




# PAUDIT: A Distributed Data Architecture for Fitness Data

Jaco du Toit<sup>(✉)</sup> 

University of Johannesburg, Johannesburg, South Africa  
jacodt@uj.ac.za

**Abstract.** Centralized data architectures are the predominant architectures used by systems today. Decentralized data architectures, making use of data link networks, allow users to store private data in personal online data stores (POD). When fitness data is stored in a POD, the user has full control over the data and may modify fitness data. Medical aid providers that makes use of fitness data for benefit calculations cannot trust fitness data in existing POD architectures. PAUDIT is an architectural model that describes a decentralized data architecture that audits changes to access control lists on POD servers. The audit information allows medical aid providers to verify if users may have changed fitness data. PAUDIT describes an architecture that allows medical aid providers to verify the validity of fitness data.

**Keywords:** Privacy · Decentralized · Audit · Permissions · Fitness · Architecture

## 1 Introduction

Fitness tracking devices measure various exercise features. These features include the number of steps taken within a period, heartrate, speed and even weight. The exercise features are normally transferred from the fitness device using an application to a smart phone or computer and sent to a remote server. The fitness application, running on the smart phone or computer, allows the user to get information regarding their fitness or activity levels and may even provide certain gamification features to entice users to exercise more [1].

Several insurance companies provide incentive programs to customers that share their fitness data with the company. The incentive program has been adopted in the medical aid industry where it was proven that medical aid members that is physically more active has lower medical aid costs [2].

It is important for the medical aid providers that the data collected from their customers was not modified and represents a fair reflection of the customer's activity and fitness levels [3]. Unfortunately, this also provides an incentive to the customer to try and manipulate the data. If the data can be changed to indicate a higher level of fitness level, then the customer can directly benefit from the data modification.

Another aspect of fitness data is that it is undeniably personal information. Regulations, such as defined by the European Union General Data Protection Regulation (GDPR), require organizations that work with private data to ensure several controls

[4]. Companies that store personal information require an investment in GDPR-based controls.

One data architecture that reduces the required GDPR-based controls is a distributed data architecture as implemented by the Solid project [5, 6]. The Solid project proposes the use of personal online data stores (PODS). Each user stores their personal information in a POD. The user has full control over the POD. When the user uses an application, the data never leaves the POD, instead it is only linked to by the application.

The problem that may arise when fitness data is stored inside a POD is that the owner of the POD has, by default, full permissions to modify the data in the POD. The owner of the POD can also modify access control of data, assigning and removing permissions to the data. The ability to modify the access control is referred to as control in this paper. The control property in a POD is a problem for medical aid providers that require data integrity of fitness data.

An abstracted version of the problem statement can be formulated as: *In a distributed data architecture, the control property belongs to the POD owner. Certain types of personal information may require oversight from a third party, to ensure that the owner do not modify data when it is stored in a distributed data architecture.* In this paper the abstracted problem statement is quantified to apply specifically in cases of fitness data inside a specifically implemented distributed data architecture.

This paper proposes an architectural model, called PAUDIT. PAUDIT describes an architectural model that allows data generated by fitness trackers to be stored in a POD. PAUDIT further describes the ability of a medical aid provider to read the fitness data in a POD and monitor audit logs which keeps track of changes to access control lists. PAUDIT provides information to a medical aid provider that can be used to determine if the user modified fitness data. The audit component of PAUDIT has been implemented as a prototype and the prototype design and testing is discussed in this paper. PAUDIT applies existing theory, mechanisms, and aspects to a unique problem domain that does not see adoption from industry. It may be argued that the problem described in this study may act as a deterrent for industry adoption of distributed data architectures.

This paper is organized as follow. Section 2 discusses the methodology followed during the study. Section 3 provides background information on distributed data architectures, specifically the implementation proposed by the Solid project. Section 4 describes the PAUDIT architectural model that allows fitness data to be stored on a POD, but still establishing a level of trust using audit logs. Section 5 describes the prototype implementation and testing of the audit component of the PAUDIT model. Section 6 discusses some of the lessons learned and future research resulting from the research conducted for this paper.

## 2 Research Methodology

The research presented in this study follows the design science research paradigm. Design science concerns itself with the creation of “purposeful artifact for a specified problem domain” [7]. The artifact created in this study is a model and is called

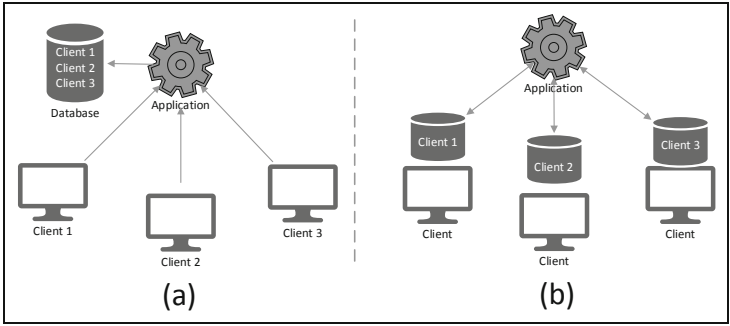
PAUDIT. The problem domain in which PAUDIT plays a role is a distributed data architecture designed to facilitate the storage of personal information in a POD.

The research presented in this study follows six activities defined by Peffers et al. [8]. Each of the six activities is discussed below, as well as a where these activities are applied in this paper [8]:

1. **Problem identification and motivation.** Section 1 described the problem with fitness data that may be stored in a distributed data architecture. The problem as, specified in Sect. 1 is that private information stored in a POD, may require oversight from a third party, to ensure that any modification of the personal information is audited.
2. **Objectives of the solution.** The solution has as set objective. The objective is to create a model, based on requirements defined in both Sects. 1, 4 and 5. The requirements that are generated to specifically address the problem defined in Sect. 1.
3. **Design and development.** The artifact created in this study must be designed and developed. Sections 4 and 5 discusses the creation of the PAUDIT model, as well as the specific audit module.
4. **Demonstration.** To ensure the model solves the problem a prototype of the audit module was developed and implemented on an existing distributed data architecture framework. Section 3 discusses the background of distributed data architectures with the prototype implementation discussed in Sect. 5.
5. **Evaluation.** The audit module goes through a testing phase in Sect. 5.3. The testing is performed on the prototype implementation of the PAUDIT model, with the focus on the integrity aspect of the personal information.
6. **Communication.** The last activity in design science is the communication of the problem. The communication activity concerns itself with communicating the study to a relevant audience. Communication may occur when presented as a paper or at conferences [8].

### 3 Background

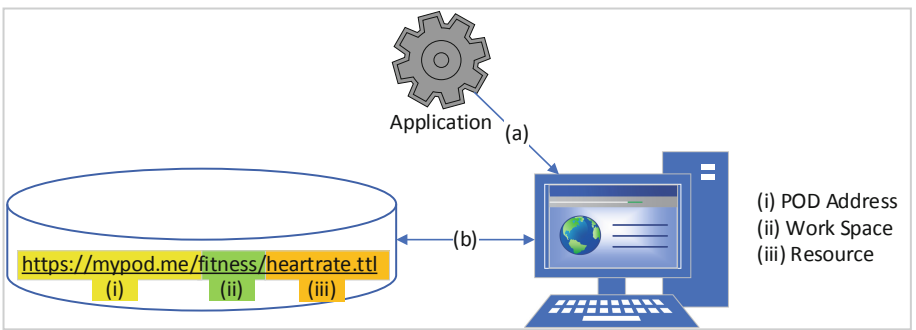
From an application's perspective, a user's personal information is usually stored in a centralized database (a in Fig. 1). Another architectural design is where the data is decentralized from the applications perspective (b in Fig. 1). The decentralized approach can also be called the sovereign approach. In the sovereign approach, each user has sovereign power of his/her personal data. The approach introduces the aspect of a personal online data store (POD). This section describes the existing standards and technologies that enables a decentralized data architecture.



**Fig. 1.** Data storage architectures in (a) a typical centralized data architecture and (b) decentralized data architecture (By author).

The decentralized data architecture makes use of standards and concepts originally defined by the semantic web [9, 10]. The semantic web proposes a “web of data” instead of the existing “web of documents”. The “web of data” can be applied to personal information, that is stored on the Internet and only linked to by applications. Solid proposes a POD to store personal information on the Internet.

The scenario depicted in Fig. 2 is used to describe the various components in a data linked network. A hypothetical data linked application that makes use of the sovereign approach is described in Fig. 2. The developers of the application host the application on a server on the Internet. The user opens a web page and navigates to the web site of the application. The application may be a JavaScript-based application, which are (a in Fig. 2) executed by the user’s Internet browser. When the application executes the user is required to login. The login credentials, provide by the user, consist of a WebID [11], which helps locate the user’s POD. The application sends an authentication request to the user’s POD (b in Fig. 2). The application uses the address of the WebID (i in Fig. 2) to get the URL of the POD. After a successful authentication, the application retrieves other components from the POD. The components may include various workspaces (ii in Fig. 2) and application specific resources (iii in Fig. 2).



**Fig. 2.** A data linked application (By author).

A WebID is a standard way in which entities are identified. A WebID may have the following syntax: <https://mypod.me/card#me>. The #me references the entity inside a document, which, in the example, is the user. At this point in time, authentication of a WebID is supported using either a username and password, or a client-side X.509 certificate-based authentication mechanism, called WebID-TLS.

Access to resources and workspaces in the POD is controlled by the POD server using WebAccessControl [12] and Cross-Origin Resource Sharing (CORS) [13]. After successful authentication, the application may access the resources in the POD through CORS.

WebAccessControl define four modes of access. The modes are Read, Write, Append and Control. The modes are defined in access control lists and may be implemented as .acl – files. Read, write, and append allow the specified subjects to read, write or append resources. The control mode specifies the subjects that can set access control lists. This implies that the control permission allows a subject to modify access control lists.

By default, the WebID of the POD owner has read, write, append, and control authorisation to any resources in the POD. It is possible to remove the default permissions on a resource and only allow a specific subject access to a resource. It may even be possible for the WebID of the owner of a POD to remove itself from the control of a resource, effectively loosing control over a resource stored inside the POD.

Fitness data, such as heart rate, is personal information and should be stored in a POD in a decentralized data architecture. A medical aid provider may require access to the fitness data to measure the user's fitness and exercise activities to offer cheaper medical aid rates, or other loyalty program advantages. The next section provides an overview of the PAUDIT model that uses a POD for fitness data and allow a medical aid provider access to the data.

## 4 Overview of Architectural Model

When a medical aid provider uses fitness data for benefit calculations, the provider needs a high level of assurance that the data has not been modified. One of the challenges in a distributed data architecture is the sovereign property of data. This implies that a user has full control over all the data stored inside a POD.

To help define an architecture that solves the above problem a list of requirements is defined. The requirements are:

1. The user of the data should stay in control of the fitness data and should not give away control to a third party.
2. The user should not have permissions to modify the fitness data after it has been created.
3. The medical aid provider should have read permissions to the fitness data.
4. The medical aid provider should have the ability to verify any access control changes that might have occurred over time on the fitness data. Changes in access control may highlight potential changes in permission that might have allowed the user to modify the fitness data.

The WebAccessControl standard supports the above requirements except for requirement four. An implementation of the WebAccessControl in an access control list is displayed in Fig. 3. The access control list depicted in Fig. 3 is applied on a Fitness folder inside the user’s POD. Fitness data is stored inside the Fitness folder on the user’s POD.

The access control list consists of three sections. (a) in Fig. 3 show that the user has Read and Append permissions on the resource. The read permission allows the user to read the resources in the Fitness folder. The append permission allows the user to append data to the existing set of resources, but it does not allow the user to change any of the existing data. (b) in Fig. 3 shows a medical aid provider’s WebID having Read permissions. The Read permissions allow the medical aid provider the ability to read the fitness data in the folder. (c) in Fig. 3 enable the user to stay in Control of the access control list. The Control permission allows the user to modify the access control list.

```

@prefix : <#>.
@prefix n0: <http://www.w3.org/ns/auth/acl#>.
@prefix A: <./>.
@prefix c: </profile/card#>.
@prefix c0: <https://medicalaid.me/profile/card#>.

```

<pre> :AppendRead   n0:accessTo A;;   n0:agent c:me;   n0:defaultForNew A;;   n0:mode n0:Append, n0:Read. </pre>	(a) Append and read permissions assigned to the user
<pre> :Read   n0:accessTo A;;   n0:agent c0:me;   n0:defaultForNew A;;   n0:mode n0:Read. </pre>	(b) Read permissions assigned to MAP
<pre> :Control   n0:accessTo A;;   n0:agent c:me;   n0:defaultForNew A;;   n0:mode n0:Control. </pre>	(c) Control permissions assigned to the user

Fig. 3. Access control list for fitness data (By author).

The WebAccessControl settings described in Fig. 3 fulfils requirements, one, two and three, but not requirement four. The fourth requirement states that medical aid provider should have the ability see changes to the access control list over time. This implies that if the user should change the access control list and potentially modify the fitness data, the medical aid provider should have the ability to see that the access control list has changed in such a way.

PAUDIT is a model that describes how fitness data is generated on a fitness tracking device, gets uploaded to a POD and allows a medical aid provider to read the fitness data. It also allows the medical aid provider to track changes to access control lists on the POD.

Figure 4 describes the architectural model called PAUDIT. The model consists of six major components. The interaction between the components describe the flow of data from a fitness device into a POD. It further clarifies how the user stays in control of the fitness data while a medical aid provider can verify modification of access control lists which may indicate data tampering.

The six components of the PAUDIT model is described in Fig. 4 and is the following:

- (a) The user makes use of a fitness tracking device. Fitness devices can track various biometric features of the user. This include fitness activities such as heart rate, exercise location tracking, and even sleep patterns and blood glucose levels [14]. Once the device has collected the relevant data the fitness device may interact with a relevant app on a smartphone or computer. Data is usually uploaded from the fitness device to the smartphone using protocols such as Bluetooth [3].
- (b) The app on the smartphone is written to make use of a distributed data architecture. The user is identified on the app using the user's WebID. The app uploads the fitness data into a fitness specific folder on the user's POD. Even though Fig. 4 does not indicate this, the fitness device may also have an app installed that directly uploads the data to the POD, instead of first going through an app installed on a smartphone.
- (c) The POD has been configured with a Fitness folder. The Fitness folder contains several other folders and resources. The sub-folders describe data sorted according to year, month, and day. The fitness data reside inside resource descriptive framework (RDF) files.
- (d) The Medical Aid Provider receives the WebID of the user, when the user registers for the provider's special fitness related program. During the registration process, the Medical Aid Provider request read access to the Fitness folder on the POD.
- (e) The Medical Aid Provider also requests the user to change the permissions on the Fitness folder so that the user only has Read, Append and Control permissions. If the user is denied Write permissions to the Fitness folder, then the Medical Aid Provider is assured that the user cannot modify the existing data.
- (f) One each POD server, a special audit module runs that keeps track of changes in access control lists. Whenever an update to an access control list is made the audit module takes the requested change and records the change in an audit log file.

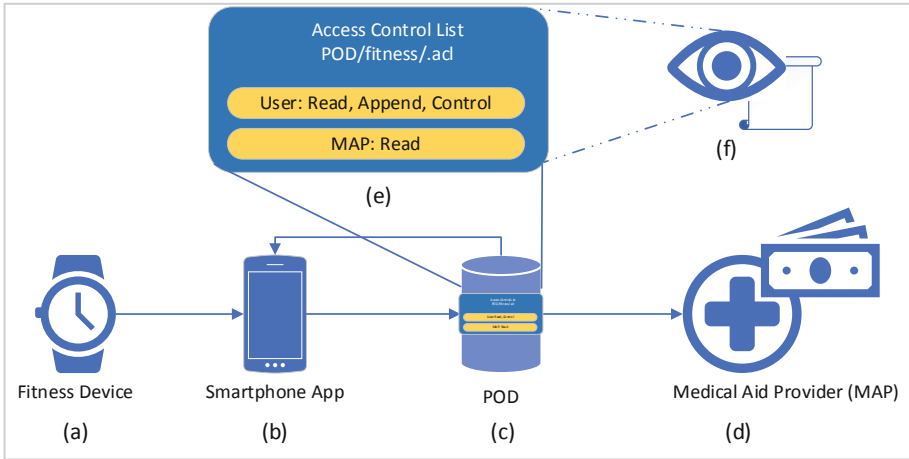


Fig. 4. PAUDIT fitness application architecture (By author).

The PAUDIT model allows a user to use a fitness device and still stay in control of the fitness data generated. What makes the model unique is the audit module that keeps track of changes in access control lists. The change tracking functionality provided by the audit module is not a standard or an accepted feature in existing implementations of a decentralized data architecture.

The next section describes how the audit module works and provide results from a prototype implementation of the audit module.

## 5 Auditing Module

The audit module creates audit log files and stores them in the same folder as the relevant access control lists. The Solid project provides existing POD server software that can be used for testing or production purposes. The existing POD server implementations are available on GitHub as part of a `node-solid-server` repository<sup>1</sup> in the solid organization<sup>2</sup>. No audit feature currently exists in Solid server implementations. A proof of concept audit module was written for testing purposes. The POD servers are referred to as Solid servers from this point onwards, to indicate that the testing was conducted using source from the `node-solid-server` GitHub repository.

This section describes the design, implementation and testing of the audit module in the following corresponding sections.

<sup>1</sup> <https://github.com/solid/node-solid-server>.

<sup>2</sup> <https://github.com/solid>.



## 5.1 Audit Module Design

The design of the audit module defines several design requirements, the content available in the log files and the integration of the audit module into the existing components of the solid server.

The following requirements were defined for the audit module:

1. For each access control list file, a corresponding audit log should exist.
2. Audit logs should have the same filename as the relevant access control list file, but with an .adt file extension.
3. Audit logs should be appended to instead of overwritten.
4. Audit log files should be searchable, and the data in the log should have the capacity to be filtered and ordered.

The content of the audit log should provide enough information so that changes to access control lists can be tracked. The specific information provide by the audit log files are the following:

- **Timestamp.** The timestamp records the date and time on the server when an action occurs.
- **UserId.** The userId field is the WebID of the user that initiates the change in the access control file.
- **Action.** The action field describes the specific node-solid-server handler that fulfils the request. There are two distinct handlers that can modify an access control list. The two handlers are: put and patch.
- **Detail.** The detail field is the data that gets applied by the handler. Depending on the handler, the data is either a just an update to the existing access control file, or the file gets re-written with new content.

## 5.2 Audit Module Implementation

The Solid server uses handlers to modify resources stored in the user's repository. The two handlers that handle changes to existing resources in the POD repository is the put and patch handler.

The put handler overwrites the existing resource with new content. It effectively deletes the existing resource and creates a new resource with new content. Testing have shown that the put handler is used in most cases when access control changes occur.

The patch handler creates, update or delete data in existing resources. It corresponds with typical SQL insert and delete commands, except it affects data in existing RDF resources. The language used to update the resources is called SPARQL. In the tests performed the node-solid-server user interface does not make use of the patch handler to update access control lists, but the handler may be called indirectly by users using operating system commands such as curl [12, 15].

The audit module is implemented as a JavaScript module that gets called by both the put and patch handlers. The put and patch handlers are modified to call the audit module only when the resource in question is an access control list. The audit module receives an object from the calling module describing various properties of the system at the time of calling.

The audit module extracts the properties of the system and creates a relevant .adt file if it is not already created. The timestamp, user, action, and details of the change is appended to the .adt file. The .adt file is a json data file describing the different fields, so that it can easily be interrogated by an external application.

Figure 5 is an extract of the prototype audit log that was created after a change was initiated on an .acl file. Figure 5 highlights the four components created for each log entry. The “timestamp” field was the time in universal time coordinate on the server when the specific action occurred. The “userId” field shows the userId of the subject that initiated the change. “action” highlights the solid server handler that handled the change to the .acl file. The last field, “detail” shows the content of the newly created .acl file in this example.

The logfile displayed in Fig. 5 was created as a uniform resource identifier (URI) with the following path: <https://user.localhost/public/fitness/.adt>. The /.adt filename given to the audit log filename corresponds to the /.acl file which is the corresponding access control list. The relationship between the two files can be used by the Medical Aid Provider to track the changes to the access control file as it occurs over time.

```
{
  "logentries": [
    {
      "timestamp" : "2019-04-22T09:47:19Z",
      "userId" : "https://user.localhost/profile/card#me",
      "action" : "put",
      "detail" : "@prefix : <#>.
                 @prefix n0: <http://www.w3.org/ns/auth/acl#>.
                 @prefix A: <./>.
                 @prefix c: </profile/card#>.
                 :ReadWriteControl
                 n0:accessTo A;;
                 n0:agent c:me;
                 n0:defaultForNew A;;
                 n0:mode n0:Read, n0:Write, n0:Control."
    }
  ]
}
```

Fig. 5. Audit log for a specific.acl file (By author).

Section 5.1 listed four requirements for the audit module. The four requirements together with a comment on the implementation of each requirement is summarized in Table 1. The implementation of the prototype audit module fulfilled each of the original requirements.

**Table 1.** Design requirement tracking.

Number	Requirement	Implemented? (Yes\No)	Comments
1	For each access control list file, a corresponding audit log should exist	Yes	When an .acl file is created or modified an .adt file is created
2	Audit logs should have the same filename as the relevant access control list file, but with an.adt file extension	Yes	The name of the .acl file is used to create the filename of the .adt file
3	Audit logs should be appended to instead of overwritten	Yes	.adt files are appended to by default
4	Audit log files should be searchable, and the data in the log should have the capacity to be filtered and ordered	Yes	The .adt files are in json format, which can be imported into external tools

The audit module was implemented to create and update audit log files for corresponding access control files. The next section describes the tests performed on the audit module to verify if the audit module is functional and fulfils requirement four of the PAUDIT model.

### 5.3 Audit Module Testing

The implemented audit module was implemented in a testing environment. Requirement four of the PAUDIT model states that the medical aid provider should have the ability to verify any access control changes over time on the fitness resources. The following tests were performed on the audit module to see if there were specific cases which did not generate a log entry.

1. Verify existence of .adt files of default .acl files.
2. Verify existence of .adt files after default .acl files were modified.
3. Verify the existence of an .adt file after a new .acl is created.
4. Verify log entries are recorded for changes to an .acl file.

The results of the four tests are discussed in each of the following sub-sections.

#### **Audit Log Files Are Created for the Default Access Control Files**

When a new user registers on a POD server, a standard template with various folders are created for the user. The default folders are the root folder, profile, inbox, public and private. Each of these folders contain a nameless .acl file.

Each .acl file is a template that gets assigned to each new user. Since there is no put or patch activities on these access control files, not audit files are created automatically.

**Result:** Initially no audit log files are created for default access control files.

### **Audit Log Files Are Created When Default Access Control Files Are Modified**

To verify that an audit log file is generated when a default access control list is modified, the following steps were performed:

1. Select the private folder.
2. Allow Everyone viewer authorisation on the private folder.
3. Use the browser to access /private/.adt file and confirm that it contains the details of the changed access control list.

The relevant .adt file was created and contained a full record of the new .acl file as it was created. The audit file also contained the timestamp, the user and put action that caused the change.

**Result:** Changes to default access control files are audited.

### **Newly Created Access Control Files Have a Corresponding Audit Log File**

The following steps were performed to confirm if an audit log file is created when a new access control list is created.

1. Select the private folder.
2. Create a sub-folder called, fitness.
3. Change the option on the fitness folder to “Set specific sharing for this folder”.
4. Use the browser to access /private/fitness/.adt file and confirm that it contains the details of the new access control list.

The .adt file was created and it also contained the timestamp, user, handler, and detail of the access control list as it was created.

**Result:** Newly created access control lists also generates a corresponding audit log file.

### **Changes to Existing Access Control Lists Are Appended to Existing Audit Log Files**

The following steps were performed to confirm if changes to existing access control files appends an entry to the audit log file.

1. Select the /private/fitness folder.
2. Allow Everyone submitter permissions on the fitness folder.
3. Use the browser to access /private/fitness/.adt and confirm if it contains two entries. The first entry should correspond to the initial creation of the .acl file. The second entry must have the latest timestamp and should reflect that Everyone has submission authorisation.

The relevant .adt file contained two entries. The entries were the correct information and reflected the changes as it occurred in the access control list file associated with the fitness folder.

**Result:** Changes to existing access control lists are appended to audit log files.

All the tests performed on the prototype were successful, except that there were no initial audit log files created for the initial default access control lists. The creation of the initial audit log files may not be problematic, except that it may generate unusual

circumstances that may require special handling by third-parties when looking for audit log files and not finding any.

The next section evaluates the PAUDIT model and states if the audit module as implemented for the prototype solved the problem stated in Sect. 1.

## 6 Conclusion

A distributed data architecture allows users of systems, to stay in control of their personal information. A distributed data architecture introduces the concept of a personal online data store (POD) that stores personal information. A user has full control over all data stored inside their POD and can authorize other users access to the data.

Medical aid providers (MAP) may use fitness data to provide more benefits to its members. The benefits may include lower monthly premiums, but it may also include other benefits, such as cheaper gym memberships, airplane tickets etc. A MAP that uses fitness data to determine benefits needs assurance that the user cannot manipulate fitness data after it has been generated by a fitness device.

The PAUDIT model describes a model which takes data generated from a fitness device and either directly or indirectly appends the data to a fitness folder on the user's POD. The PAUDIT model describes the access control required to ensure that the user stays in control of the data, while allowing the user to append fitness data to the existing folder. PAUDIT also describes the access control required on the fitness folder to ensure that the MAP can read the data.

PAUDIT contributes to the existing Solid server implementation, by describing an audit module. The audit module creates audit log files when access control files are modified. The audit log files describe the changes to access control. The MAP can use the information in the log file to determine if a user changed permissions and potentially modified the data, thereby making the data untrustworthy. Untrustworthy data cannot be used by the MAP to calculate benefits.

The audit module was implemented in a testing environment and passed all the tests described in this paper. It should be noted that tests involving operating system commands such as curl, to manipulate access control files, were not completed. Manipulation of resources on POD servers using curl, requires WebID-TLS integration, which is currently an obstacle in the test environment. It is also the author's view that before the audit module can be accepted as a trustworthy component that tests using curl should be performed. A feature request on the node-solid-server GitHub repository is also currently being investigated to test the audit module further.

Another aspect that may require future research is the adoption of digital signatures on health data. Digital signatures on health data may negate the requirement for an audit module, since digital signatures can provide non-repudiation of health data. This will allow a MAP to verify the author of health data stored in a user's POD but may have negative consequences on the battery life of fitness tracking devices.

PAUDIT fall under the exaptation quadrant of knowledge contribution as defined by Gregor and Hevner [16]. Exaptation is defined as presenting known solutions extended to new problems. The problem domain and architecture discussed in this study is a domain that has not been observed in real-world implementations. This, however, does

not detract from the validity of the problem, as the problem may become reality and act as detractor before adoption. The PAUDIT model does not define or describe any new components that does not exist in other environments. The novelty lies in the use of the existing theory, components, and mechanisms in a new problem domain.

## References

1. Cotton, V., Patel, M.S.: Gamification use and design in popular health and fitness mobile applications. *Am. J. Health Promot.* **33**, 448–451 (2019)
2. Patel, D., et al.: Participation in fitness-related activities of an incentive-based health promotion program and hospital costs: a retrospective longitudinal study. *Am. J. Health Promot.* **25**, 341–348 (2011)
3. Fereidooni, H., Frassetto, T., Miettinen, M., Sadeghi, A., Conti, M.: Fitness trackers: fit for health but unfit for security and privacy. In: 2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE), pp. 19–24 (2017)
4. Gines, K.: Getting up to speed with GDPR. *Success. Meet.* **67**, 72–78 (2018)
5. Mansour, E., et al.: A demonstration of the solid platform for social web applications. In: Proceedings of the 25th International Conference Companion on World Wide Web, pp. 223–226. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva (2016)
6. The Solid Project: Solid (2017). <https://solid.mit.edu>. Accessed 19 Jan 2018
7. Hevner, A., March, S.T., Park, J., Ram, S.: Design science research in information systems. *MIS Q.* **28**, 75–105 (2004)
8. Pfeffers, K., et al.: The design science research process: a model for producing and presenting information systems research. In: Proceedings of the First International Conference on Design Science Research in Information Systems and Technology (DESRIST 2006), Claremont, CA, USA, pp. 83–106 (2006)
9. Mohn, E.: Semantic Web. Salem Press Encyclopedia of Science (2017)
10. W3C: Semantic Web - W3C (2015). <https://www.w3.org/standards/semanticweb/>. Accessed 18 Apr 2019
11. W3C: WebID - W3C Wiki (2018). <https://www.w3.org/wiki/WebID>. Accessed 19 Apr 2019
12. W3C: WebAccessControl - W3C Wiki (2018). <https://www.w3.org/wiki/WebAccessControl>. Accessed 19 Apr 2019
13. W3C: Cross-origin resource sharing (2014). <https://www.w3.org/TR/cors/>. Accessed 19 Apr 2019
14. Kellogg, S.: Every breath you take: data privacy and your wearable fitness device. *J. Mo. Bar* **72**, 76–82 (2016)
15. Stenberg, D.: Curl - how to user. <https://curl.haxx.se/docs/manpage.html>. Accessed 22 Apr 2019
16. Gregor, S., Hevner, A.R.: Positioning and presenting design science research for maximum impact. *MIS Q.* **37**, 337–355 (2013)