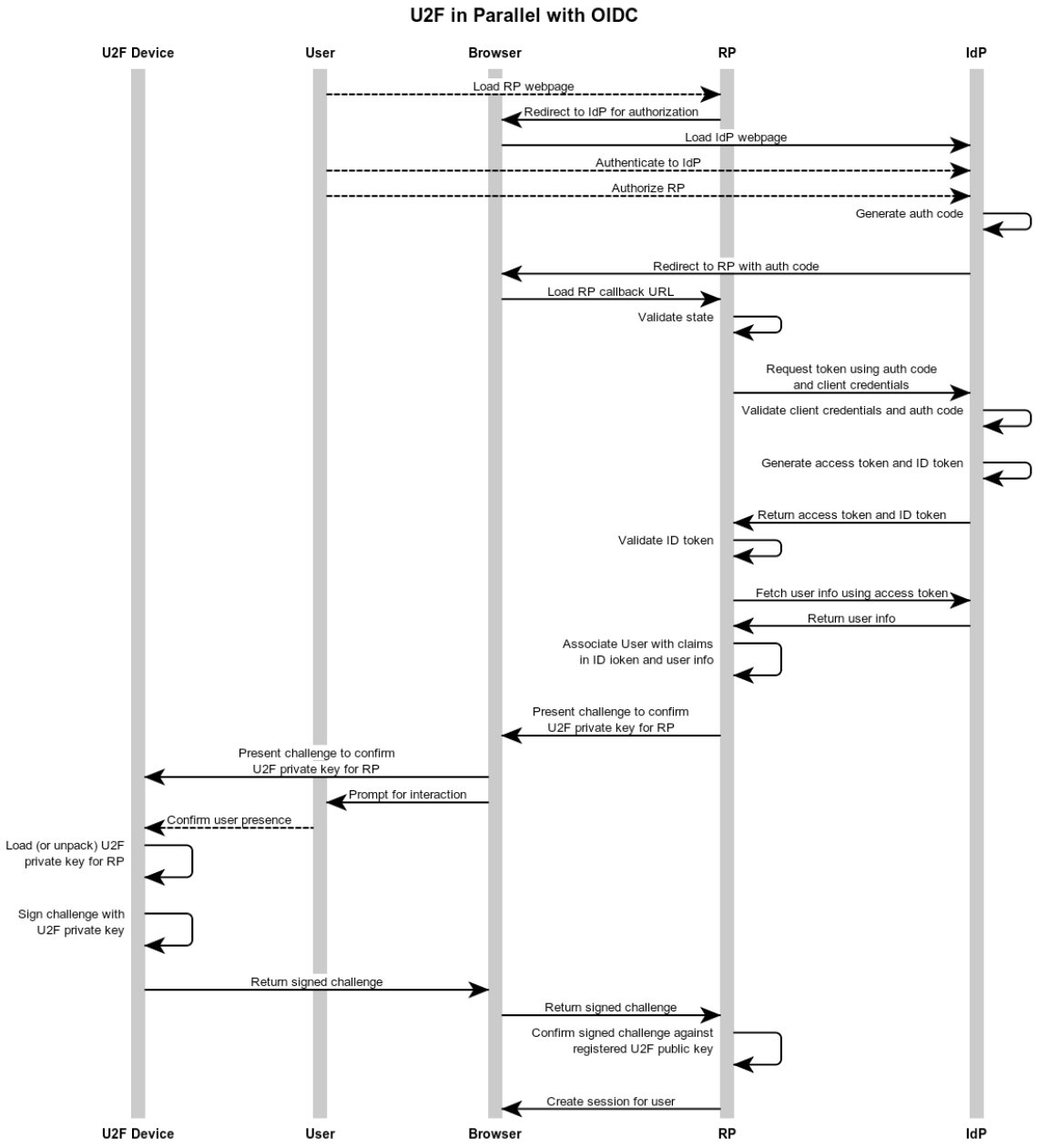


This method is mostly transparent to the RP, and its presence can be easily ignored by the RP if it so desires. The RP can request multifactor authentication through use of the “authentication context reference” and “authentication method reference” mechanisms of OIDC, and it can be told about the primary authentication that was used in the ID Token response, but otherwise the RP can simply speak the OIDC protocol and gain the benefits of U2F without having to do any direct integration. This also allows the IdP to combine the U2F protocol with any number of other primary authentication technologies and security techniques.

Furthermore, it’s recommended in OIDC (and the underlying OAuth protocol) that the client application use the system browser, as opposed to an embedded web view. This allows the user to take advantage of existing sessions and preferences, but, most important, it also takes advantage of extended functionality such as a U2F agent. Therefore, the user’s interaction with the IdP is familiar and trusted, even in the context of a U2F session.

Using U2F in parallel to federated login

A more complex method of combining OIDC and U2F is to use them in parallel at a given RP. In this model, when a user creates an account at an RP, the RP prompts them for both a federated identity as well as a U2F device registration. Both of these credentials can be presented in parallel within the same session, strongly binding the two of them to the same user at the RP. When the user logs in or tries to perform subsequent actions on the RP, the RP can decide whether either or both of these are needed to perform a sufficient login transaction.



For instance, perhaps the federated login would suffice to authenticate the user for the majority of actions available on an RP, but when a high-value action is requested by the user, the presence of their trusted (and registered) U2F device is confirmed through the use of the U2F

protocol. An attacker would then need to gain access to both the remote IdP account as well as the U2F device in order to impersonate the user at that RP, since compromise of either would not grant sufficient access. This comes at the cost of higher complexity for the RP, which now has to implement both the OIDC and U2F stacks, as well as potentially poor user experience in managing the binding of both sets of credentials across a wide range of RP sites.

This technique is particularly useful in the context of mobile applications. The use of a mobile-friendly identity federation protocol like OIDC can identify the user to a device and bind their attributes to it. The OIDC tokens can be used across applications on the device to uniquely identify the user and provide access to web API services on behalf of that user. The U2F protocol can be used locally on the mobile device to provide a higher assurance for specific actions: now an attacker needs to possess the U2F device in addition to the mobile device itself.

Potential future work

There are several other mechanisms by which U2F and OIDC could work together, but the current state of the standards or their implementations does not yet allow for them. However, with active extension work taking place on both of these protocol stacks, there are a few possible future implementations.

With Vectors of Trust and Trustmarks

A Vector of Trust (VoT) is a mechanism for conveying the independent components of an authentication transaction, including identity proofing, credential strength, and assertion presentation. This multi-part value is intended to communicate trust attributes of an authentication transaction in place of the more traditional single-value “Level of Assurance”.

A Trustmark is a verifiable attribute that can be asserted by IdPs, RPs, and other components in a network ecosystem. The presence of a valid Trustmark indicates that a trusted third party has certified aspects of the component’s operation, such as the processes used to onboard new users at an IdP or the privacy practices in place at an RP. This Trustmark binds the component in question to a set of legal and business rules which can be used to make trust and authorization decisions.

The VoT value can be carried inside OIDC’s ID Token, and the credential strength component of the VoT can be used to indicate the presence and usage of the U2F device at the IdP. This allows the RP to request and verify that the U2F device be interactively verified at the IdP for a specific login transaction.

The Trustmark can be used to advertise the certified proper use of multiple factors at IdPs and RPs on an OIDC transaction. Where the VoT can claim that a specific practice was used, the Trustmark can show that the party in question has proved their ability to use it. For instance, when the OIDC ID Token contains a VoT claim that states that an interactive U2F device was used to authenticate the user, the RP has no way to know if this actually took place or if the IdP is lying about the state of the transaction. With a Trustmark, the RP can at least verify that the IdP has the capability of using U2F and has demonstrated proper implementation and enforcement of the U2F protocol.

A Trustmark can also certify the U2F devices themselves. Each U2F device has the possibility of carrying a manufacturer-supplied certificate in addition to the ephemeral keys generated on the device itself. Instead of being tied to a certificate authority, this device certificate could be tied into a Trustmark that would indicate that the manufacturer follows secure processes and protects the device creation process sufficiently that the key in the device can be trusted. This would allow U2F RPs to more easily decide which classes of U2F devices to accept.

The standards for VoT and Trustmarks are in their early days at the time of this writing, but there is active development and wide community engagement around both efforts.

As a device identifier for OAuth2 clients

The U2F key mechanism could potentially be used as part of the underlying OAuth 2.0 protocol at the core of OIDC. If a U2F device is strongly tied to a client device, such as something

embedded in the device's firmware, its public key could potentially be registered by the client software instance as its own key. The corresponding private keys would then be used to sign an assertion to authenticate the device itself, not the user, to the authorization server at the time of token issuance.

Currently, however, the U2F protocol is optimized for a single authentication transaction with a browser as an intermediary. In order for it to be useful as a general device identification mechanism, the key verification process would have to be accessible outside of the browser. Thankfully, the U2F protocol allows the U2F client to be a non-browser application, though at the moment no such general-purpose application exists. If U2F's signing mechanisms can be extended such that they will work on an arbitrary JavaScript Object Notation (JSON) Web Token (JWT) payload, then these keys could use the advanced capabilities of OIDC, such as signed request objects.

Conclusions

The U2F and OIDC protocols are both authentication technologies, but they authenticate different things in different ways. U2F authenticates the presence of a device with access to the right keys. OIDC on the other hand authenticates the holder of an account at an IdP. U2F thinks in terms of devices and key pairs, while OIDC thinks in terms of users and non-physical tokens.

Strong authentication and internet-scale identity federation go hand-in-hand. When the two of these are used correctly in tandem, they can provide a powerful multi-factor federated authentication mechanism that is secure, usable, deployable, and functional.