Dipartimento di Informatica, Bioingegneria,
Robotica e Ingegneria dei Sistemi

Open Data and Personal Information: A Smart Disclosure Approach based on OAuth 2.0 (extended version)

G. Ciaccio, A. Pastorino, M. Ribaudo
DIBRIS, Università di Genova, Italy
{giuseppe.ciaccio,marina.ribaudo}@unige.it

Technical Reports Series **DIBRIS-TR-13-01**

DIBRIS, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy

February 2013 - Updated December 2013
http://www.dibris.unige.it/

Abstract

Currently, public administration is undergoing significant transformations, driven by a greater demand for transparency and efficiency in a participative framework involving nonprofit organizations, enterprises, and citizens, with the modern network infrastructure as a common medium. The Open Data movement is considered one of the keys to this change. One of the forthcoming evolutions of Open Data is the idea of "smart disclosure" of personal data, mostly managed by public administrations, in order to allow third party applications to provide new personalized online services to individuals and organizations. In this paper we propose a possible implementation of the "smart disclosure" idea, that takes advantage of the OAuth 2.0 authorization framework. OAuth 2.0, if properly implemented, guarantees access to selected personal data upon authorization of the individual data owner. An implementation is presented together with possible use cases.

1 Introduction and motivation

The *Memorandum on Transparency and Open Government* signed by the US president Barack Obama [23] has fostered a new era for the public sector in which *transparency*, *participation*, *collaboration* and, ultimately, *Open Government* should become central in the democratic decision process. Administrations should become more transparent and promote the use of new technologies to ensure that the data they routinely produce and manage are made available online so that they can be leveraged by any party: enterprises, private citizens, public entities, and other branches of the public administration. This document marked the official onset of the so called *Open Data movement*.

Shortly afterwards, several public administrations in the USA and UK started releasing massive amounts of Open Data in the form of aggregated datasets made available on their websites. The first catalog of Open Data was published in May 2009 by the US Government (http://data.gov), followed by the UK (http://data.gov.uk). The British Government is now at the forefront in Europe, engaging in an unparalleled effort towards widespread adoption of the Open Data paradigm.

Quoting [6], *"Data is the 21st century's new raw material"*: by means of handheld devices, social networks, cash dispensers, credit cards, people are directly or indirectly generating an unprecedented volume of data that is deemed to transform our very lives [28].

The current wave of Open Data released by public administrations is largely made of formatted datasets of a static nature (i.e., they will not reflect changes occurring after the release date) and concerning diverse fields: politics, traffic and local transportation, tourism and culture, the environment, healthcare and welfare, cartography, and many others. Such datasets are roughly of two kinds, namely: aggregated and anonymized data (e.g. number of children in each school of the region); and identification data of public entities (e.g. names and addresses of restaurants in the region). No data concerning individuals have been released due to obvious privacy issues. As already stated in a position paper of ours [8], such a lack of *personal data* in the Open Data realm, along with the static nature of the released datasets, are weaknesses of the current wave of Open Data. Without personal data and without timeliness, it is indeed impossible to build useful services tailored to the actual needs of a given individual at a given time.

Many of the data managed by public administrations as well as private entities are of a personal kind. Consider, for instance, the huge amount of personal data contributed to the various online social networks, or the electricity consumption data collected and stored by energy providers, or the telephone and internet data collected by telecommunications companies. As these data are not in the Open Data domain, those public and private entities may act as the "owners" of our data. This means they hold a monopoly on services while we, the legitimate owners of the data, must abide by their terms and conditions concerning how our data are treated and used.

By unleashing personal data "into the wild", such a monopoly would collapse and a new ecosystem of personal services based on these data could flourish. While we cannot expect a corporate giant

like Facebook to voluntarily relinquish the personal data contributed by us to their servers, we can easily imagine that a public administration could, and arguably should, do so. One possible scenario would see third parties routinely providing public online services that make use of personal data, and the administrations routinely providing personal data online to registered third parties on behalf of the legitimate data owners (the taxpayers). In such a scenario, the administrations are responsible for ensuring the authenticity and integrity of personal data, preventing any unauthorized access, yet allowing what is called a *smart disclosure* of personal data to the web.

The importance of personal data as an economic asset on its own is now being acknowledged worldwide [21], along with the need to strengthen trust by people in the possible process of smart disclosure to be undertaken by public administrations [28]. Smart disclosure of personal data is considered a forthcoming process capable of *"enormous economic and civic good opportunities"* [11]. A recent white paper from the UK Government [6] stresses the importance of smart disclosure as an enhancement of the current Open Data movement. The 'midata' initiative by the UK Government (www.bis.gov.uk/news/topstories/2011/Nov/midata) [25] and the Smart Disclosure initiative by the White House (www.whitehouse.gov/blog/2012/03/30/informing-consumers-through-smart-disclosure) are two programs aimed at promoting smart disclosure of customer's personal data which are held by companies and providers so as to allow people to make better choices.

It might be argued that adding personal data to the Open Data heap might jeopardize our privacy, if done in the wrong way. However, this risk is also present with the current process of releasing massive anonymized datasets. By definition, these datasets leak personal information, and information from many datasets may be jointly mined in search of individual profiles. The inferred profiles may sometimes be linked to real identities, leading to statistical de-anonymization or "identity disclosure through mosaic effect" [28]. The whole Open Data movement would immediately come to an end, should these confidentiality concerns prevail over the individual and social benefits of transparency and smart disclosure. A balance between privacy and transparency must clearly be sought, with the information technology playing a key role.

Another criticism is that, once disclosed and no matter how "smartly" this is done, our personal data might be copied somewhere else and we, the legitimate owners, would no longer be able to exert control over the copies. But this indeed holds without disclosure as well, as we currently have no choice but to trust the entity that stores and "owns" our data, without any actual control by us. In addition, due to the lack of smart disclosure, we are forced to input our personal information by hand every time we register for a new online service (and abide by *their* terms and conditions). It would be much easier to refer to a single master copy of our personal data, either centrally stored or scattered across servers in a distributed system, and smartly disclose them to third parties after obtaining their formal commitment online to *our* terms and conditions (for instance, prohibiting unauthorized distribution of copies).

Last but not least, a proper technology for a "sufficiently smart" disclosure of data remains to be identified, along with a number of practical use cases working as an informal definition of what a smart disclosure is. In this paper we propose a few such use cases, and we advocate the use of the OAuth 2.0 authorization framework [5] to achieve smart disclosure. On the basis of this approach, individuals are restored to their role of *resource owners* while administrations (public or private ones) are stripped of their de-facto ownership of personal data and keep a role of bare *resource managers*. A resource owner (namely, an individual) may grant online authorization to any third party application to use a given item of personal data located on a given resource manager, in exchange for a useful personalized online service that the application is expected to provide using that data item. The process of granting authorization is based on unforgeable cryptographic tokens released by a trusted *authorization server* with which individuals, applications, and resource managers, are all registered.

The balance of this paper is as follows. Section 2 introduces some significant examples of implementa-

tion of Open Data involving personal data. Section 3 introduces the OAuth 2.0 framework, showing how it can be used to grant access to personal data while preserving individual privacy. Section 4 proposes a few simple use cases in which OAuth 2.0 could be successful in supporting the smart disclosure idea. Section 5 discusses our current PHP implementation of OAuth 2.0, thanks to which we plan to demonstrate suitability with the use cases. Finally, Section 6 suggests possible directions for further research.

2 Smart disclosure stories

Several examples and case studies on the use of the current wave of Open Data do exist but, when the search concentrates on smart disclosure success stories, results become more difficult to discover. In this section we briefly recall some examples we are aware of.

The Department for Business Innovation & Skills in the UK has launched the 'midata' project (www.bis.gov.uk/news/topstories/2011/Nov/midata) which aims to give consumers more control and access to their personal data. The program involves various business sectors including energy, telecommunications, finance and retail. By letting customers access data about their purchasing and consumption habits, and safely add new data and feedback of their own, businesses have the opportunity to create rich, new person-centric applications while consumers can make better consumption decisions and lifestyle choices.

The next two examples are related to the medical context. In the private sector, Microsoft has proposed the HealthVault (www.microsoft.com/engb/healthvault/default.aspx) as *"a trusted place for people to organize, store, and share health information online."* According to the HealthVault website, Microsoft offers an open platform for security enhanced data sharing amongst health services organizations and citizens. Any information entered into HealthVault can be, with the citizen's permission, re-used across many different apps and supplemented by a growing list of devices. Apparently, the service is now available for US, UK and Germany.

In the public sector an interesting success story is the project known as the Blue Button (www.bluebutton.com). Developed in collaboration with the US Department of Veteran Affairs (VA), the project allows veterans to go to the VA website, click a blue button, and download their personal health records. These records can be individually examined or shared for example with doctors or with third parties applications. The Blue Button download capability can help individuals access their information so they can manage their health care more effectively. We were unable to find technical details concerning how Blue Button is implemented; in particular, we do not know whether the various health data belonging to a given user are routinely copied from the health care centers to the Blue Button server, or are rather left at their originating servers and collected on the user demand; this second approach, however, would be a complete yet simple example of smart disclosure.

A very successful initiative is the Green Button project (www.greenbuttondata.org), similar to 'midata', and part of the White House's Smart Disclosure Program. Thanks to this program, consumers in the US can access and download their energy usage information provided by their utility or retail energy service provider, take advantage of online services and apps, and manage their energy consumption. From the scarce information we were able to retrieve [29], it seems that the service leverages Apache Wink [1] for exporting data in a RESTful way via the Atom Publishing protocol [15], plus Spring Social [4] as the authorization framework for user-controlled smart disclosure. Spring Social leverages OAuth [12], so the overall picture is similar to the one we provide in this paper.

Falcão-Reis and Correia [10] propose coupling Electronic Health Records (EHRs) with an extended version of OpenID [3, 24] in an effort to implement a user-controlled system of Health Digital Identity for Portuguese citizens. They also propose leveraging OAuth 2.0 as an authorization technology for user-controlled access to EHRs, thus anticipating smart disclosure in the medical care field.

In all these examples, the recurring theme is that the wealth of personal data contained in medical records, telephone or energy usage reports, or other information sources, present a unique opportunity for software developers to build applications that can truly transform how individuals interact with their data to stay healthy and manage their care, to save energy and therefore money, in other words to improve several aspects of their everyday life.

3 OAuth 2.0

3.1 Goals and current status

OAuth 2.0 [5, 12, 9] originated in 2010 from a complete redesign of the previous OAuth 1.0 specification [16]. OAuth 2.0 is too high-level to be defined as a protocol specification. It should be considered as a blueprint of a protocol, within which many implementations are feasible, although possibly not interoperable with one another. This is maybe (and hopefully) just a sign of immaturity of the specification. Nevertheless the interest around this technology is huge: the IETF OAuth working group include members like Google, Facebook, Salesforce, Microsoft, Twitter, Deutsche Telekom, and Mozilla [9], and OAuth 2.0 has already been adopted by Google [7], and Facebook [2], just to cite a few. We are quite confident that OAuth will soon become the reference technology for authorization and controlled access on the internet.

OAuth stems from a typical Web 2.0 use case with online social networks. Suppose a third party client application is offering a user a personalized service making use of that user's personal features provided by a social network (e.g. pictures, contact list, posting a comment) via a RESTful API. The naive approach of requiring users to release their credentials (username and password) to the third party client so that the latter could get those features from the social network is highly risky. A more appropriate solution is to release cryptographic proof of *authorization*, issued by the user (at least in principle) to the third party application, and to let the latter subsequently spend such authorization proof at the social network API in order to get the required features. In addition, the authorization proof may work as a form of user *authentication* whenever it grants access to user identification data stored in the social network or elsewhere.

3.2 Abstract protocol

Abstractly, OAuth 2.0 identifies four actors exchanging information in an ordered way (Figure 1). These actors are the *client application* or just "client", the *resource owner* or "owner" for short (or "user" when human), an *authorization server* (AS, the trusted entity), and one or more *resource servers* (RS) hosting data or services to be smartly disclosed. The AS and the various RSs may each belong to a distinct administration domain, but this is not mandatory.

The protocol is started by the client requesting authorization to access a given *resource* (e.g. a set of personal data), subject to a set of constraints called *scope* (e.g. read vs. write access to specific fields of personal data during a specified time window). The resource owner is shown the request and, if they agree with the scope, they may yield an *authorization grant* bound to the resource and scope. Such a grant is then exhibited by the client to the AS, which validates it and returns an *access token* valid for the required resource at a specific RS subject to the scope. The client finally passes the access token to the given RS along with the resource request; upon token validation and scope verification, the RS returns the resource to the client.

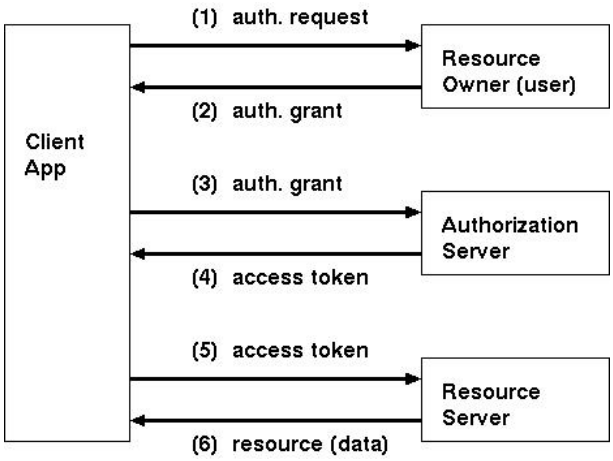

Figure 1. OAuth 2.0 abstract protocol, with the four actors (client application, resource owner, authorization server, and resource server) and their interplay. Actions are ordered by increasing number.

3.3 Four ways of obtaining authorization

In practice, the authorization grant can be obtained in four possible ways. These four ways are called *flows* in OAuth jargon (IETF could not call them "protocols" due to the many details that are omitted or left undefined), and are nothing but instantiations of the abstract protocol scheme discussed above.

In the first flow the grant is obtained indirectly, with the AS acting as an intermediary under the control of the resource owner. The flow is depicted in Figure 2. With this flow, the client is typically (but not necessarily) a web application on a remote server, and the resource owner is typically (but not necessarily) a human with a browser initially pointed to the web application. The grant is represented by an *authorization code* issued by the AS after obtaining consent from the resource owner, and is delivered to the client through the browser of the resource owner via HTTP redirection (see the Figure).

In the current IETF draft, TLS protection is not required when the AS redirects the browser to the client after the authorization step; in other words, it is legal for the client redirection URI (Figure 2) to be an HTTP endpoint of the web application instead of an HTTPS one. On the one hand, this shortcoming is meant to cover client applications that are unable or unwilling to provide TLS endpoints (due to lack of resources, for instance). On the other hand, an authorization code sent to a non-TLS endpoint is transmitted in plain text and could therefore easily be eavesdropped. The stolen authorization code could then be used by an attacker client to obtain an access token from the AS. But if the client application has a persistent identity registered at the AS, that is, it shares a secret with the AS and can prove that it holds it (a "confidential" client, in the OAuth jargon), then the AS will prompt the client application to authenticate before converting the authorization code into an access token, thereby preventing the use of stolen authorization codes by rogue clients. As an additional security measure, the access token itself may be bound to the specific client identity (a so called "proof token" [17], as opposed to an anonymous "bearer token"). In contrast, if the client does not hold a secure and persistent secret registered with the AS (a so called "public" client) then the flow is insecure, unless the client redirection URI is an HTTPS endpoint.

A security analysis of OAuth 2.0 is beyond the scope of this paper, and can be found in a dedicated IETF draft [17].

In the second flow the authorization grant is *implicit*, that is, no intermediate authorization code is

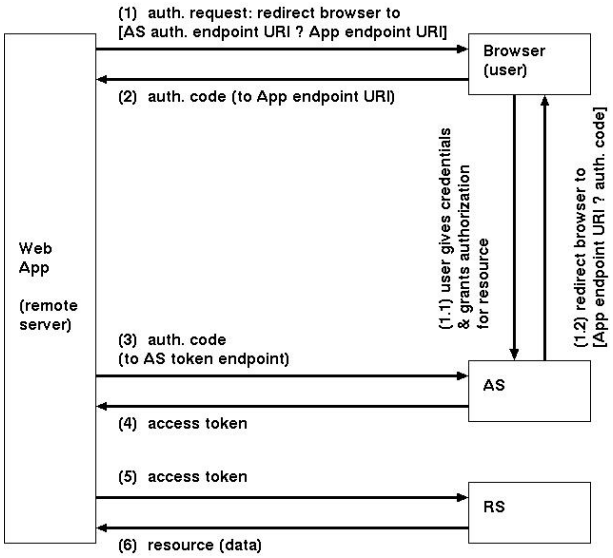

Figure 2. OAuth 2.0 Authorization Code flow, for web applications. The "?" denotes an HTTP GET parameter. See [12] for more details.

released. As apparent from Figure 3, with this flow the resource owner is typically a user with a browser, and the client is a Java applet or an embedded Javascript code, loaded by the browser from the client's website. The user yields credentials to the AS and grants authorization through the browser, and the applet is then given an access token directly.

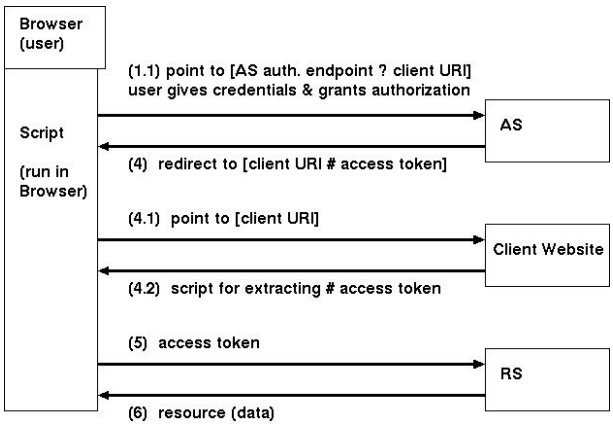

Figure 3. OAuth 2.0 Implicit flow, for applets and scripts running in browsers. The "#" denotes a URI fragment. See [12] for more details.

The Implicit flow is said to be possibly more responsive than the previous one [12]. On the other hand, its typical application scenario is less secure. A client consisting of an applet or embedded script can in principle build a secret, but such a secret is not persistent (it might not survive from one run to another), so it cannot be registered at the AS (the client is "public", in other words) and thus the access token cannot be bound to any specific client identity. In addition, the received access token would be

stored in the browser at the user device, whose security level is generally low. The access token could be easily stolen by a malware affecting the user device and then sent out to an attacker elsewhere, who can well make use of it, because it is not bound to any specific client identity.

The remaining two flows are more simplified. In the *Resource Owner Password Credential* (ROPC) flow (Figure 4), the client is typically a native application installed in the user device, and the user interacts directly and uniquely with it. The application takes credentials from the user, and sends these credentials to the AS for obtaining an access token. There is no easy way for the user to inspect the access scope being requested by the client to the AS, thus the user must trust the client. A native application can generate a persistent secret so it can have an identity, but such identity is as secure as the user device where the application is installed; so the typical scenario for the ROPC flow is generally vulnerable to, for instance, client impersonation attacks.

Lastly, with the *Client Credentials* flow (Figure 5) the client appears to be resource owner so it sends its own credentials to the AS, and get the access token directly. In fact this straightforward flow requires that the authorization by the actual resource owner is given outside of the OAuth flow (in the jargon of the old OAuth 1.0 protocol, this would be called a "two legged" variant of the protocol).

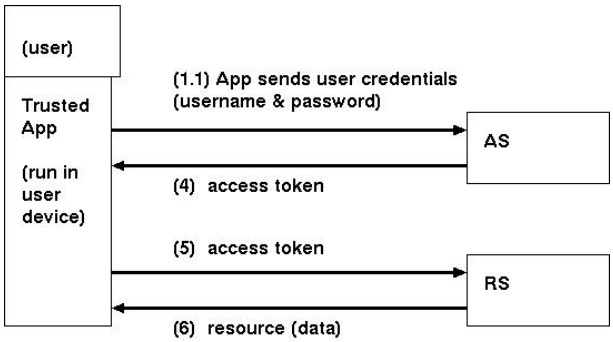

Figure 4. OAuth 2.0 Resource Owner Password Credentials (ROPC) flow, for trusted apps installed in the user device. The user credentials are sent by the app to the AS as proof of authorization grant by the user.

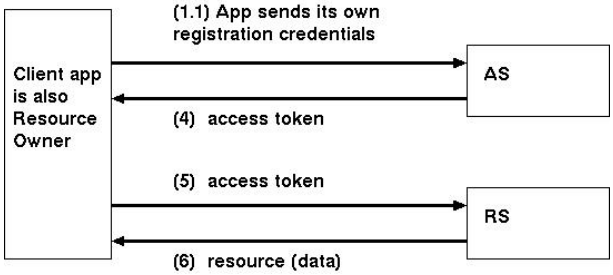

Figure 5. OAuth 2.0 Client Credentials flow, for client applications that are also resource owners. A simplified form of the ROPC flow with no user involved.

3.4 Token expiration and refresh

When a resource owner gives authorization, they may specify a scope (what resources and for what kind of access) and also a lifetime (the expiration time of the authorization). So, in principle, an access token might inherit a lifetime equal to the authorized lifetime. In practice, however, each access token might be given a shorter lifetime, after which it must be refreshed, so a form of revocation can be easily obtained by simply refusing to refresh an expired token. More in detail, each new access token can be accompanied by a corresponding *refresh token* (Figure 6). The client can spend the refresh token back at the AS in order to get a new access token (and corresponding new refresh token) for the same resource under the same scope or a narrower one, until the overall authorization lifetime, as originally granted by the owner, has expired. A user could revoke authorization by instructing the AS to refuse refresh tokens from that client for the time being, but nothing can be done to invalidate an access token after it has been issued.

Refresh tokens are not permitted with the Implicit and Client Credential flows. With these flows, instead, an expired access token can only be replaced by a new one after replaying the flow from the beginning.

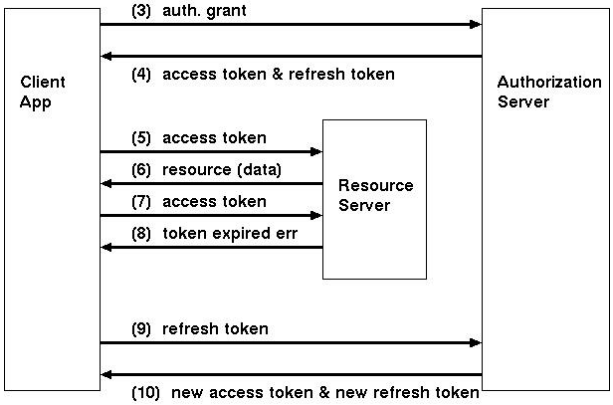

Figure 6. Refreshing expired access tokens in OAuth 2.0. Actions are ordered by increasing number.

3.5 Registration and authentication to the Authorization Server

From the above it is reasonable to deduce that resource owners (users), clients, and RSs, should establish a relation with the AS before engaging in any OAuth 2.0 protocols. The current IETF draft explicitly leaves out of scope (but does not forbid) any interaction between an RS and the AS, and seemingly ignores the registration of resource owners, although this is indeed a necessary step if the AS is to issue authorization grants on their behalf. Only client registration to AS is explicitly mentioned in the IETF draft. After registration, the client is given a unique identifier that is valid at the AS. "Confidential" clients will be allowed to associate their secret credentials with the identifier at the AS for future authentication.

4 Use cases

This section introduces two possible use cases that show what a smart disclosure is and how to obtain it using OAuth 2.0. For each use case we first informally sketch the problem, then we discuss how OAuth

2.0 could improve current solutions and present aspects which are challenging to the current OAuth 2.0 flows.

4.1 Medical use case

This trivial use case is also found in the work by Falcão-Reis and Correia [10], although they failed to mention some potential challenges arising from it.

Scenario. Alice needs professional medical care and asks for advice; a friend recommends Dr. John Smith. Alice makes an appointment and takes all the medical certificates she can find at home with her. She then summarizes her medical history, presents her problem and listens to the doctor's response.

How OAuth 2.0 could help. In many countries, health records are still predominantly paper records, given to patients after medical examinations. Patients' data are stored in local databases managed by various laboratories, hospitals, or other health care settings, all of which may be using different technologies and data representation that do not usually interoperate; storing data as PDF files is a common practice. Health information is therefore scattered across many places, and accessing the medical history of an individual is a complex task.

A possible solution requires the adoption of a machine-readable representation of medical data, along with the definition of an architecture for retrieving and merging records spread over several databases. In the medical context, standard data representations do exist, for instance the HL7 Clinical Document Architecture, a document markup standard that specifies the structure and semantics of clinical documents for the purpose of exchange. OAuth 2.0 could be the enabling technology of this architecture if adopted to get authorization grants across different resource servers (the different medical databases). The use of OAuth 2.0 does not require centralizing data into a single repository: data are kept where they have been produced and are accessed upon authorization by the owner.

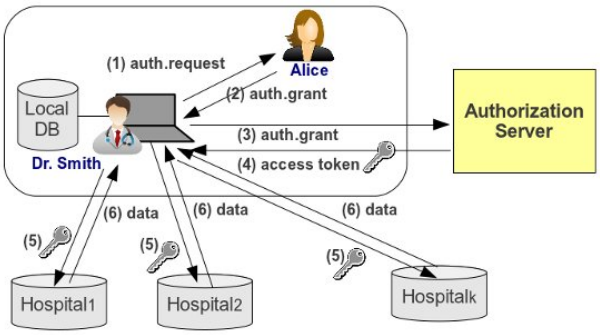

Figure 7. Dealing with distributed health data via OAuth 2.0.

Figure 7 shows Alice and Dr. Smith. In order to know Alice's medical history, Dr. Smith uses a web application that connects to the distinct databases of hospitals that provide online health data upon verification of access tokens (resource servers Hospital1, Hospital2, ..., Hospitalk in the picture).

Alice is in front of the doctor, so she can give online consent to access her personal resources (arrows numbered 1 and 2) by interacting with the web application being used by the doctor. After receiving an authorization grant, the web application applies for an access token (arrows 3 and 4) which is subsequently used to collect Alice's data stored within the resource servers (arrows 5 and 6), thus building a view of Alice's medical history without resorting to any paper document and, perhaps more importantly, without missing any of Alice's medical records.

Challenges. Alice might not want to show all of her medical records to the doctor. Some illnesses, like HIV infection for instance, pose serious privacy concerns and might not be significant for the specific disease Alice is asking advice for. Scopes in OAuth 2.0 allow to specify which fields in a generic record can be accessed and which cannot, but they are not able to discriminate among distinct instances of the same field in distinct records. A workaround might be to have a specific field for privacy-sensitive information and letting the user decide whether to authorize disclosure of those sensitive fields or not.

In a different scenario, Alice might not be able to give authorization, perhaps because she is unconscious, due for example to a car accident. In this case, there must be another entity which has enough privileges to give authorization without being the data owner. This entity might be the head physician or a close relative of Alice's, and currently this scenario is not explicitly covered by OAuth 2.0, which contemplates a single owner for each data item.

In the medical context it is also very important to have access to aggregated and anonymized datasets for different purposes, for instance to perform statistical analysis for a given disease or to compare the performance of different hospitals. This release of large anonymized datasets is not covered by the current OAuth 2.0 flows which indeed grant access to specific records of an individual user. We will briefly discuss these points in Section 6.

4.2 Tax payment

Scenario. Bob needs to fill in his annual tax return. As usual, he visits his business accountant bringing lots of pieces of paper: his annual salary, medical expenses, documents concerning his properties, other expenses he can deduct from his income, and so on. The other possibility is to use a web application provided by the government and fill in an online form by manually copying all the data printed into the various paper documents.

How OAuth 2.0 could help. Figure 8 shows Bob in front of an innovative online service. He still has access to a web application to fill in the online form for his annual tax payment. This year, however, most of Bob's income information comes directly from various remote databases where it is scattered, upon a simple online authorization by Bob himself. Filling in the form is simpler, quicker and less prone to mistakes.

Bob is the owner of the data. He applies for the authorization grant (arrows 1 and 2) so that the web application he is connected to can directly access his restricted access resources stored in different databases (the Employer Registry, the Estate Registry, the Medical Expenses Registry shown in the figure). He then fills in the form adding only data that could not be obtained via online third parties.

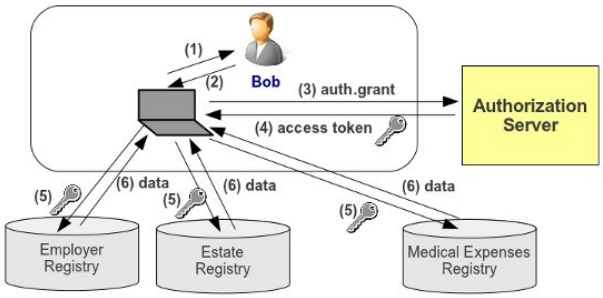

Figure 8. Tax payment via OAuth 2.0.

Challenges. Even in this case, we can identify entities that might need to access data without being the owners. Consider for instance the Judiciary or the Inland Revenue Office in case of lawsuits related

to tax evasion. Moreover, the release of aggregated data is desirable for statistical purposes in this use case too.

In this scenario there is also another subtle point. The same resource, for instance a receipt of payment, involves two people. For example the receipt for plumbing work carried out in an apartment involves the plumber who did the work and received the money, and the owner of the apartment who paid for the work. For the former the resource represents income, while for the latter it is an expense; there is only one resource but it has two distinct resource owners. How to deal with resources of this type is not explicitly specified by OAuth 2.0 (see Section 6).

5 A working prototype

Currently we have a working implementation of the OAuth 2.0 authorization server, that we have called *OAuthwo*. Due to the high level nature of the OAuth 2.0 framework, many features are left to the implementor's choice. In this section we describe the main features of OAuthwo, especially those ones regarding the mitigation of some security issues and how it handles multiple resource servers with possibly multiple credentials of the same user.

OAuthwo is free software, written in PHP as a Zend (framework.zend.com) module and available for download at GitHub (github.com/andou/oauthwo_zend/). Although OAuthwo is written in PHP, we will not bind its description to any specific implementation language.

5.1 Data elements

In OAuthwo the authorization server AS must keep some information about resource owners, clients and resource servers.

5.1.1 Information about clients

Information for a client C retained by the AS are:

- C_{ID}, a unique ID for C
- C_{secret}, a shared secret between AS and C, present if C is a confidential client and used for client authentication
- $C_{profile}$, the profile of C, with possible values *web*, *user-agent* or *native* ([12], Section 2.1)
- $C_{redirectionURI}$, the client redirection URI as specified in Section 3
- C_{name}, a human-readable name for C.

These data are provided by each client during the preliminary client registration phase.

In OAuthwo, public clients as well as confidential clients are required to register their redirection URIs. Each client must register one and only one complete redirection URI, preventing the "Authorization Code Redirection URI Manipulation" and mitigating the "Client Impersonation" and the "Open Redirectors" problems described in [17].

5.1.2 Information about resource servers

Information for a resource server RS retained by the AS are:

- RS_{ID}, a unique identifier for RS

- RS_{secret}, a secret key shared between AS and RS, to be used for encrypting access tokens; see Section 5.2.2
- $RS_{idMethod}$, the way by which RS identifies users within its realm, e.g. by email, social security number, phone number or others; see Section 5.1.3
- $RS_{endpointURI}$, the URI from which RS's data could be accessed.

Furthermore, AS retains information about allowed scopes on each RS. All these information are gathered during trust establishment between each resource server and the authorization server.

5.1.3 Information about resource owners

In order to deliver user-specific protected resources, a RS employs one out of several available user identification methods (e.g. email, social security number, phone number, etc.). Thus the same user can have different identifiers at different RSs and even another different one at the AS. For example a resource server RS_1 could identify users by email, RS_2 by social security number and RS_3 could use email too, but with different addresses. AS must keep track of the identification and authentication scheme in use at each RS. More precisely, the AS must build and maintain an *identity equivalence class* for each resource owner. This is necessary because, for instance, when the AS issues an access token T to be used at an RS to retrieve information on the resource owner U, T must contain an identification key that is valid at the RS, so that RS could properly identify U.

In order to build the identity equivalence class, during the user registration phase the user is prompted with a list of RSs known by the AS and is asked for an identification key for each of these. The AS should then verify that the specified identification keys do actually belong to the user. Some of the identification keys are easily verifiable online (e.g. email, social network nickname), others may require a third party digital certification (e.g. social security number).

An OAuth 2.0 authorization server only needs to abstractly identify and authenticate resource owners, regardless of any authentication mechanism. Consistently OAuthwo is not bound to any specific login mechanism, to foster its use as an authorization component or plugin inside other software applications with their own login mechanisms.

5.2 Access Tokens

An access token represents the authorization originally given by a resource owner for accessing a certain set of resources, subject to given constraints (scope and lifetime).

5.2.1 Identifier vs self-contained token

While the purpose of access tokens is well defined within the OAuth specification, their format and structure are not. Yet, two main categories can be distinguished. The access token may be an opaque identifier, used by the RS to retrieve the authorization information; or it may self-contain such information in a verifiable form.

The former kind of token does not require resource servers to implement any cryptography and allows for timely revocation of authorization. On the other hand this solution implies a further communication between token receiver and issuer to retrieve authorization information and it is thus less scalable.

The latter kind of token, however, requires resource servers to be able to interpret cryptographic messages and implies the use of short-living access tokens due to impossibility to revoke a token once

issued. As the access information is self-contained, this solution requires less communication and is therefore more scalable.

OAuthwo uses self-contained bearer tokens [18] whose format and content is described below.

5.2.2 Access token format

When a client seeks authorization for protected resources stored in multiple resource servers $RS_1, ..., RS_n$, the OAuthwo AS issues a composite access token T made of several chunks $T_1, ..., T_n$. The generic chunk T_i is to be spent at resource server RS_i.

Access tokens are served to clients through the access token parameter in the successful response (see [12], section 5.1). In OAuthwo this parameter is a base 64 encoding [14] of a JSON [13] structure which serializes an associative array where the keys are the URIs of resource servers $RS_1, ..., RS_n$ and the values are the chunks $T_1, ..., T_n$. Each T_i consists of a triple $< U_i, S_i, P_i >$. S_i, the scope, instructs the client on what actions are allowed on what data at resource server RS_i, using U_i as endpoint and P_i as access token for that particular resource server. P_i is thus a monolithic self-contained access token for a single resource server. Being self-contained, P_i carries all the information needed by the resource server RS_i in order to provide the requested resources, namely:

- $P_{(i,userReference)}$, a unique identifier of the user (resource owner) recognized at RS_i
- $P_{(i,scopes)}$, a space separated list of the scopes allowed by the token
- $P_{(i,validity)}$, the token lifetime.

These information are serialized as claims in a JSON Web Token [22]. To prevent token manifacture and modification [18], P_i is digitally signed by the AS with a JSON Web Signature [20].

The user identifier $P_{(i,userReference)}$ valid at RS_i is obtained by the AS using the identity equivalence class it maintains (Section 5.1.3). Such identifier must be considered as a sensitive information; to prevent disclosing it to the client [18], each serialized and signed P_i is finally encrypted with the secret key shared between the AS and RS_i (Section 5.1.2) using JSON Web Encryption [19].

5.3 Authorization Codes and Refresh Tokens

In OAuthwo, both authorization codes and refresh tokens are generated by means of random numbers. Such a random number is associated with the client for which it was issued and the resource owner for which the grant was requested, along with the set of yielded scopes and a timestamp used to check time validity. In OAuthwo the generation of this random number relies on a pseudo-random generation of 20 bytes[1], which ensures the generated number to be resonably random and hard to guess by a potential attacker. The probability of an attacker guessing the generated authorization code is 2^{-160}, as required in [12].

Each issued authorization code is short lived (10 minutes) and single use, while each issued refresh token is long lived (5 days) and single use, which means that a new refresh token is issued with every access token refresh response. The previous refresh token is then discarded.

[1]OpenSSL RAND_pseudo_byte(). Description and synopsis of this cryptographic function can be retrieved at http://www.openssl.org/docs/crypto/RAND_bytes.html

6 Conclusions and further work

In this paper we have made a case for releasing personal data in the Open Data domain by following a smart disclosure approach, and have proposed the adoption of the OAuth 2.0 authorization framework to this end. OAuth 2.0, if properly implemented, guarantees access to selected personal data upon authorization of the individual data owner. We have presented OAuthwo, a free and modular PHP implementation of OAuth 2.0 based on the current IETF draft specification. OAuthwo will allow us and everybody else to play with the OAuth 2.0 technology and implement proof-of-concept smart disclosure systems supporting a number of interesting use cases that leverage personal data securely disclosed as Open Data.

A prof of concept in tax payment domain (Section 4.2) has been developed as a master thesis [26]. The prototype uses OAuthwo to build a web application (https://dione.disi.unige.it) supporting a simplified yet representative example of tax return, namely, the italian *modello 730* (devoid of its graphical skin for simplicity) in the case of a single income plus medical expenses and the property of a single residential apartment, along with anagraphic personal data.

One of our current tasks is to extend the set of use cases. This will allows us to strengthen our claim about OAuth 2.0 being a key technology for smart disclosure. But more important, this will allow us to identify significant scenarios that put the current OAuth 2.0 framework at challenge, ultimately suggesting new features to be added to OAuth 2.0. In Section 4 we already pointed out some of these challenges.

One challenge has to do with single resources owned by multiple users. OAuth 2.0 seems at odds with such a setting. As illustrated in the medical use case, the life of an unconscious patient might depend upon the rescuing physicians being able to access that patient's online medical record, yet the resource owner (the patient) would be unable to provide consent. One possible solution is to allow multiple owners, and then asynchronously warn each of the owners when one of them is granting authorization for their common resources. In some cases the set of owners is static (e.g. the plumber and the householder in the receipt for plumbing work, the citizen and the Judiciary in all tax-related data), but in other cases it is not (e.g. the unconscious patient after an accident and the physicians involved in the rescue). This whole point needs investigation.

Another challenge is related to anonymized datasets. OAuth 2.0 lacks mechanisms for defining what is an anonymized version of a resource. It is unclear when and how a resource owner could authorize access to anonymized resources, and it is also unclear whether or not the resource owner should be able to exert ownership at all over anonymized versions of their data. In any case, authorization should be given ahead of time since the aggregation of anonymized data might take place at any time in the future and even repeatedly. The aggregation procedure itself might need special primitives for bulk verification of authorization grants, since verification on an individual basis would be highly inefficient when extracting a large anonymized dataset.

Last but not least, the whole OAuth 2.0 mechanism seems unable to support any form of delegation of access rights to third parties. Such delegation is very useful in a number of cases where the data owner needs to grant access right ahead of time, so that a third party can subsequently and asynchronously run an application that makes use of personal data of that owner. Even in a simple use case like tax payment there is a obvious need for the user to be able to delegate a professional for doing the entire job of the annual tax return. It would be impractical for the professional to require the user be present during the procedure just for giving consent to the data accesses being performed. A solution might be found in an evolution of the OAuth 2.0 technology, namely, User-Managed Access (UMA) [27]. We are currently investigating on this very point.

References

[1] Apache Wink. incubator.apache.org/wink/.

[2] Facebook login – Facebook Developers. developers.facebook.com/docs/concepts/login/.

[3] OpenID Foundation. http://openid.net/.

[4] Spring Social — SpringSource.org. www.springsource.org/spring-social.

[5] OAuth 2.0. oauth.net/2/, 2010.

[6] Open Data White Paper: Unleashing the Potential. Technical report, Her Majesty Government, UK, June 2012. Retrieved Dec. 2013 from www.cabinetoffice.gov.uk/resource-library/open-data-white-paper-unleashing-potential.

[7] Using OAuth 2.0 to Access Google APIs. code.google.com/accounts/docs/OAuth2, 2012.

[8] G. Ciaccio and M. Ribaudo. Open Data for the Masses: Unleashing Personal Data into the Wild. In *8th International Conference on Web Information Systems and Technologies (WEBIST 2012)*, pages 201–206. SciTePress, June 2012.

[9] E. Hammer-Lahav. Introducing OAuth 2.0. hueniverse.com/2010/05/introducing-oauth-2-0/, 2010.

[10] F. Falcão-Reis and M. E. Correia. *Patient Empowerment by the Means of Citizen-managed Electronic Health Records*, pages 214–228. Number 156 in Studies in Health Technology and Informatics. IOSPress, June 2010.

[11] A. Howard. *Data for the Public Good*. Strata: Making Data Work. O'Reilly, 2012.

[12] IETF. The OAuth 2.0 Authorization Framework. tools.ietf.org/html/draft-ietf-oauth-v2-31.

[13] IETF. The application/json Media Type for JavaScript Object Notation (JSON). www.ietf.org/rfc/rfc4627, July 2006.

[14] IETF. The Base16, Base32, and Base64 Data Encodings. www.ietf.org/rfc/rfc4648, Oct. 2006.

[15] IETF. The Atom Publishing Protocol. tools.ietf.org/html/rfc5023, 2007.

[16] IETF. The OAuth 1.0 Protocol. tools.ietf.org/html/rfc5849, 2010.

[17] IETF. OAuth 2.0 Threat Model and Security Considerations. tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel-07, Aug. 2012.

[18] IETF. The OAuth 2.0 Authorization Framework: Bearer Token Usage. tools.ietf.org/html/draft-ietf-oauth-v2-bearer-23, Aug. 2012.

[19] JOSE Working Group. JSON Web Encryption (JWE). tools.ietf.org/html/draft-ietf-jose-json-web-encryption-05, July 2012.

[20] JOSE Working Group. JSON Web Signature (JWS). tools.ietf.org/html/draft-ietf-jose-json-web-signature-05, July 2012.

[21] K. Schwab *et al.* Personal Data Ownership: The Emergence of a New Asset Class. Report, World Economic Forum, Jan. 2011. Retrieved Dec. 2013 from www3.weforum.org/docs/WEF_ITTC_PersonalDataNewAsset_Report_2011.pdf.

[22] OAuth Working Group. JSON Web Token (JWT). tools.ietf.org/html/draft-ietf-oauth-json-web-token-06, July 2012.

[23] B. Obama. Transparency and Open Government, 2009. Retrieved Dec. 2013, from www.whitehouse.gov/the_press_office/TransparencyandOpenGovernment.

[24] D. Recordon and D. Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM.

[25] N. Shadbolt. *Midata: Towards a Personal Information Revolution*, pages 202–224. IOSPress, June 2013.

[26] A. Tolstenco. Architetture web per "smart disclosure": il caso della dichiarazione dei redditi. Master's thesis, DIBRIS, Università di Genova, July 2013.

[27] UMA Work Group. User-Managed Access (UMA) Profile of OAuth 2.0. http://tools.ietf.org/html/draft-hardjono-oauth-umacore-06, Dec. 2012.

[28] W. Hoffman *et al.* Rethinking Personal Data: Strengthening Trust. Report, World Economic Forum, May 2012. Retrieved Dec. 2013 from www.weforum.org/reports/rethinking-personal-data-strengthening-trust.

[29] D. Wollman. NIST Green Button Presentation — Department of Energy. energy.gov/downloads/nist-green-button-presentation.