Mobile Personal Identity Provider Based on OpenID Connect

Luigi Lo Iacono[1], Nils Gruschka[2(✉)], and Peter Nehren[1]

[1] Cologne University of Applied Sciences, Betzdorfer Str. 2, 50679 Cologne, Germany
{luigi.lo_iacono,peter.nehreng}@th-koeln.de

[2] Kiel University of Applied Sciences, Grenzstr. 5, 24149 Kiel, Germany
nils.gruschka@fh-kiel.de

Abstract. In our digital society managing identities and according access credentials is as painful as needed. This is mainly due to the demand for a unique password for each service a user makes use of. Various approaches have been proposed for solving this issue amongst which Identity Provider (IDP) based systems gained most traction for Web services. An obvious disadvantage of these IDPs is, however, the level of trust a user requires to place into them. After all, an IDP stores a lot of sensitive information about its users and is able to impersonate each of them.

In the present paper we therefore propose an architecture that enables to operate a personal IDP (PIDP) on a mobile device owned by the user. To evaluate the properties of our introduced mobile PIDP (MoPIDP) we analyzed it by means of a prototype. Our MoPIDP architecture provides clear advantages in comparison to classical IDP approaches in terms of required trust and common threats like phishing and additionally regarding the usability for the end user.

1 Introduction

In the last decade, the Web has become an essential part of human society. Nearly every aspect of our daily life is infused by Web services (e.g. news reading, travel booking, shopping etc.) and in some areas the digital services have partly or completely eliminated the traditional services by now. In the recent years, this trend was further pushed with the ubiquitous usage of Internet service on mobile devices. With the increasing number of services used also the number of user accounts is rising, which leads to one of the main open issues of digital services: identity management [20].

Identity management (IDM) typically includes two aspects. The first is authentication, a basic security control required in any application that interacts with a user. The user claims his identity by presenting an identifier and then giving some proof for this identity. The proof can be realized in many different forms (e.g. fingerprint, cryptographic protocol), but the most widespread authentication factor by far is the password. Password authentication can be easily implemented by a service and does not require any special software or

J. Lopez et al. (Eds.): TrustBus 2017, LNCS 10442, pp. 19–31, 2017.
DOI: 10.1007/978-3-319-64483-7_2

hardware on the user side. However, passwords are prone to many attacks and misuses. Further, password authentication burdens the user the task of password management, leading very often to simple passwords or reusing the same password for different services. Another problem of authentication is the linkability of identities across distinct services. Most services require a valid email address as identifier, leading the user to use the same identifier with different services.

The second aspect of identity management is management of attributes. For performing its service, the service provider needs a number of attributes from the user (e.g. address, birthday, credit card number). Nowadays, a service asks the user during the registration to enter all attributes which the service might need eventually. This creates some privacy problems. First, most services request more attributes than necessary. Second, the user has to update attribute changes (e.g. after moving houses) at multiple places. Finally, the distributed storage increases the danger of unwanted attribute disclosure.

IDM systems aim to solve the issues presented before. The typical IDM architecture includes a third party, the so-called Identity Provider (IDP). The user authenticates directly only to the IDP. The IDP than creates authentication assertions which can be used to authenticate to services (also called *relying party* (RP) in this context), this requires obviously a trust relationship between the service and the IDP. Now the user only needs authentication credentials to the IDP, i.e., in the case of password authentication, only one (complex and unique) password. This enables also a single sign-on (SSO) functionality. Once the user has logged into the IDP he can access multiple services without entering her credentials again even when provided by distinct service providers.

Further, the IDP can also operate as attribute provider. In that case, services are not supposed to store user attributes locally, but request attributes on demand at the IDP and get the current values (if the user has granted access to the attribute). To increase privacy, the assertions given to the service might even masquerade confidential attributes. For example: if a service requires legal age of costumers, instead of sending the birth date the IDP can send an assertion "is older than 21 years". Finally, the IDP can create assertions for authorization delegation, e.g. authorizing service A to access the users pictures stored at service B.

Despite the mentioned benefits of IDM systems, some problems remain. First, the user must completely trust the IDP, as it can impersonate the user to the services. This can be solved by operating a personal identity provider. In this paper, we present a personal IDP, which can be self hosted on the user's mobile device. Second, most existing IDM systems are not very widespread and very often are discontinued after a while. In order to ease the distribution, our system is based on OAuth and OpenID Connect, two protocols which are already used by a number of large Internet companies. Further, a self hosted IDP usually induces the problem of discovering it. We overcame this by triggering the OpenID protocol flows from the IDP (instead of from the service). This solves finally also the problem of phishing attack, i.e. redirecting the user to a spoofed IDP site and phishing the user's IDP password.

The paper is organized as follows: the next section gives an overview of the related work on authentication and identity management. Section 3 presents our solution for a Mobile Personal Identity Provider. In Sect. 4 the security and privacy properties of our solution is discussed and compared to other solutions. Finally, Sect. 5 concludes the paper.

2 Related Work

2.1 Authentication and Identity Management

Since more than 20 years efforts are undertaken to replace the simple (static) password as authentication method to Internet and especially Web services. One possibility is using different passwords for every authentication action, named *one-time passwords (OTP)*, e.g. the well known S/Key from Leslie Lamport [15]. However, OTP requires either manual password lists for every service or special hardware tokens, which is unhandy or rather expensive. A further possibility are authentication methods based on cryptographic protocol, e.g. TLS client authentication [9]. However, this typically requires complex management of cryptographic credentials at the user side (e.g. transferring private keys from one computer to another one). Further, biometric attributes (e.g. fingerprint, face, voice) can be used for user recognition and authentication [17]. However, until a few years ago, this required special hardware attached to the computer.

With the wide spread of mobile phones the situation has changed. Mobile devices can act as credential storage, password generator and even as biometric recognition sensor. Thus, it comes as little surprise, that a number of authentication systems have emerged, which use a mobile phone as primary or secondary authentication factor. Examples are the FIDO UAF system [2], which combine strong local authentication at the smart phone (e.g. fingerprint recognition) with cryptographic authentication protocols, or Google Authenticator [13], a mobile application for creating OTPs with limited lifetime [22]. However, these systems only support the authentication aspect of identity management (IDM), lacking authorization, delegation and attribute management.

The most known IDM protocols are SAML, OpenID and OAuth. The Security Assertion Markup Language (SAML) [19] is a full-fledged IDM system, offering data formats and communication patterns for all IDM functionalities presented in the previous section. However, due to the high complexity, SAML was only disseminated in special areas (e.g. as Shibboleth [21] in higher education facilities or in SOA environments [25]).

OpenID [12] is a simpler identity provider (IDP) based authentication protocol specially for Web use cases. OpenID also did not find the broad adoption its creators have hoped for. Finally, OAuth [16] is a protocol especially for authorization delegation. OAuth was widely adopted for API authorization e.g. at Google [14], Facebook [10], Twitter [26] etc. As authorization requires prior authentication, some API provider "misused" OAuth for pseudo-authentication to avoid the additional implementation of OpenID, leading to proprietary OpenID extensions. To overcome this issue, OpenID Connect [23] was proposed, extending the OAuth protocol and workflow with OpenID authentication.

Some security and usability problems remain also in OpenID Connect which where known from OpenID [8]: (1) The user has to trust the IDP with his data. (2) When redirected to the IDP for authentication, the user has to check carefully, that it is really the IDP and not a phishing site. (3) The user has to remember and enter her OpenID URI at the service provider.

To overcome problem (1) the user can host her own IDP, also called a *Personal IDP (PIDP)*. Further, problem (2) can at least be mitigated, if the PIDP is running of the users mobile devices. In this case the user is not interacting with a (probably spoofed) Web site, but with a local application on her mobile phone. The idea of a personal mobile is not new, examples are can be found in [11] or [1]. However, these systems have been developed for the outdated OpenID 2.0 and additionally still lack from problem (3).

This paper introduces a mobile personal identity provider (MoPIDP) which eliminates the above mentioned problems. First of all, it is—to the best of our knowledge—the first PIDP based on the current OpenID Connect standard. Also, as a personal IDP, impersonation by the IDP is not a problem. Further, we invert the flow control in the beginning of the protocol: instead of redirecting the user from the RP to IDP, the MoPIDP gets the required information for the first protocol step via a barcode from the RP and the MoPIDP contacts the RP. This eliminates the phishing problem. Further, this increases the usability of the overall process, as the user does not need to remember and enter her IDP's URI.

2.2 OpenID Connect

To illustrate the differences to our MoPIDP architecture we present how OpenID Connect works. In this paper we will solely regard the *Authorization Code Flow*, as other OpenID flows have been proven to be insecure [24].

Before a service application can use OpenID authentication with a certain IDP, it is required to register the service at the IDP. The standard way for this is that developers register their application once at the IDP out of band. As part of this process, the application gets a `client_id` and a `client_secret` to make authenticated requests to the IDP.

Once this step is completed, OpenID Connect Authentication like shown in Fig. 1 is possible. Here, the user (using a Web browser) accesses a restricted resource at the service, which requires authentication. The user enters her OpenID URI and the service redirects the user to the appropriate IDP. The redirect contains an authentication request, which holds a number of parameters including the service's `client_id` and a redirect url for returning to the service. The user has to authenticate to the IDP. This can be done in different ways, e.g. by user name and password. If the user has authenticated before and still has an active session with the IDP, this step is omitted. Then the user is redirected to the service (using the redirect URL from the authentication request).

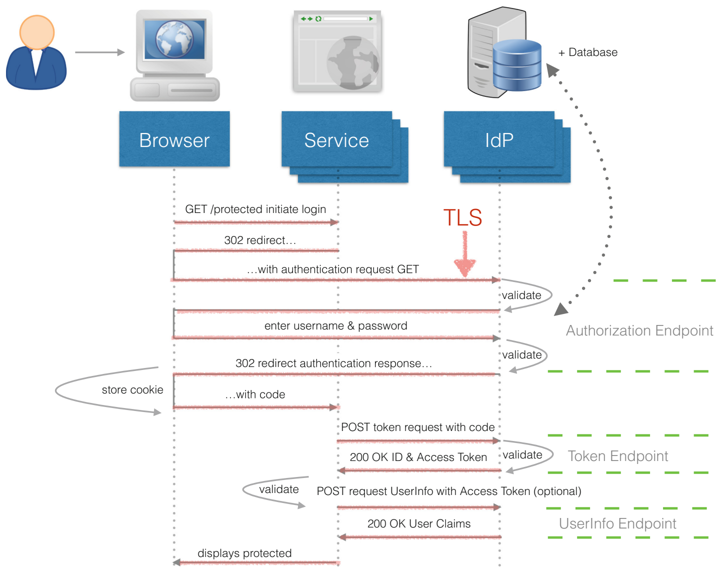


Fig. 1. OpenID connect authentication

The redirect to the service contains the authentication response, which again contains a `code` parameter, the authorization code.

This authorization code can be used by the service in a token request to the IDP. If the code is correct, the IDP returns an access token[1] and an ID token. The access token can be used to access further services, e.g. requesting detailed user info or (which is the standard OAuth use case; not shown in the figure) invoking an external service on behalf of the user. The ID token is the most relevant extension in OpenID connect compared to OAuth 2.0. It contains information on the authenticated user like a subject identifier and an expiration time stamp.

Now finally, the service returns the restricted resource to the user's Web client.

3 Mobile Personal Identity Provider

3.1 Requirements and Preconditions

From the aspects discussed in the previous section we derive the following requirements for our Mobile Personal Identity Provider (MoPIDP):

Self-control. Only users themselves have full control over the MoPIDP and no other third party is involved into the whole process. This control contains all kind of information stored securely inside the application and who gets

[1] Until here the protocol is identical to OAuth 2.0.

access to it. The attributes are only stored inside the users MoPIDP. The only exception are encrypted exports for backup purposes.

Convenience. It offers an easy and secure way for login and registration on Web services via a smartphone application. Users are not forced to enter any kind of credential to the computer keyboard.

Profile Management. Users can add multiple profiles respectively digital identities for different areas of application. For that reason these profiles are connected to other kinds of attributes.

One important prerequisite for a mobile service is the addressability of the smartphone over the Internet, even if it is connected over WiFi or mobile networks. In future this could be handled with IPv6 causing its bigger address room where all devices could have their own IP address. Nowadays a so called reverse proxy could be used by which Internet accessibility of local servers running on any kind of computers or devices can be provided.

3.2 Overview

The architecture of the MoPIDP is based on OpenID Connect. However, due to the requirements stated before the architecture has to be modified slightly. In OpenID Connect an IDP is always on-line, hosted at the same domain and accessible every time. The MoPIDP is always with the user and just on-line if the user wants to make use of it. Another difference is, that an normal MoPIDP is responsible for multiple users. In the new system every user owns his own personal MoPIDP.

As every user has her own IDP, a static out of band registration like in OpenID Connect is not reasonable. Instead, applications register dynamically with the user's MoPIDP on their first use. Such an use case is supported by OpenID Connect with its specification of dynamic client registration.

Like shown above, an OpenID Connect authentication process usually starts with the service redirecting to the IDP for entering the user's credentials. With their own IDP in hand users are now able to start the process right from the device. For starting the authentication process with a smartphone, the application on the phone needs the information where the user wants to login to. In our implementation an unique QR code on the service's website is used for that purpose. The QR code is easy to create for service providers and as well easy to scan and evaluate for smartphone applications. When the user scans it to use the service authenticated, she gets an overview on her smartphone on the service and the kind of information the service is requesting. After that the user has the possibility to decide whether she wants to login respectively register with the service or not. If she agrees a specific protocol flow starts between the service and the MoPIDP which results in an successful authenticated user.

3.3 Protocol

In this section the protocol flow between a service and a MoPIDP will be explained in detail. The user needs to have the application installed on his response_type=code

&scope=openid%20name%20email%20picture

&state=af0ifjsldkj

&redirect_uri=https://client.example.com/authorize

Fig. 2. MoPIDP QR code example

smartphone and the service provider has to support authentication over the MoPIDP. The QR code displayed on the website contains the content of the query string which is used in OpenID Connect for the authentication request. Only the `client_id` is not included because it differs for every MoPIDP. An example QR code is displayed in Fig. 2.

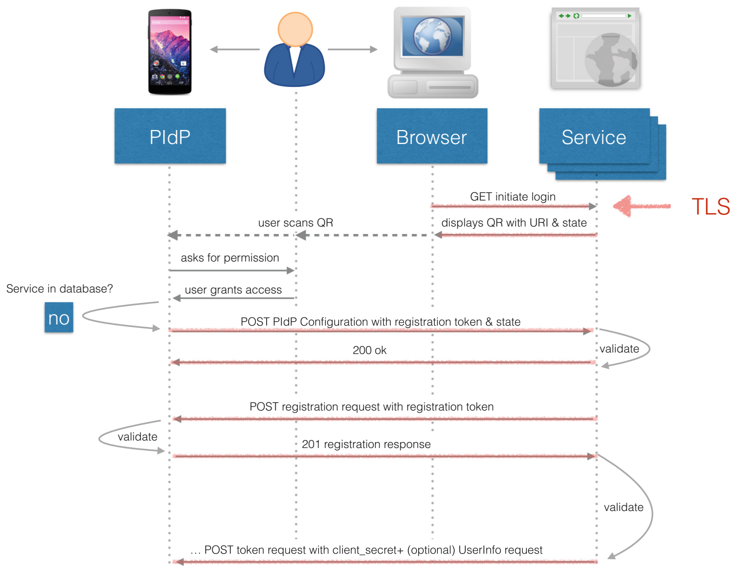

PIdP
Browser
Service
TLS
GET initiate login
user scans QR
displays QR with URI & state
asks for permission
Service in database?
user grants access
no
POST PIdP Configuration with registration token & state
200 ok
validate
POST registration request with registration token
validate
201 registration response
validate
… POST token request with client_secret+ (optional) UserInfo request

Fig. 3. MoPIDP registration flow

Registration. The user starts the MoPIDP application and selects one of her digital identities. After scanning the QR code the application checks if the service has been visited before with this selected identity. If not, the MoPIDP registration flow illustrated in Fig. 3 starts after a user consent. The MoPIDP sends its own configuration data to the registration endpoint of the service. It is the same configuration data which is placed at `/.well-known/openid-configuration` in

OpenID Connect. So the service gets informcation about endpoints, public key location, algorithms and much more. Also the `state` parameter from the QR code and an access token for future requests to the MoPIDP registration endpoint is included. The service has to check if there is an running session for this incoming `state` parameter. If it is valid the service stores the configuration data for this user temporally. With this information the service can make a default registration request to the MoPIDP registration endpoint. Also the answer from the MoPIDP is OpenID Connect standard and includes the parameters `client_id` and `client_secret`. This happens once for every service registering with an digital identity from an MoPIDP.

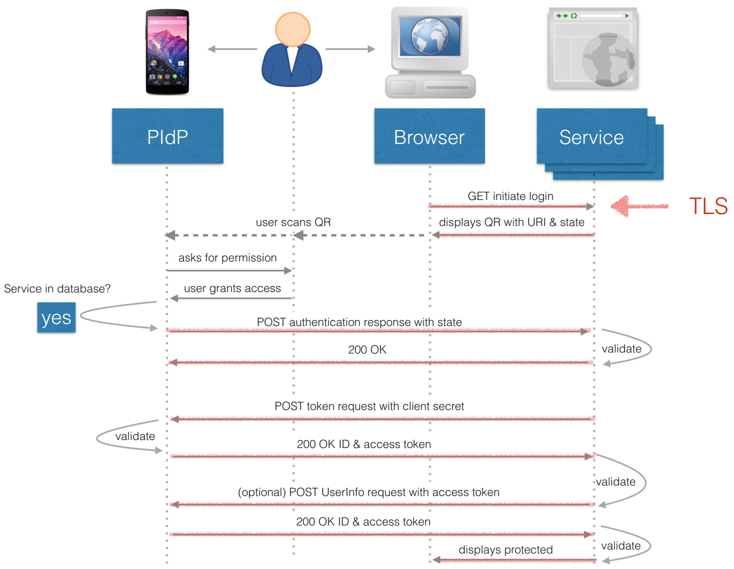


Fig. 4. MoPIDP authentication flow

Authentication. When the user scans the QR code and registration with the used digital identity was already performed, the MoPIDP authentication flow starts which is shown in Fig. 4. This happens without any user interaction because the application records all visited services. The MoPIDP sends an requests to the service which contains the same parameters as they where in the OpenID Connect authentication response. In addition to these parameters it includes the `client_id` so the service can assign the right user data out of its database.

Our system does not use redirects and also does not transfer any kind of sensible data over Web browsers. That is the reason why there is no temporary `code` required like in the OpenID Connect Authorization Code Flow. After a token request, tokens can be send back to the service directly. This token request contains `client_id` and `client_secret` for trust reasons. This client authentication method is called `client_secret_post` in OpenID Connect.

As soon as the service validated the tokens, the user is successfully authenticated. If additional user data is needed, there can be made another request with the just received access token. The structure of this UserInfo request is the same as in OpenID Connect. This mostly happens just once after initial registration because it could replace usual registration forms.

3.4 Implementation

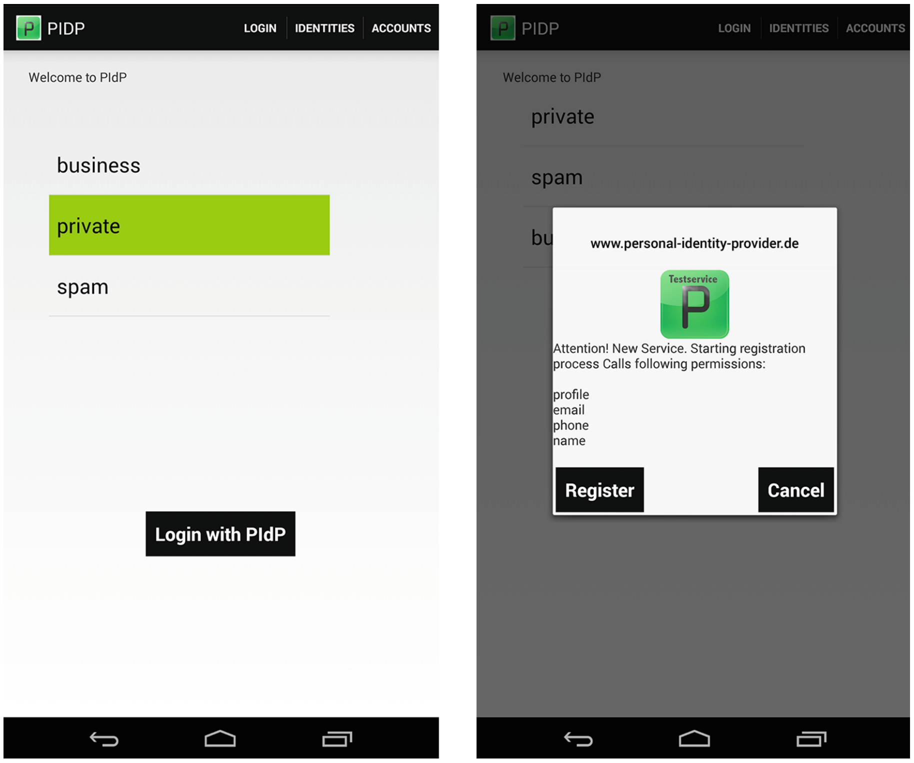
To demonstrate the feasibility of this approach, a functional prototype of the system has been developed. One part of it is a dummy Web application in which potential users want to register via the MoPIDP protocol. On the start page of the service the unique QR code required for authentication is displayed. The QR code includes the above mentioned information like `scopes`, `state` and `redirect uri`. Further, the service includes a protected area with user-specific data.

The MoPIDP application shown in Fig. 5 has been implemented for the Android operation system and offers the possibility to register and log on to MoPIDP-instrumented Web applications with different user-profiles respectively digital identities. It includes the necessary scanning mechanism for reading the QR code, a simple web server for handling requests made to the MoPIDP application and cryptographic libraries for signing and encrypting responses. Firstly, the scanning of QR codes was implemented by an external application installed on the smartphone. However, as little information as possible should leave the MoPIDP application sandbox, which is why this functionality was integrated to the application afterwards.

Another problem which had to be addressed was the continuous connection between the web page opened in a browser and the web server. This connection is necessary for the automatic authentication or registration process, since the server must tell that the QR code has been scanned and the user is successfully authenticated. The problem could be solved by polling, which however leads to a permanent reloading of the page. A more efficient solution is to use WebSockets, a TCP-based protocol that enables bi-directional connections between a web page rendered inside a browser and a web server. The client establishes a TCP connection to the server, which remains open unlike in the HTTP protocol. From then on, both parties can exchange data without new HTTP requests whereby the server can tell the client that an authentication has taken place.

As is also the case with OpenID Connect, tokens are represented as JSON Web Tokens (JWT) [5]. JWT is an open standard for representing and sharing claims for e.g. authorization purposes. JSON Object Signing and Encryption (JOSE) [3] is used for signing or encrypting such JWTs to ensure integrity, confidentiality and authenticity of the contained data. Frameworks for implementing JWT and JOSE are available in many programming languages such as Java [7], C/C++ [6] and Ruby [4]. The Java library of the JWT and JOSE is used for the prototype, both for the service, as well as for the Android application.

In order to be able to sign and encrypt tokens required by the protocol, keys must be known by the parties. Therefore, the service hosts a so called JSON Web Key Set (JWKS) [18] on a specified `.well-known` location to synchronize information about supported algorithms and the service's public key. During the Registration Flow, the MoPIDP application temporary hosts a JWKS too, containing the public key of the chosen identity. Both public keys are used to build so called `Nested JWTs` for the MoPIDP Flows. For example the ID Token is such a Nested JWT since the PIDP signs the requested claims with the identity's private key and subsequently encrypts it with the services public key which was pulled from the `.well-known` JWKS source. So indeed the ID Token is a signed JSON nested inside an encrypted JSON.



Attention! New Service. Starting registration process Calls following permissions:

profile
email
phone
name

Fig. 5. MoPIdP android application

4 Security and Privacy Considerations

Our approach obviously inherits most security properties of OAuth/OpenID Connect. For example, authentication and confidentiality between the peers is performed by the underlying TLS protocol. This means, the transport security relies on the security of TLS and in extend on certificate and PKI security. However, the use of TLS also eases the deployment as no additional exchange of keys is required.

One desired property for IDP systems is the possibility of using different identifiers for different services and prohibiting the linkability of different accounts. Here, a personal IDP has disadvantages, as there is a one-to-one mapping between IDP and user. Thus, even when using different identifiers, these ids can be easily linked. However, as the different identifiers are rather rarely used, this disadvantage is probably not very relevant in practice.

In identity management systems, a high level of trust in the IDP is required, because it can completely impersonate the user. A personal IDP (fixed or mobile) has a huge advantage compared to a (traditional) external IDP, as the user itself is the operator of the IDP.

However, the most severe security threat for IDP based authentication is phishing. Usually, the user accesses the RP and is forwarded by the RP to the IDP. A rogue RP can easily redirect the user to a phishing IDP instead of the correct IDP. Therefore, the user must very carefully check the IDP before authenticating. However, most users neglect this level of caution. As soon as the phisher has gained the IDP authentication token (usually just a password), he can completely impersonate the user on all of the connected sites. With a personal IDP the phishing threat is slightly reduced, but still feasible. With our MoPIDP, however, phishing is not possible, as the redirection step is omitted. Instead the (personal) IDP communicates from the beginning directly to the RP.

Table 1. Comparison of security and privacy aspects

	IDP	PIDP	MoPIDP
Impersonation	−	+	+
Phishing	− −	−	+
Identity linkability	+	o	o
Transport security (TLS)	o	o	o

Table 1 summarizes the above described security and privacy properties. It compares a "standard" OpenID Connect deployment (IDP) with a personal IDP (PIDP) and with our mobile personal IDP (MoPIDP).

Finally, the security of MoPIDP obviously depends on the security of the mobile device. Access to the MoPIDP app and the database must be restricted, e.g. by a PIN or a fingerprint. This prohibits misuse by an adversary in case of lost or stolen device. In that case also the legitimate user looses the credentials stored in the MoPIDP database. This is a standard problem of authentication systems which are based on tokens stored on a mobile device. To solve this issue, either an additional mechanism for account recovery at the service provider must be added or regular backups of the MoPIDP database to a secure location must be created.

5 Conclusion

Authentication and identity management are still open issues in our world of online services. Despite their obvious disadvantages, password authentication and "manual" management of identities by the end user are still the predominant solutions.

In this paper, we presented an approach using a smart phone as personal Identity Provider. This solution exceeds usual identity management solutions with regards to security and usability. As discussed above, a personal IDP has small privacy disadvantages compared to standard IDP deployments (i.e. the possibility of identity linkability). However, our solution is still a major improvement compared to current authentication schemes using an email address (with most people: always the same address) as identifier.

References

1. Abe, T., Itoh, H., Takahashi, K.: Implementing identity provider on mobile phone. In: Proceedings of the 2007 ACM Workshop on Digital Identity Management, DIM 2007, pp. 46–52. ACM, New York (2007). http://doi.acm.org/10.1145/1314403.1314412
2. Alliance, F.: FIDO UAF Architectural Overview (2016). https://fidoalliance.org/specs/fido-uaf-v1.1-rd-20161005/fido-uaf-overview-v1.1-rd-20161005.html
3. Barnes, R., Mozilla: Use Cases and Requirements for JSON Object Signing and Encryption (JOSE) (2014). https://tools.ietf.org/html/rfc7165
4. Bennett, A.: Jose library for ruby. https://github.com/potatosalad/ruby-jose
5. Bradley, J., Sakimura, N., Jones, M.: JSON Web Token (JWT) (2015). https://tools.ietf.org/html/rfc7519
6. Cisco Systems: cjose - jose library for c/c++. https://github.com/cisco/cjose
7. Connect2id: JOSE + JWT library for Java. https://connect2id.com/products/nimbus-jose-jwt
8. Dhamija, R., Dusseault, L.: The seven flaws of identity management: usability and security challenges. IEEE Secur. Priv. **6**(2), 24–29 (2008)
9. Dierks, T.: The Transport Layer Security (TLS) Protocol Version 1.2 (2008). https://tools.ietf.org/html/rfc5246
10. Facebook: Access Tokens - Facebook Login - Documentation (2017). https://developers.facebook.com/docs/facebook-login/access-tokens/
11. Ferdous, M.S., Poet, R.: Portable personal identity provider in mobile phones. In: 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 736–745. IEEE (2013). http://ieeexplore.ieee.org/abstract/document/6680909/
12. Foundation, O.: OpenID Authentication 2.0 (2007). http://openid.net/specs/openid-authentication-2_0.html
13. Google: Google Authenticator (2016). https://github.com/google/google-authenticator
14. Google: Using OAuth 2.0 to Access Google APIs | Google Identity Platform (2016). https://developers.google.com/identity/protocols/OAuth2
15. Haller, N.: The S/KEY One-Time Password System (1995). https://tools.ietf.org/html/rfc1760

16. Hardt, D.: The OAuth 2.0 authorization framework (2012). https://tools.ietf.org/html/rfc6749.txt
17. Jain, A.K., Ross, A., Prabhakar, S.: An introduction to biometric recognition. IEEE Trans. Circuits Syst. Video Technol. **14**(1), 4–20 (2004)
18. Jones, R., Microsoft: JSON Web Key (JWK) (2015). https://tools.ietf.org/html/rfc7517
19. Lockhart, H., Campbell, B.: Security assertion markup language (SAML) V2.0 technical overview. OASIS Comm. Draft **2**, 94–106 (2008). https://www.oasis-open.org/committees/download.php/14360/sstc-saml-tech-overview-2.0-draft-08-diff.pdf
20. Lopez, G., Canovas, O., Gomez-Skarmeta, A.F., Girao, J.: A SWIFT take on identity management. Computer **42**(5), 58–65 (2009)
21. Morgan, R.L., Cantor, S., Carmody, S., Hoehn, W., Klingenstein, K.: Federated security: the shibboleth approach. Educ. Q. **27**(4), 12–17 (2004). http://eric.ed.gov/?id=EJ854029
22. Rydell, J., M'Raihi, D., Pei, M., Machani, S.: TOTP: Time-based One-time Password Algorithm (2011). https://tools.ietf.org/html/rfc6238
23. Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C.: Openid connect core 1.0. The OpenID Foundation p. S3 (2014). http://openid.net/specs/openid-connect-core-1_0-final.html
24. Sun, S.T., Beznosov, K.: The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 378–390. ACM (2012). http://dl.acm.org/citation.cfm?id=2382238
25. Thomas, I., Meinel, C.: An identity provider to manage reliable digital identities for SOA and the web. In: Proceedings of the 9th Symposium on Identity and Trust on the Internet, IDTRUST 2010, pp. 26–36. ACM, New York (2010). http://doi.acm.org/10.1145/1750389.1750393
26. Twitter: OAuth Twitter Developers (2017). https://dev.twitter.com/oauth