Enhancing Cloud based Data Platforms for Smart Cities with Authentication and Authorization Features

Philipp Lämmel
Fraunhofer Institute for Open Communication Systems
Berlin, Berlin, GER
philipp.laemmel@fokus.fraunhofer.de

Nikolay Tcholtchev
Fraunhofer Institute for Open Communication Systems
Berlin, Berlin, GER
nikolay.tcholtchev@fokus.fraunhofer.de

Ina Schieferdecker
Fraunhofer Institute for Open Communication Systems
Berlin, Berlin, GER
ina.schieferdecker@fokus.fraunhofer.de

ABSTRACT

The protection and securing of data platforms and related services plays a major role in the development of safety critical infrastructure for Smart Cities. Therefore, this paper specifies and develops an *Integerated Component for Cloud Services (ISCS)* that enables secure and trusted access to data and related services in the cloud. That means that the ISCS controls and handles access-related aspects such as authentication, authorization and registration. Furthermore, it is deployed and used in a large scale research project, in which it secures cloud services relating to electric mobility. ISCS is realized using OAuth and OpenID, whereas both are implemented by existing open source libraries. OAuth is a standard allowing services and applications to access safety critical resources in the cloud using a trustworthy infrastructure. OpenID is a popular standard originating from the web community, facilitating cross-domain authentication for portal users. The combination of both standards provides rich functionality, covering substantial aspects related to cloud service protection and security.

CCS CONCEPTS

• **Security and privacy** → **Authentication**; **Access control**; **Authorization**; • **Computer systems organization** → *Cloud computing*;

KEYWORDS

Authentication, Authorization, Open Data, Smart City

ACM Reference Format:
Philipp Lämmel, Nikolay Tcholtchev, and Ina Schieferdecker. 2017. Enhancing Cloud based Data Platforms for Smart Cities with Authentication and Authorization Features. In *Proceedings of UCC '17: 10th International Conference on Utility and Cloud Computing Companion (UCC'17 Companion).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3147234.3148087

1 INTRODUCTION

The protection of access to communication network services is relevant for their general safety and reliability. Thus, authentication and authorization of users play an important role. The term authentication means that an individual identifies himselfs unambiguously and evidentiary. Typically, a username and a password are used for authentication. Authorization describes the process of checking whether a user has access rights to a specific resource. Thus, authorization presupposes authentication. In recent years, more and more applications - granting users mobile access to services, e.g. via Internet - have been deployed. This implies the question whether it is reasonable to store login credentials in multiple applications. OAuth has been developed to evade this situation.

Authorization protocols like OAuth provide a process, allowing that no login credentials (username and password) need to be stored in an application locally. As a result, the confidentiality of login credentials and therefore their security is increased. For this reason, OAuth was selected as the used protocol in Fraunhofer *GeMo* project [2] to secure the provided cloud services. The aim of the project, among other things, is to extend car-sharing- service infrastructure in a manner, such that the provided services can be utilized by any user in a secure and mobile way. The secure usage will be guaranteed by OAuth.

A shared cloud infrastructure is the most important prerequisite for the realization of joint mobility, provided in context of the GeMo project in the form of smartphone- and car OBU (on-board unit)-applications. As mentioned, authorization presupposes authentication of the user in question. As a part of the GeMo project, the OpenID decentralized authentification system is used. This leverages the advantage that the user identifies himself to a supporting system by providing a URL after registering to an OpenID provider once. The present paper presents a component developed in the GeMo project, which combines OAuth and OpenID to permit secure access on services, respectively for applications and users. The services are running in a cloud environment developed and operated in the course of the GeMo project. The *Integrated Component for Securing Cloud Services* will be subsequently referred to as *ISCS*.

The rest of this paper is organized as follows: Section 2 presents the high-level requirements which were driving the design of the proposed solution towards extending urban data platforms. The following section 3 presents the overall architecture of the data cloud developed within GeMo as well as the proposed ISCS component and illustrates the interactions that enable the integration of Smart City data platforms with the proposed ISCS solution. Section 4 gives

UCC'17 Companion, December 5–8, 2017, Austin, TX, USA

ACM ISBN 978-1-4503-5195-9/17/12…$15.00
https://doi.org/10.1145/3147234.3148087

an overview of the realized testing and the resulting performance measurements which were conducted on the developed research prototype. Section 5 gives an overview of related authentication and authorization systems, whereas the last section draws conclusions as well as drafts future research and development directions.

2 CAPTURING HIGH LEVEL REQUIREMENTS

This section presents some basic high-level requirements, which were captured in the course of developing the proposed component. The requirements originate from various discussions and from the fundamental needs of the research project (*Fraunhofer GeMo*), for which the ISCS component was developed. Obviously, we needed a piece of software service that can secure a number of flexible open cloud services in the context of electric mobility, which posed the demand to come up with an independent component that enables the authorization of end users as well as of applications running on various devices - such as on-board-units, smartphones as well as other cloud services in general. Hence, the component was to be realized as a standalone service, which can be reached over the Internet (i.e. in general over IP technology) and provides open interfaces allowing various clients[1] to connect. In addition, a number of requirements stem from the need to enable end users to use the emerging component, in order to access portals and services that are utilized via a user interface. Based on such considerations, the following tangible list of requirements was defined.

Requirement 1 Clients that want to utilize ISCS-protected cloud services should have the possibility to register for receiving the rights to interact with the desired cloud services. The registration should be initially done by a user and can be realized using any suitable user interface, e.g. web form.

Requirement 2 All the information required for registering a client should be persistently stored within the ISCS component.

Requirement 3 It should be also taken into consideration that clients are able to register only once with the ISCS.

Requirement 4 End users should also be able to register with the ISCS. The registration and further authorization and authentication of end users is to be realized in a way that a cross-service access is possible (e.g. via OpenID), which means that the user should register and enter his credentials only once, and should be able to access multiple services within a particular session.

Requirement 5 Similarly as in Requirement 2, all the information required for registering an end user should be persistently stored within the ISCS component.

Requirement 6 Similarly as in Requirement 3, it should be taken into consideration that users are able to register only once within the ISCS. That is, duplicate registrations should be avoided.

Requirement 7 The developed ISCS component should ensure that each client is authorized by an end user before accessing protected cloud services in a session.

Requirement 8 In-line with Requirement 7, each user should be authenticated, in order to gain the capabilities to authenticate a client. This ensures that end users are properly involved in the process of authorization and authentication of clients.

Requirement 9 In-line with Requirement 7 and Requirement 8, a client should be authenticated before gaining full access to a service.

Requirement 10 The ISCS component should check whether all required information for a registration is indeed present and should validate this information when required, i.e. before registering a client or an end user.

Requirement 11 The Identity Management of the ISCS component should be easy to integrate in state-of-the-art Smart City related systems, such as CKAN, Liferay and others. This implies that established and recognized standards should be used as guidelines for realizing the interfaces of the emerging component.

The above consideration give a high level view of key requirements on the emerging ISCS component. These requirements drive the architectural design and the implementations in the coming sections and ensure that the proposed solution is indeed usable in the context of urban platforms for smart cities.

3 CLOUD BASED DATA AND SERVICE ARCHITECTURE

In this section both the architecture of the City Data Cloud (CDC) and the Integrated Component for Securing Cloud Services (ISCS) will be presented.

3.1 City Data Cloud

The City Data Cloud (CDC) aims to provide versatile, mobility-focused datasets via a centralized platform. Various data sources, like map resources or governmental road data are integrated and available via the CDC and therefore, diverse applications and services can make use of this data. An exemplary application could be an embedded application in an On-Board Unit or a web application accessed via a web browser. The CDC consists of the components *1) Data Portal, 2) Data Registry, 3) Data Store, 4) Triple Store* and *5) Service Layer* and the CDC's architecture can be seen in figure 1 [18]. The components 1,2 and 3 form the Open Data[2] Platform of Fraunhofer FOKUS [14]. The Open Data Platform originates from the Open Cities project [6], in which the CDC can be seen as an extension of this open source approach. The particular modules of the CDC are described below.

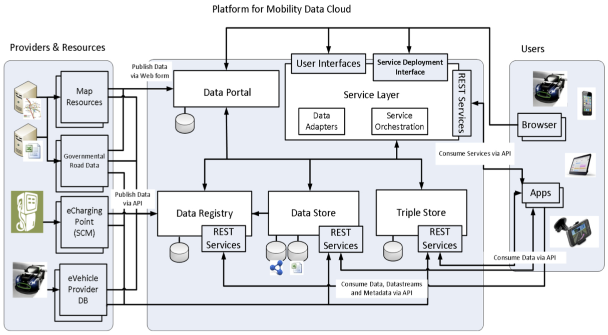

Figure 1: Architectural view on the CDC [18]

[1] Clients are meant to be different services or applications that utilize the component developed in the presented work.

[2] Data is considered open, if it is freely usable and distributable by anyone who complies to the authors regulations, for example in terms of attribution and dissemination (share-alike) [1].

1) Data Portal

The Data Portal is a component that is derived from the Open Data Platform. It presents the data of the CDC to potential users, for instance app-developer. In addition, it provides the possibility of adding new data and data sources to the CDC. In this regard it is mentionable that the CDC can store data within its architecture as well as references to external sources. In addition, the portal provides social media features like a commentary function and a discussion board [14, 18].

2) Data Registry

The Data Registry is based on CKAN and is responsible for the indexing and administration of data sources within the CDC, and thus it stores the metadata entries, describing the data available in the CDC. This applies to data referenced externally as well as data stored directly within the architecture [18].

3) Data Store

Arbitrary data is stored in the Data Store component. For this purpose, a REST-Interface is provided on top of the underlying database. The component can be either realized by a NoSQL database [13] or via an open REST interface on top of the underlying database. This interface can be used by applications and services to determine the data interchange format dynamically [18]. An implementation of this component has been carried out as a part of [11].

4) Triple Store

The Triple Store gives the opportunity to store data within the CDC. Ontology-based data can be stored and made available for using a REST interface (for example to query a SPARQL endpoint) [18].

5) Service Layer

With the aid of the Service Layer, mobile services and applications that are based on data available in the CDC can be deployed. For this purpose, a Service Deployment interface has been developed, which allows developers to add new and update existing services [18]. Thereby, the openness of the platform in terms of PaaS is implemented [18]. The mentioned services are not limited to the usage of data stored within the CDC, they can use data from external data sources, which have been indexed by the Data Registry, too. As a consequence, the implementation of adapter has to be considered eventually, which transform the raw data into the required format. Services can provide specific user interfaces, for instance in the context of web-based mobile applications, which are accessible via the HTTP protocol as part of the service layer, as well. Furthermore, these services provide a REST interface which grant access to JSON- and XML-based APIs. The combination of services is also possible so that complex services emerge. This could be realized by using a Services Orchestration Module and in this case, services are chained up and in- and outputs are transferred between adjacent services.

3.2 Integrated Component for Securing Cloud Services

In this subsection the particular components of the ISCS are presented regarding their functionality. figure 2 shows the architecture of the integrated component. The left-hand side shows the Open Data Platform of Fraunhofer FOKUS, which is the basis of the CDC, and the clients. The component itself and its modules are shown in the center, while the database, accessed by the integrated component, is shown on the right-hand side.

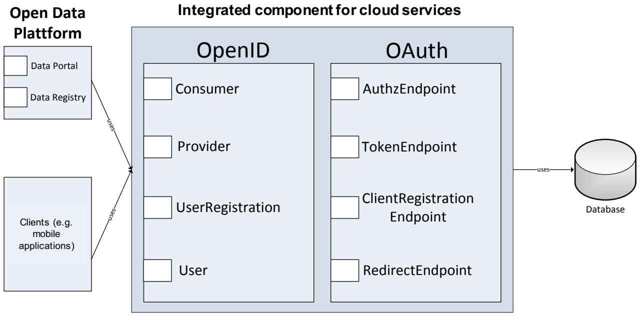

Figure 2: Architectural view on the developed ISCS

As a consequence of requirement 1,2,4 and 5 there has to be a component for the registration of users and clients respectively. To create a solution as modular and maintainable as possible, these two components have to be independent and separate from each other. The modules *UserRegistration* and *ClientRegistrationEndpoint* are designated for the registration of users and clients and are shown in figure 2. The figure also illustrates the independence of the components from one another.

Referring to requirement 7, a component is required that authorizes the clients. The *AuthzEndpoint* is designated for that purpose. Given that the authentication of the user is presupposed for authorization, an appropriate component is required as well. For that purpose, OpenID is used which is a well established community standard. Thereby, the *Consumer* and *Provider* modules are in charge of performing authentication. These modules should be independent from the aforementioned modules as well. The *User* module sends an XRDS (Extensible Resource Descriptor Sequence) document that identifies the *Provider* in return for a request. Furthermore, a module determining and validating access rights needs to be available. This task is executed by the *TokenEndpoint* within the OAuth component. Finally, the OAuth component encompasses a *RedirectEndpoint* that serves as an alternate endpoint for clients without an own redirect URI. That guarantees that the data is resent to the requesting client. The requirements 2 and 5 imply that all data is persisted. Therefore, a database (see figure 2) that allows independent access is required. The underlying database should be also interchangable, which can be achieved by the application of a persistence framework like Hibernate [3].

Figure 3 shows the typical requests to the ISCS. the topmost sequence shows a request for authorization. In this request, a client sends a request along with the required parameter to the AuthzEndpoint, which belongs to the OAuth-subcomponent (see message *requestAuthcode()* in figure 3). In the next step the transmitted information are validated (see message *validateInput()* in figure 3) and also checked for validity. Subsequently, the client needs to be authorized by the user (see message *askUserForAuthorization* in figure 3). After the authorization is completed, an authorization code is issued and saved in the database (see message *saveAuthCode()* in figure 3).

The next sequence shows the exchange of an access token and a refresh token against a previously received authorization code (see message *requestToken()* in figure 3). The authorization code is send

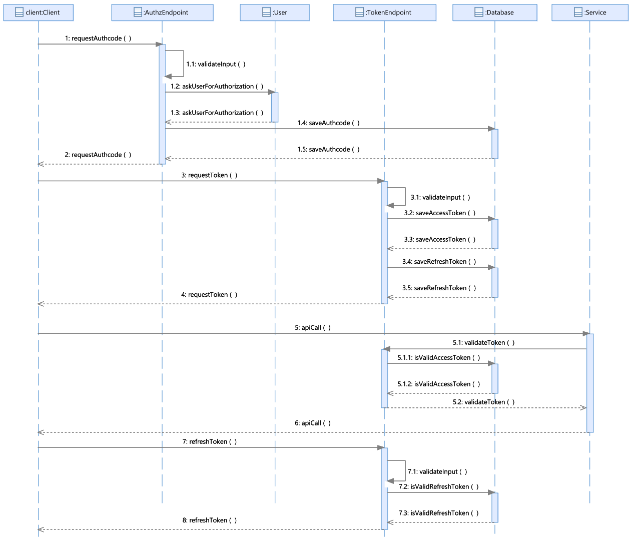

Figure 3: Illustration of typical requests

to the TokenEndpoint, which belongs to the OAuth-subcomponent, and is exchanged with the mentioned tokens in case of a valid authentication. The generated tokens are additionally persisted in the database (see message *saveAccessToken()* and *saveRefreshToken()* in figure 3). In this way requirements 7,8 and 9 are fulfilled, addressing the authorization of a client corresponding to the specification of OAuth.

The next sequence shows the request to an access protected service (message *api_call()* in figure 3). At this point, the ISCS validates the received parameters for the service (message *validateToken()* and *isValidAccessToken()* in figure 3) and thereby fulfills requirement 10.

Access token have a restricted validity, and thus clients have to use the provided refresh token in order to get a new access token. This process is realized by the message *refreshToken()* shown in figure 3). Thereby, the input is validated again and the refresh token is checked for its validity (message *validateInput()* and *isValidRefreshToken()* in figure 3).

Finally, you can login to the Open Data Platform via OpenID. This procedure does not inhere further effort since the platform is based on Liferay, which is delivering OpenID by default. For that purpose, the ISCS would start the OpenID protocol and the user could use his credentials to log in. Requirement 11 is thereby fulfilled.

4 EVALUATION: MEASUREMENT RESULTS

In this section, the Integrated Component for securing cloud Services (ISCS) will be evaluated. Due to the compactness of this paper, the presentation will focus on performance tests, because we think that these measurements provide the most insights. During the development of the ISCS, the authors did a lot of dynamic testing of the system by means of unit tests, integration tests and fuzz tests. Dynamic testing helps to investigate the behavior of the system. The software is executed and input data is sent to the system. Afterwards, the output of the system is checked for validity. Unit tests perform isolated tests on a subsystem with respect to the required and expected functionality. In the context of integration testing, these subsystems are grouped and tested in combination. Fuzz testing describes a kind of security testing by sending invalid data to the system. In this case the robustness and stability of the ISCS is investigated.

More than 10000 tests were implemented and generated and ensured the correctness as well as robustness of the emerging solution. The performance of the component was investigated and will be discussed in the next subsection.

4.1 Performance Tests

Performance tests have been executed in order to evaluate the developed ISCS with respect to non-functional requirements such as high performance and low resource usage. Various integration and fuzzing tests were selected and adapted to measure runtime and memory usage.

4.1.1 Measuring the Execution Time. The results are shown in figure 4. Four typical scenarios are displayed which represent frequently executed use cases of the ISCS (e.g. ClientRegistration). The results are shown as a boxplot. 120 samples were created for the request of an access token, while 220 samples were created for the request of a refresh token and 400 samples were created for the remaining requests. With more than 100 test data sets for each use case, the sample is representative. As illustrated in figure 4, the median of the response time is below 30 ms for every single request. At the same time 75% of the data is below 32 ms. Thus, the result can be considered as good. At the same time it has to be mentioned that the input data for the deployed fuzz tests is often invalid, which means that the ISCS does not access the database in some of those samples. Nevertheless, these tests and obtained measurements give a reasonable indication for the ISCS' performance. Furthermore, these tests validated the system in terms of changes in behavior. It can be concluded that the system operates correctly and with good performance under data fuzzing as well as valid requests.

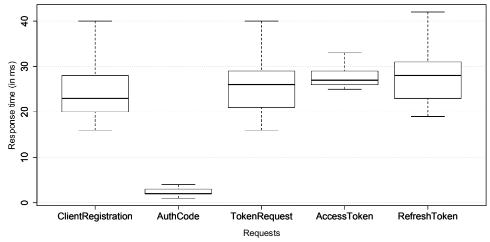

Figure 4: Performance measurements regarding typical requests to the ISCS

According to a study performed by Jako Nielsen, the response time should lie between 0.1 and 1.0 second, giving the user the feeling of immediate response [17]. Within this timespan the user's train of thought is not disrupted, avoiding the need for sending further feedback [17]. However the user would still notice a delay which is why it should be pointed out that the measurements rarely exceed a value of 300 ms. The performance of the ISCS can be considered as very well due to these results.

4.1.2 *Memory Usage.*

In this subsection the memory usage of the ISCS is investigated. Test cases refering to the runtime measurement were adapted in a way, such that the usage of the heap memory for the ISCS component is displayed. The Java Virtual Machine has been started with different heap size parameter values, taking a value of 64 MB, 128 MB and 256 MB. Again fuzz tests were used in the first step. The results are shown in figure 5 and figure 6. The observed memory usage is typical for a java application, raising to a certain point before the garbage collector is called and releasing unused memory space after each periodic garbage collector invokation. The resource usage is considered to be sufficiently small. Memory is used as long as it is available, without ever becoming excessively large, which in turn would be an indication for a memory leak. It can be assumed that the developed solution is free of memory leaks or other greedy sub-optimal code fragments.

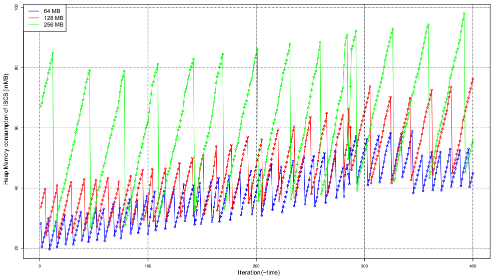

Figure 5: Token Request

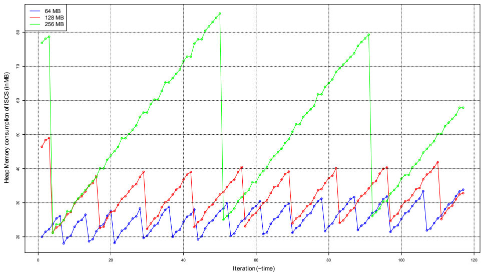

Figure 6: Validating an Access Token

5 STATE OF THE ART

Having presented the way data platforms for Smart Cities can be enhanced as to protect and secure the data and corresponding services, we proceed with a state-of-the-art review of technologies which are related to the area of authentication and authorization in distributed systems. These technologies can all play a key role when designing security solutions for urban data platforms.

Kerberos is a very popular legacy system for realizing authentication and authorization in distributed systems. Currently, it is available in its version 5 and was originally developed by the Massachusetts Institute of Technology (MIT) [16]. Kerberos is meant to provide an infrastructure for authentication on top of an insecure network. It is partially based on the protocol of Nedham and Schroeders for authentication [15] as well as on further developments, which were proposed by Denning and Sacco in [10]. Thereby, the Kerberos architecture uses a centralized instance that issues tickets to the entities needing authentication/authorization for access to network resources.

Mozilla Persona, originally known as *BrowserID*, is another system of general interest and was initially developed by the Mozilla Foundation [5]. It implements a decentralized authentication procedures that allows the access to multiple websites based on a one-time authentication using an email address. The system security is based on an asymmetric cryptographic system utilizing digital signatures. After login, a so-called Identity Assertion is being created, which contains the email address of the user in question. This Identity Assertion is signed with the private key of the user in question and subsequently sent to the corresponding resource hosting entity (e.g. server), which in turn requests the verification of the assertion to be conducted by an Identity Provider.

OpenID [7] is probably the most popular technology in that field. It is also one of the technologies utilized in the current work. Since it usage was widely exemplified in the current work, we are not going to further ellaborate but just mention OpenID for the sake of completeness.

The Security Assertion Markup Language (SAML) was developed by the Services Committee of the Organization for the Advancement of Structured Information Standards (OASIS) [6] [8]. SAML is an XML based framework, which enables both the authentication and authorization of users. Based on SAML, it is possible to conduct various assertions based on single identities, attributes, and access rights.

Shibboleth was developed by Internet2/MACE as an authentication and authorization system for webservices [9]. The goal is to enable the one-time authentication of users, which in turn can use the services of various providers. Shibboleth is based on an extension of the previously presented SAML standard [8] [9]. Similarly, Shibboleth involves three different parties: an Identity Provider, a Service Provider and an optional Discovery Service [4]. The task of the Identity Provider is the management and verification of user identities, whilst the Service Provider is in charge of securing the belonging web service. By utilizing the Discovery Service, it is possible to establish connections to various Identity and Service Providers. These components can be operated independently and can dynamically support and facilitate the authentication and authorization processes.

Finally, for the sake of completeness, we should mention the second key technology used in this work - OAuth 2.0 [12]. It is a widely used standard for cross-service authentication and authorization, which is supported by key players on the market (e.g. Facebook and Twitter). Since the usage of OAuth 2.0 for the purpose of enhancing cloud based urban data platforms was widely exemplified within the current paper, the authors decided to omit further details of the standard.

The current section has elaborated on technologies with respect to the broader picture of solutions and methods for enhancing Smart City data platforms with security features. These approaches can be used in similar works to achieve the goal of providing services in a secure way.

6 CONCLUSIONS AND FUTURE WORK

In this paper an Integrated Component for securing Cloud Services (ISCS) has been presented. The component is based on the IETF standard *OAuth* and the community standard *OpenID*. It has been developed in context of the Fraunhofer GeMo project to guarantee the secure access of services, respectively applications and their users, to the cloud infrastructure used within GeMo. This cloud infrastructure is a typical example for a data platform for Smart Cities with corresponding services running on top of it, adding value to the data and presenting it to end users in a consumable way. Hence, the paper presents a concept that allows to enhance a cloud based data platform for Smart Cities with authentication and authorization features.

The presented design is based on a set of high-level requirements, which are subsequently used to drive the specification of an architecture and belonging interaction flows for the ISCS. Based on these specifications, a prototype implementation of the ISCS was developed and evaluated in terms of stability and performance, as well as memory usage. The obtained results show that it is possible to realize the ISCS in a way that it responds fast and efficiently despite the multiple interactions required for achieving its goals. Furthermore, the research prototype shows some moderate memory usage and increased robustness based on the employed testing techniques.

Our future work will move into the direction of deploying the ISCS component in various research projects on European level as well as in different industrial projects. Furthermore, we believe that aspects such as billing and accounting for the usage of urban data and services can be further integrated into the ISCS. Hence, this would be another direction requiring some more specific investigation in the coming years.

REFERENCES

[1] 2017. Definition of Open Data. http://opendefinition.org/. (2017). last accessed: October 13, 2017.

[2] 2017. GeMo. https://www.gemo.fraunhofer.de/en.html. (2017). last accessed: October 13, 2017.

[3] 2017. Hibernate. http://hibernate.org/. (2017). last accessed: October 13, 2017.

[4] 2017. How Shibboleth Works: Basic Concepts. http://shibboleth.net/about/basic.html. (2017). last accessed: October 13, 2017.

[5] 2017. Mozilla Persona. https://developer.mozilla.org/en-US/Persona. (2017). last accessed: October 13, 2017.

[6] 2017. Online Community for the Security Assertion Markup Language (SAML) OASIS Standard. http://www.saml.xml.org. (2017). last accessed: October 13, 2017.

[7] 2017. OpenID Authentication 2.0 - Final. http://openid.net/specs/openid-authentication-2_0.html. (2017). last accessed: October 13, 2017.

[8] 2017. Security and Privacy Considerations for the OASIS Security Markup Language V2.0. http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf. (2017). last accessed: October 13, 2017.

[9] 2017. Shibboleth. http://shibboleth.net/. (2017). last accessed.

[10] Dorothy E Denning and Giovanni Maria Sacco. 1981. Timestamps in key distribution protocols. *Commun. ACM* 24, 8 (1981), 533–536.

[11] Benjamin Dittwald. 2012. *Eine webbasierte Infrastruktur zur Speicherung und Abfrage von strukturierten offenen Daten*. Master's thesis. Freie Universität Berlin.

[12] D. Hardt. 2012. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard). (Oct. 2012). http://www.ietf.org/rfc/rfc6749.txt

[13] M. Indrawan-Santiago. 2012. Database Research: Are We at a Crossroad? Reflection on NoSQL. In *Network-Based Information Systems (NBiS), 2012 15th International Conference on*. 45–51. https://doi.org/10.1109/NBiS.2012.95

[14] Evanela Lapi, Nikolay Tcholtchev, Louay Bassbouss, Florian Marienfeld, and Ina Schieferdecker. 2012. Identification and Utilization of Components for a Linked Open Data Platform. *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops* 0 (2012), 112–115. https://doi.org/10.1109/COMPSACW.2012.30

[15] Roger M. Needham and Michael D. Schroeder. 1978. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM* 21, 12 (Dec. 1978), 993–999. https://doi.org/10.1145/359657.359659

[16] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. 2005. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard). (July 2005). http://www.ietf.org/rfc/rfc4120.txt Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649, 6806.

[17] Jakob Nielsen. 1994. *Usability engineering*. Elsevier.

[18] Nikolay Tcholtchev, Benjamin Dittwald, Thomas Scheel, Begüm Ilke Zilci, Danilo Schmidt, Philipp Lämmel, Jurma Jacobsen, and Ina Schieferdecker. 2014. The Concept of a Mobility Data Cloud: Design, Implementation and Trials. In *IEEE MidArch - 8th IEEE International Workshop on Middleware Architecture in the Internet co-located with COMPSAC 14*.