Preserving User Data Privacy through the Development of an Android Solid Library

Alexandria Lim

Follow this and additional works at: https://scholarworks.uark.edu/csceuht

Part of the Computational Engineering Commons, Data Storage Systems Commons, and the Digital Communications and Networking Commons

Citation

This Thesis is brought to you for free and open access by the Computer Science and Computer Engineering at ScholarWorks@UARK. It has been accepted for inclusion in Computer Science and Computer Engineering Undergraduate Honors Theses by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu.
Preserving User Data Privacy through the Development of an Android Solid Library
Preserving User Data Privacy through the Development of an Android Solid Library

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, AR
May, 2023

by

Alexandria Lim
Abstract

In today’s world where any and all activity on the internet produces data, user data privacy and autonomy are not prioritized. Companies called data brokers are able to gather data elements of personal information numbering in the billions. This data can be anything from purchase history, credit card history, downloaded applications, and service subscriptions. This information can be analyzed and inferences can be drawn from analysis, categorizing people into groups that range in sensitivity — from hobbies to race and income classes. Not only do these data brokers constantly overlook data privacy, this mass amount of data makes them extremely vulnerable to data breaches. To solve both of these problems of data privacy and security, one can adopt the Solid framework which prioritizes user data autonomy and allows users to choose their own applications and services. While there currently does not exist any technological support for Android, the objective of this thesis will be to start to develop an Android Solid library which will encourage the adoption of the Solid framework.
THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed ________________________________
Alexandria Lim

Refused ________________________________
Alexandria Lim
ACKNOWLEDGEMENTS

I would like to first thank the University of Arkansas Honors College for supporting this research through the Honors College Research Grant, and the Honors College fellowship for funding my studies during my time here.

I would like to thank my advisor Dr. Alexander Nelson for mentoring and guiding me through this work. I would also like to thank my thesis committee — Dr. Alexander Nelson, Dr. Pat Parkerson, and Dr. Dale Thompson. Finally, I would like to thank Zach Grider for being the graduate student mentor of the research topic during the project.
# TABLE OF CONTENTS

Abstract ................................................................. ii
Acknowledgements ....................................................... iv
Table of Contents ....................................................... v
List of Figures ............................................................ vi
List of Tables ............................................................. vii

1 Introduction .......................................................... 1

2 Background ............................................................ 3
   2.1 Pilot Studies in Protecting Personal Telemetry ................. 3
      2.1.1 Data Analysis of Personal Telemetry ....................... 3
      2.1.2 Fully Homomorphic Encryption and IoT Sensors .......... 4
      2.1.3 Semantic Web Project Solid ............................... 6
   2.2 Background Information for the Solid Protocol ................. 7
      2.2.1 Solid Protocol Terms ....................................... 7
      2.2.2 Solid Data Exchange ....................................... 8
      2.2.3 Solid Resources for Android ............................. 9

3 Implementation and Architecture .................................. 11

4 Results ..................................................................... 17

5 Discussion .............................................................. 29

6 Conclusion .............................................................. 31
   6.1 Summary .......................................................... 31
   6.2 Future Work ...................................................... 31

Bibliography .............................................................. 32
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Total Activity Count</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Sigmoid Function</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Solid WebID</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Solid OIDC Flow</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>List of Identity Providers</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>OIDC Configuration File</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>Android Manifest File</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Inrupt Login Screen</td>
<td>19</td>
</tr>
<tr>
<td>4.2</td>
<td>Inrupt Login Success</td>
<td>20</td>
</tr>
<tr>
<td>4.3</td>
<td>Update File List</td>
<td>21</td>
</tr>
<tr>
<td>4.4</td>
<td>View File Activity</td>
<td>22</td>
</tr>
<tr>
<td>4.5</td>
<td>Image View Activity</td>
<td>23</td>
</tr>
<tr>
<td>4.6</td>
<td>Delete Activity</td>
<td>24</td>
</tr>
<tr>
<td>4.7</td>
<td>Create Activity</td>
<td>25</td>
</tr>
<tr>
<td>4.8</td>
<td>Create New File</td>
<td>26</td>
</tr>
<tr>
<td>4.9</td>
<td>Updated Files after Create Activity</td>
<td>27</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 4.1: Statistics for Post Requests . . . . . . . . . . . . . . . . . . . . . . 28
Table 4.2: Statistics for Delete Requests . . . . . . . . . . . . . . . . . . . . 28
1 Introduction

“Who owns my data?” This question is complicated, and there is often no one definitive answer. Data brokers like Acxiom, Corelogic, and Datalogix would argue that their company owns your data. Data brokers are companies that collect consumers’ personal information and resell or share that information with others, and have come under scrutiny in recent years.

There are several problematic findings in a 2014 report from the Federal Trade Commission about the practices of data brokers. First, data brokers collect data numbering in the billions “largely without consumers’ knowledge”. Then, those data brokers “combine and analyze data about consumers to make inferences about them”, creating intimate digital profiles of everyone who uses the internet. These inferences may range from “dog owner” to classifications of ethnicity and income levels such as “Urban Scramble”, which “include a high concentration of Latinos and African Americans with low incomes”, or “Married Sophisticates”, who are childless, upper-class “thirty-something couples” [1]. There are two main problems that arise from these practices.

First, there is a data privacy issue. The general public is “largely unaware that data brokers are engaging in these practices” and how this data is used “may be difficult to find and understand” [1]. Reportedly, “non-financial data” is analyzed to create “pseudo credit-scores/consumer-scores” and in the worst-case scenario, “the credit score of an individual can severely affect one’s ability to secure a loan at a particular interest rate and a poor credit score could result in an individual being denied a loan or being refused a job offer” [2].

Second, there is a data security issue with the scale of the mass data aggregation in only a few locations. Data breaches are severe issues that are happening more frequently and across all industries. According to the Privacy Journal, the biggest breach of personal data of the 21st century happened on September 7,
2017, at Equifax [3]. Almost 145 million US consumer accounts were affected, and records of “social security number, date of birth, full name, and driver license numbers” and information on “credit card numbers, salary history, and loans” were compromised [4]. Anthem, the “second-largest health insurer in the U.S.”, lost 78.8 million medical records. In May 2014, eBay experienced a cyberattack and “names, addresses, dates of birth, and encrypted passwords of all of its 145 million users” were exposed. JPMorgan Chase, which is the largest bank in America, also experienced a breach, but “no money was lost and no sensitive personal information compromised.” [3]

Data brokers are able to do these practices due to the mass amount of data provided to them from companies such as Facebook, Google, or any mobile or web application that has access to user data. Data can be generated from any activity including “using a mobile device, shopping for a home or car, subscribing to a magazine, making a purchase at a store or through a catalog, browsing the Internet, responding to a survey in order to get a coupon, using social media, subscribing to online news sites, or entering a sweepstakes.” Many technology companies that users interact with “collect information about them and, in many instances, provide or sell that information to data brokers” [1].

From these practices it is clear that there are problems with the way that modern day technology handles user data. Companies should not be storing large amounts of user data due to the risk of data breaches. Companies should also not have access to as much data as they want because of the violation of user privacy. Companies should not be able to do whatever they wish with the user data such as selling it to data brokers whose practices were previously described.

Users should be able to have more autonomy over their own data. However, balancing the need to preserve user data privacy while providing data for the third parties to analyze and deliver value poses a significant challenge.
2 Background

2.1 Pilot Studies in Protecting Personal Telemetry

Three pilot studies were conducted that led to the formulation of this thesis.

2.1.1 Data Analysis of Personal Telemetry

Personal telemetry collects information that pertains to the physical actions or environment of a person themselves. Frequent collectors of personal telemetry are applications that have access to data from a mobile smartphone’s accelerometer, GPS, and heartrate sensors. There are many applications that perform these kinds of operations — Google Fit, Fitbit, and smartwatch applications. As mentioned in the introduction, the companies that own these applications are most likely selling this information to data brokers as well.

It might seem that the raw sensor data could not be as directly incriminating as a Google search history or credit score. However, analyzing that personal telemetry data is likely to expose “contexts”. For example, through analyzing the user’s location data or their accelerometer on their phones, it can be possible to determine where the user has been as well as what activities they did over a long period of time, which are potentially sensitive pieces of information.

To investigate the extent of how much information could be gained from analyzing raw personal telemetry data, there was a study in which various datasets of pregnant mothers and their daily activity were analyzed. A literature review of how to analyze accelerometer data was performed to find a consensus for what the definition of “sedentary” means. Analyzing 6 different sources led to the conclusion that “sedentary” activity means an activity count — or minute by minute observations of recordings — of less than 100 counts per minute [5], [6], [7], [8], [9], [10].
From the dataset of the physical activity of pregnant mothers, knowledge can be gleaned of what these women might be doing depending on the time of day, how active they might be, and other information that could be considered potentially sensitive. As an example of analyzed data from this study, in Figure 2.1 the percent zero over total activity count could show how inactive these women were.

From this study, it can be inferred that through analyzing even raw personal telemetry sensor data there is still leakage of potentially sensitive information, and that there needs to be a way to analyze this data while keeping user data private.

2.1.2 Fully Homomorphic Encryption and IoT Sensors

One method for preserving data privacy is to keep total obscurity between the user and the third party that wishes to analyze the user’s data by way of encryption. Traditional encryption obscures the true contents of a piece of data by converting the plaintext data into a ciphertext, which can only be readable if
the private key in symmetric encryption or the secret key in asymmetric encryption is used. This fulfills the need to keep the contents of data private between trusted parties but fails to provide untrusted parties a way to access data’s contents without violating user privacy.

However, there is a special type of encryption called fully homomorphic encryption that allows for encrypted data to be processed and produces an encrypted answer that, when decrypted, matches the answer as if the unencrypted data were processed directly [11]. So theoretically, a user could encrypt all of their own data with fully homomorphic encryption and send their data to be processed by a third party, who would process the received ciphertexts and return the post-computed ciphertexts, which the user could decrypt.

Thus, the second pilot study conducted was to attempt to create a proof-of-concept IoT system that encrypts all data values using fully homomorphic encryption before processing, so as to preserve user privacy. The IoT system used the PySEAL library [12], which is an open-source library that has functions to encode and encrypt plaintexts, perform arithmetic operations such as “multiply” and “add” to ciphertexts, and finally decrypt and decode ciphertexts to get the plaintext answers. Within this IoT system, there were multiple sensors that collected personal telemetry such as a light and presence sensor, as well as a Raspberry Pi that served as the aggregation point. The sensors transmitted their data through MQTT to the Raspberry Pi, which encrypted the data values from the sensors, and send the ciphertexts to the “untrusted cloud”, which in this system was a computer. The computer performed computations on the ciphertexts and sent back the results to the Raspberry Pi. The Raspberry Pi could decrypt the ciphertexts and depending on the results of a sigmoid function which places analog values as a discrete value ranging from 0 to 1, the sensors would perform an action. When attempting to perform a sigmoid function as seen in Figure 2.2 using the sensor’s data values, which involve cubing and multiplying to the 5th power, the ciphertext became too corrupted with “noise” to decrypt back into any usable result.

Thus, devices such as smartwatches or smartphones probably would not
be able to handle the resource heavy computations necessary for performing fully
homomorphic encryption.

Since fully homomorphic encryption’s inception by Craig Gentry in 2009,
there has been work done to lessen the computational burden. There is a method
called bootstrapping, which resets the amount of noise when computing on cipher-
texts [11]. However, as of now, fully homomorphic encryption is not a good fit for
protecting personal telemetry of users.

2.1.3 Semantic Web Project Solid

Lastly, the use of the Solid framework was considered. The Solid framework
is an idea from Tim Berners-Lee, the father of the internet. Berners-Lee was
concerned with the mass aggregation of data on the web between only a couple of
tech giants and noted that users had to give away their data for “perceived value” of
the applications. To further develop his vision for an Internet that would prioritize
decentralized information and user data privacy and sovereignty, Sr. Berners-
Lee founded the open-source project Solid, and later founded a startup company
called Inrupt. Inrupt’s current pilot projects include partnerships with “Britain’s
National Health Service and with the government of Flanders, the Dutch-speaking
region of Belgium” [13].

Within the Solid framework, all users have a personal pod. One can store
any kind of data in a pod and all data is interoperable, meaning that different
applications can all work with the same data. The user can grant specific access
to each piece of data in their pod. Access may also be revoked at any time. This
gives users control over their own data autonomy. Solid prioritizes the values of “an
equitable, informed and interconnected society” and the development of a space
where users “maintain their autonomy, control their data and privacy, and choose applications and services to fulfil their needs” [14].

2.2 Background Information for the Solid Protocol

This section will detail basic information about the Solid specification. This information comes from the Solid Specification document [14].

2.2.1 Solid Protocol Terms

Each data resource in the Solid pod can be represented as an RDF document with a specific URI (Unique Resource Identifier). All of the data in the pods are represented in linked data format. Each person, organization, or software is represented with a WebID. Each WebID is unique and is the primary identifier within the SOLID ecosystem. A WebID is an HTTP URI that resolves to an RDF document representation of the WebID profile which shows the location of personal storage links and the authentication endpoints, as shown in Figure 2.3 [14].

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix solid: <http://www.w3.org/ns/solid/terms#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<https://id.inrupt.com/partyboybcb> a foaf:Agent;
    <http://www.w3.org/ns/pim/space#storage>
    <https://storage.inrupt.com/96cf53c1-c48d-414e-84d5-b0001339c2d2>;
    solid:oidcIssuer <https://login.inrupt.com>;
```

Figure 2.3: Solid WebID
2.2.2 Solid Data Exchange

Solid uses OIDC as its form of authentication and there exist OpenID Providers specifically for Solid that can issue access tokens necessary to inter-
act with the resources within the Solid pods. These providers have their own configurations, documented under “./well-known/openid-configuration”. In these configurations are several endpoints necessary for performing authentication. The detailed flow of the authentication is shown in Figure 2.4 taken from the Solid OIDC documentation [15].

Servers and clients of the SOLID specification must use HTTP 1.1 and TLS connections. Once authenticated, the server and clients use HTTP requests to communicate.

To create a resource, one must make a POST request to the storage location of the wanted file. Optionally, one can set a title for the file by setting a SLUG header. In addition, one can define the content type of the message, which may be a text note or a picture.

To read a resource, one must make a GET request to the storage location of the wanted file. One can set the “Accept” type header to further specify what kind of resource is expected to be returned.

To delete a resource, one must make a DELETE request to the storage location of the wanted file along with the specific file name that will be deleted. This must be done carefully to make sure that only standalone resources are deleted and not directories that contain multiple resources.

To edit a resource, one edits the access control list of a resource. To do this, one must make a GET request to the storage location of the specific file and read the parameter of the link header “acl” for access control list. To do the actual editing of the access control list, one must make a PATCH request and change the RDF values in the list which is available under the location within the “acl” header [14].

2.2.3 Solid Resources for Android

There has been a lot of development with Solid applications in the JavaScript language. On the Solid Project website, there is an extensive list of JavaScript li-
braries for tasks such as authentication, login, and session management, as well as JavaScript libraries for querying and manipulating RDF like resources on Solid servers. There are also web applications for purposes that range from note taking, movie tracking and recommendation sharing, chatting, and managing files on a Solid pod. Surprising, however, is the lack of tools or libraries specifically for Android mobile development, and there are also not many Solid compatible mobile applications [16]. This is concerning, since as previously discussed, mobile applications have access to not only all the data that a desktop application, but it can also track a person’s personal telemetry data as well. In addition, 70 percent of the global market share of smartphones use Android [17].

However, unlike JavaScript developers, Android developers have currently no help in development, so there is a need for a library specifically for Android developers. Thus, this project’s end goal is an Android library that can reduce time and effort when integrating into the Solid framework, which prioritizes user data autonomy.
3 Implementation and Architecture

To reduce the amount of effort needed to develop applications using the Solid Protocol for mobile applications, there should exist an Android Solid library that could perform the basic CRUD operations (create, read, update, and delete) when interfacing with the resources within a Solid pod. There will be five steps to creating this library - creating the Solid pod, configuring authentication with the Solid pod, implementing the CRUD operations, demonstrating the library’s capabilities, and refactoring the necessary code into a single library.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Responsible for Domain Name and Terms</th>
<th>Responsible for Hosting</th>
<th>Hosting Location</th>
<th>Solid Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inrupt Pod Spaces</td>
<td>Inrupt, Inc.</td>
<td>Amazon</td>
<td>Germany</td>
<td>ESS</td>
</tr>
<tr>
<td>inrupt.net</td>
<td>Inrupt, Inc.</td>
<td>Amazon</td>
<td>USA</td>
<td>NSS</td>
</tr>
<tr>
<td>solidcommunity.net</td>
<td>Solid Project</td>
<td>Digital Ocean</td>
<td>UK</td>
<td>NSS</td>
</tr>
<tr>
<td>solidweb.org</td>
<td>Solid Grassroots</td>
<td>Hosteurope</td>
<td>Germany</td>
<td>NSS</td>
</tr>
<tr>
<td>trinpod.us</td>
<td>Graphmetrix, Inc.</td>
<td>Amazon</td>
<td>USA</td>
<td>TrinPod</td>
</tr>
<tr>
<td>use.id</td>
<td>Digita</td>
<td>DigitalOcean</td>
<td>EU</td>
<td>CSS</td>
</tr>
<tr>
<td>solidweb.me</td>
<td>Meisdata</td>
<td>Hosteurope</td>
<td>EU</td>
<td>CSS</td>
</tr>
<tr>
<td>Data Pod</td>
<td>iGrant.io, Sweden</td>
<td>RedPill Linpro, AWS, GCP</td>
<td>EU</td>
<td>NSS</td>
</tr>
</tbody>
</table>

Figure 3.1: List of Identity Providers
The first task was to create the Solid pod to interface with. On the Solid website instructions to create a pod [18], there is a list of choices varying on location as seen in Figure 3.1. The identity provider Inrupt Pod Spaces, which uses the new Inrupt’s Enterprise Solid Server, was selected. There was also an option to self-host a pod, but this route was not chosen since the majority of users would not use a self-hosted pod.

The second task was to configure authentication with a Solid OpenID provider. To even begin to interface with Solid Pods, authentication must be completed with the Solid Identity Provider. After authentication, the applications will receive access tokens with which the application can access the resources within the Solid pods. To perform this key OIDC authentication on a mobile application, a mobile library is needed that would be general enough for accommodating all kinds of Identity Providers and not just a single one, such as OneLogin or Okta applications.

After some research on what options there were available to perform OIDC authentication for Android, the most popular and developed library called AppAuth, an “Android client SDK for communicating with OAuth 2.0 and OIDC providers” was chosen [19]. Thus, this was the library that was chosen that needed to be configured to use the Solid Identity Provider Inrupt.

To set Inrupt as the Identity Provider, the mobile application code must be changed. The configuration file as shown in Figure 3.2 must contain the discovery URI. First, the redirect URI can be left the default value, which was the AppAuth Application. The client ID field was not necessary, since this application would not be registered with the Inrupt Identity Provider. The discovery URI is the link to the Identity Provider’s OpenID configuration, which is a JSON file with various endpoints such as the authorization endpoint, registration endpoint, and the scopes that were available for the specific identity provider. Every Identity Provider such as Okta or OneLogin has a configuration file. Then, the redirect scheme must also be defined to be the Solid mobile application as shown in Figure 3.3.

The third task was to implement the CRUD functions within Android Solid
Figure 3.2: OIDC Configuration File

```json
{
  "client_id": "",
  "redirect_uri": "net.openid.appauthdemo:/oauth2redirect",
  "end_session_redirect_uri": "net.openid.appauthdemo:/oauth2redirect",
  "authorization_scope": "openid webid offline_access",
  "authorization_endpoint_uri": "",
  "token_endpoint_uri": "",
  "registration_endpoint_uri": "",
  "user_info_endpoint_uri": "",
  "https_required": true
}
```

Figure 3.3: Android Manifest File

```xml
<activity
    android:name="net.openid.appauth.RedirectUriReceiverActivity"
    android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="https"
          android:host="appauth.demo-app.io"
          android:path="/oauth2redirect"/>
  </intent-filter>
</activity>
```
Library. The Android Solid Library should enable users to create, read, and delete files in the Solid Pod, as well as edit the users who can access files.

To access the resources within a Solid Pod, specific HTTP requests must be crafted by using the library HttpURLConnection. For all of these HTTP requests, the Authorization header was required with the Bearer access token.

To create a resource, the HttpURLConnection method is “POST” and targets the future storage location of the file. The SLUG header can be defined with a specific value which would be the file name in the Solid pod, as well as the content type header, which may be a text note or a picture. In Solid Photos, it should be a JPG image. The user is able to take a picture with their camera and upload it to their Solid Pod with the push of a button, which listens for clicks and makes these HTTP requests. Upon a successful request, the response code should be 201, which means that “the request succeeded, and a new resource was created as a result” [20].

To read a resource, the HttpURLConnection method is “GET” and targets the storage location of the file. In SolidPhotos, the images are displayed when the application decodes the stream from the response of the request into a bitmap, which is decoded again into an image to display on the phone. The request is successful if the HTTP code 200 is returned.

To delete a resource, the HttpURLConnection method is “DELETE” and targets the storage location of the file. The response code from the request should be 204, which means that the request was successful but “there is no content to send for this request” [20].

The edit function for access to the file is currently not implemented. Part of the reason is due to the fact that there is no native way to parse RDF in Android at this moment. Different ways to natively parse RDF on Android were explored, but none worked. The first option was Rio, an Eclipse library that could perform RDF parsing and queries. This library just did not work because its DatatypeFactory could not be properly instantiated. The second option was the Apache Jena library. However, this library was too large and overwhelmed the
limited memory for mobile applications. Lastly, the jena-android library which was written 8 years ago and seems to be not updated anymore was tried. But there were still issues with the library being too large to fit on limited memory. In the end, it was concluded that there was no current way to natively parse RDF in Android and other solutions needed to be explored.

The solution was then to send the received data from the WebID GET request to a separate web server on the UARK network which could instead process the RDF and send back storage links and a list of all the files that were in a pod. Of course, using a webserver is not completely ideal and the mobile application should be able to parse all the RDF natively to ensure that there is no data leakage between the mobile application and the webserver.

The fourth task was to create a Solid compliant photo gallery mobile application. The Solid compliant photo gallery, or Solid Photos, should be able to let a user view their photos in their Solid Pod (read operation), delete photos (delete operation), upload photos from storage or access the camera to get a photo (create operation), and edit who can have access to the photo (edit operation). In addition, there were some functions that needed to be implemented for the Android application itself, such as a function to get the list of current files within the Solid pod, and a method to get the storage URL of a user based on their unique WebID.

The last step was to separate the code for the CRUD operations into their own library. This step is also still a work in progress since not all of the CRUD operations are implemented.

In conclusion, there are six different functions in the Solid Photos application that could later be moved to a separate Android Solid library.

- Create resource – uploads a photo to a Solid pod through Solid Photos
- Read resource – views a photo on a Solid pod through Solid Photos
- Update resource – edits who can access photos in a Solid pod through Solid Photos
- Delete resource – removes a photo from a Solid pod through Solid Photos
- Get storage URL – returns the storage URL of a pod user
- Get file list - returns to the list of files of a pod user
4 Results

For the results, the capabilities of the current Solid Photos application, a Solid compatible photo gallery application, are demonstrated. Then, the amount of time needed for the POST and the DELETE requests will be tested and a statistical analysis will be performed.

The application can perform the operations of viewing and deleting photos in their Solid pod. However, there are issues with uploading a photo to the Solid pod, so this is demonstrated through uploading text to their Solid pod. The edit function was never completed due to the roadblock of there not being a way to parse RDF and write SPARQL queries in Android.

First, through using the AppAuth login feature configured with the Identity Provider Inrupt, logging in is possible. Figure 4.1 shows the login screen and Figure 4.2 shows a successful login attempt.

Once logged in, users can view the starting screen. The “Update List” button displays list of all of the files that are currently in the Solid Pod as shown in Figure 4.3.

Each of the “View” buttons are used to view each resource as shown in Figure 4.4. This application was meant to be a photo gallery application for viewing pictures, but of course there are a multitude of files that can be stored in the Solid Pod.

To demonstrate the viewing function, the “bunny.jfif” file’s View Button was selected. The response of the request should be a bitmap that is shown in the ImageView, as shown in Figure 4.5.

Each of the delete buttons will delete the file on the same line. Here is the file with the name “e5da94f5-c702-4e76-8c66-36d7b14e160c” as shown in Figure 4.6.

Now the upload activity is demonstrated. It would have been ideal to have the functionality to upload a photo taken by the camera, but there were issues
with uploading data that could be accepted by the Solid pod. The uploaded image was corrupted and could not be uploaded to the pod. Thus, instead of uploading images, the app uploads text files to demonstrate the create function as shown in Figure 4.7. After the file is changed in Figure 4.8, the new updated file list is shown in Figure 4.9.

Next, several metrics were taken of the post and delete requests.

To test the post requests, there were text files of varying sizes that were uploaded to the Solid pod. Generally, as the file sizes were increased, the average times to complete the requests also increased, and the varying differences between the times also increased. This makes sense as the file sizes increased and thus there is more time needed to send the file to the Solid pod.

The delete requests were tested right after the post requests by deleting all the files that were created in the post requests. The numbers stay about the same because nothing is required to be sent back to the mobile application except the code 204 which shows that the delete request was successful.

The first request’s time was always removed, because the first request takes a long time due to all the handshaking that goes on. The later requests are shorter and more consistent in time. After each request, the HTTP connection was disconnected, and the thread making the requests was put to sleep to ensure that the connections were not being reused and skewing the resulting times. These statistics were taken from a dataset of 100 trial runs of POST requests and DELETE requests.
Figure 4.1: Inrupt Login Screen
Figure 4.2: Inrupt Login Success
Figure 4.3: Update File List
Figure 4.4: View File Activity
Figure 4.5: Image View Activity
Are you sure you want to delete the resource?

/96cf53c1-c48d-414e-84d5-b0001339c2d2/e5da94f5-c702-4e76-8c66-36d7b14e160c

Figure 4.6: Delete Activity
Figure 4.7: Create Activity
Figure 4.8: Create New File
Figure 4.9: Updated Files after Create Activity
### Table 4.1: Statistics for Post Requests

<table>
<thead>
<tr>
<th>File Sizes</th>
<th>Avg Times</th>
<th>Standard Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB</td>
<td>293.84 ms</td>
<td>36.72</td>
</tr>
<tr>
<td>8KB</td>
<td>306.9 ms</td>
<td>35.13</td>
</tr>
<tr>
<td>16KB</td>
<td>419.91 ms</td>
<td>71.72</td>
</tr>
<tr>
<td>32KB</td>
<td>575.62 ms</td>
<td>41.27</td>
</tr>
<tr>
<td>64KB</td>
<td>791.64 ms</td>
<td>92.51</td>
</tr>
<tr>
<td>128KB</td>
<td>1173.49 ms</td>
<td>128.78</td>
</tr>
<tr>
<td>256KB</td>
<td>1747.91 ms</td>
<td>151.21</td>
</tr>
<tr>
<td>512KB</td>
<td>2503.85 ms</td>
<td>183.15</td>
</tr>
<tr>
<td>1MB</td>
<td>3417 ms</td>
<td>258.53</td>
</tr>
</tbody>
</table>

### Table 4.2: Statistics for Delete Requests

<table>
<thead>
<tr>
<th>File Sizes</th>
<th>Avg Times</th>
<th>Standard Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB</td>
<td>218.18 ms</td>
<td>45.66</td>
</tr>
<tr>
<td>8KB</td>
<td>204.49 ms</td>
<td>15.84</td>
</tr>
<tr>
<td>16KB</td>
<td>217.21 ms</td>
<td>34.93</td>
</tr>
<tr>
<td>32KB</td>
<td>214.99 ms</td>
<td>19.06</td>
</tr>
<tr>
<td>64KB</td>
<td>221.22 ms</td>
<td>25.53</td>
</tr>
<tr>
<td>128KB</td>
<td>212.55 ms</td>
<td>18.36</td>
</tr>
<tr>
<td>256KB</td>
<td>216.84 ms</td>
<td>24.21</td>
</tr>
<tr>
<td>512KB</td>
<td>212.51 ms</td>
<td>37.06</td>
</tr>
<tr>
<td>1MB</td>
<td>222.18 ms</td>
<td>30.92</td>
</tr>
</tbody>
</table>
5 Discussion

While encouraging the adoption of the Solid ideals for the modern web is a step in the right direction of placing ownership of data back into the user’s hands, there are still issues that remain to be resolved.

Firstly, when the user gives a third-party access to their data, nothing is preventing the companies from making a copy of the data. If the user refuses to give away their data, then the company cannot process the data. This is a legitimate problem, but the principles of Solid will at least prevent all of the user’s data from being stored immediately with the third-party company. Fully homomorphic encryption could still be a solution to this problem to ensure total obscurity of the data between the user and the third-party company.

Secondly, while users will be more aware of the data they give away to the companies, there will be a massive mental strain on the user if they have to give consent to every bit of information that applications request for. Further experimentation is needed to ensure that the users of the data being requested are fully aware of what data is needed but not overwhelmed.

Thirdly, it is true that Solid is not a mainstream framework yet. There will have to be an influential technology company like Google or Amazon that will have to make the conscious decision to transition towards the Solid framework which prioritizes user data autonomy. Then other companies will have to either also respond with their own adoption of more user data autonomy friendly policies or present their own solutions. Or there will be a law like the General Data Protection Regulation (GDPR) that will force companies to comply.

However, if these questions can be solved, honoring user data autonomy would be much easier for mobile applications, which are still privy to a large amount of sensitive personal data.

Recently there was a Java Solid client library written by the Inrupt co-
munity [21]. This shows that there is an interest in encouraging development with the Solid framework in multiple languages, and especially in a language that is of use to Android. However, there still exists no Android-specific libraries.
6 Conclusion

6.1 Summary

User data privacy and autonomy are not prioritized enough in today’s technological systems. Data brokers are able to gather mass amounts of personal information to make digital profiles which can be used to make decisions that affect people in their day-to-day lives. One can adopt the Solid framework within all types of applications to ensure that the users have autonomy over their own data. While there currently does not exist any technological support for Android, the objective of this thesis was to start to develop an Android Solid library which will encourage the adoption of Solid and its ideals of user data autonomy.

6.2 Future Work

Further software development is needed to natively parse RDF on Android. The lack of resources for parsing RDF on Android was a big obstacle to the total completion of the Android Solid library, which is surprising considering that RDF is the industry standard for representing data on the internet. Realistically, the mobile application should not have to ask a separate server to parse the RDF of HTTP responses like the current solution of the thesis work is now.

The editing access for users to different resources in the Solid Pod, which is the most important function of the Solid pod, was not able to be implemented in time. This would be one of the highest priority tasks for future work on this mobile application and Android Library.

Additionally, the Java Android library could be converted to Kotlin Multi-mobile Platform due to development being possible in both Android and iOS.
Bibliography


[12] Lab41, “Lab41/pyseal: This repository is a fork of microsoft research’s homomorphic encryption implementation, the simple encrypted arithmetic library (seal). this code wraps the seal build in a docker container and provides python api’s to the encryption library.” [Online]. Available: https://github.com/Lab41/PySEAL


