Permissioned IoT Cloud

Sarah Colpitts and River Gillis

***Overview*—We aimed to create a framework for storing Internet-of-Things data in the cloud that will give the user full control over who has access to his or her data. This has been accomplished with the use of a server exposing a REST API that anyone can connect to, or host their own version locally. We have set up a system by which users can invite friends and family to read/write from their personal cloud space. We have demonstrated this framework's potential using a Raspberry Pi hooked up to a Si702 se1nsor that reads temperature and humidity. The Pi continuously sends the sensor readings to the server we have set up and is accessible only to those with access to the cloud. Viewing the data and controlling permissions is accomplished by a mobile app.**

I. Introduction

THE internet was initially built to support shared data across interconnected networks. It was developed around an idea of net neutrality wherein access to data was made available to all individuals regardless of class or regional privileges. While the internet has become the greatest catalyst for our growing global economy, our perpetual increase in data sharing means an increase in storage of personalized data within centralized areas, most held under the control of large corporate companies. According to a report published by digital security specialists Gemalto, 4.5 billion data records were compromised worldwide in the first half of 2018 alone. On average, only 4 percent of data breached was encrypted [1]. Not only that, the mean-time-to-identify a data breach for companies was 197 days. The mean-time-to-contain a breach was 69 days [2]. These numbers indicate not only a lack of security protocols under which most data is kept, but also a lack of efficiency in responding to and containing data breaches when they occur.

Even when data has not been explicitly hacked, corporations give themselves a vast amount of leniency when it comes to allowing third party access to user data. Facebook, for example, allowed data from 50 million users to be accessed by the political data firm Cambridge Analytica in 2016. Cambridge utilized this collected data by creating comprehensive psychological profiles on users that were used for targeted political ads and biased news articles. Facebook maintains this manipulation of user data was not a security breach, but rather a violation of terms of service [3]. Either way, Facebook failed to protect its users from having their private data exposed and manipulated. Cambridge Analyticas former CEO Alexander Nix claimed none of the data gathered on users was particularly intrusive or sensitive and that he had, in fact, saved users time by presenting them with information on issues fundamentally important to them [4]. This goes to show, there is a great deal of legal ambiguity when it comes to user privacy. Most countries, the U.S. included, have failed to pass laws properly dictating boundaries and regulations regarding individuals personal data.

As the market surrounding smart home devices continues to grow, more and more data is being collected regarding the day to day habits and movements we exhibit inside our homes. These products are meant to simplify user's lives by automating what were once manual tasks. Through ubiquitous data collection and machine learning, thermostats can now anticipate user tendencies and preempt desired temperature settings without any conscious decision making from the user. Smoke detectors can automatically call 911 when smoke is detected in a room [5]. Security systems can be monitored remotely on the user's mobile devices and respond automatically when a break-in occurs [6]. Invisible computing, such as this, helps to make users lives easier and, in some instances, safer. But just as Facebook failed to protect users social media data by sharing it with undisclosed third parties, smart home device manufacturers such as Nest and Honeywell have already been selling data to energy providers in order to profile and predict users energy usage [7]. As of yet, such data sharing between smart thermostats and utility companies has primarily been to reduce strains on electrical grids during periods of heavy usage in order to reduce power failures. Honeywell vice president Jeremy Eaton professes that everything is completely transparent for the end user and that utility [companies] are incredibly sensitive to data and privacy issues [8]. It is however, conceivable to imagine an instance in which such external control over home devices might pose as a vulnerability for users in the future, be it from utility companies themselves or malicious hackers.

Tim Berners-Lee, initial creator of the world wide web, has become a vocal critic of the level of data harvesting performed by Big-Tech data centers today. He has called for a new decentralized structuring of the internet wherein users have singular access to and control over their own data.

Our solution to this problem is a personalized cloud storage framework for storing IoT device data that will allow users to have complete control over the sharing of their personal data through a permission system. Permissions are set by the user to allow data access to select individuals. Users can add specific restrictions regarding when, at what threshold, and what degree of precision permission-ed individuals can access their data.

Ideally, our solution will not rely on any single cloud provider to accomplish this goal but will give the user the ability to store his or her data wherever and access it however he or she wishes. Our solution will involve a demo featuring a sensor connected to a Raspberry Pi that will be sending data to the cloud server. We will create a mobile app that will allow the user to view the data and control who else has permission to view that data.

II. Related Work

In 2018 Tim Berners-Lee launched a new company called Inrupt to support his goal of creating a decentralized web based on his previously conceived Solid project [9]. Solid utilizes a socially linked data network wherein users have complete ownership of their personal data and are able to control outside access to data through a permissions based system [10]. Due to the novelty of the Inrupt platform, many functional aspects are still in need of refining and not all-together convenient for users. Thus, we are attempting to build off Berners-Lee's Solid concept, maintaining the primary emphasis of separating applications and the data they utilize. We hope to incorporate existing devices and thus enable greater home automation while providing an environment in which data is kept in a secure location, owned and controlled solely by the user.

III. Application Design

There are multiple components that comprise this project, the main two being the back end code (the aforementioned framework) and the mobile app, which gives the user a way to view the data and manipulate permissions. We utilized a Model-View-Controller pattern for our development. The Raspberry Pi acts as a client (the view), sending data to the server (the controller) which stores the data in a database (the model). The mobile app also acts as a view interacting with the controller.

A. Mobile App (built by River)

The mobile app is built using React Native, which allows for it to run on iOS and Android. This makes it much easier to perform the network operations required to view the data, as API requests to the back end are done using the built-in `fetch()` function in JavaScript. React Native works well since the app is simple and used solely to demonstrate the usefulness of the framework.

Currently, the app features a login page on startup that will accept an email and password and make an API request to the server attempting to log that user in. On success, the server responds with an authentication token that is stored in the app's state using Redux [11]. This allows for the app to keep track of whether the user is logged in or not, simply by checking the validity of the token.

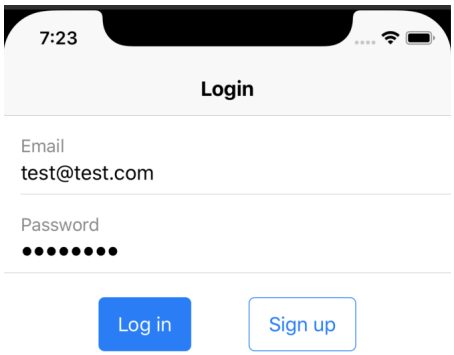

Fig. 1. Login page for the demo app

Once logged in, the user is able to see a list of devices that they either own or have read access to, along with the most recent data that has been sent to the device. The app refreshes the data every second, and caches the 10 most recent payload properties for each device, where each payload property is composed of two key-value pairs (value and units). The cached value list for each property is presented via a graph SVG, generated using the `react-native-responsive-linechart` library [12]. When the data is updated, the charts are all re-rendered to give the most up-to-date view.

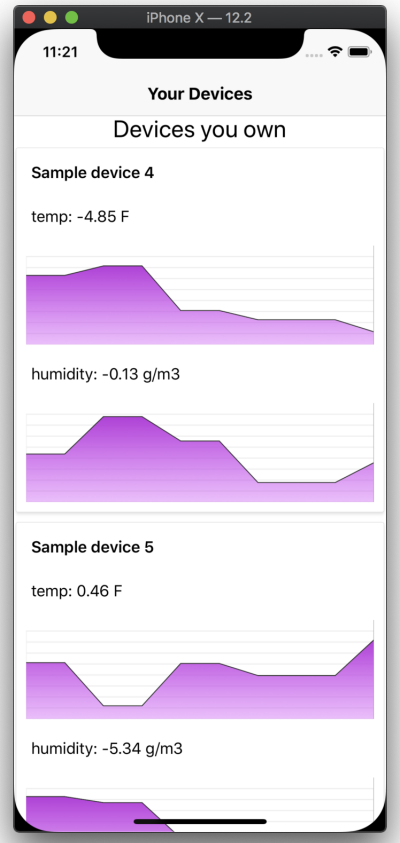

Fig. 2. User view of their list of devices with payload data

For devices that the user owns, the user is able to edit what other users have access to read the data, via email address. For each person that the owner gives read access to, the owner can chose to implement granular permissions that determine how that user is able read the data. This will give users the ability to control the extent to which other users are able to read their data. It will also work for services. For example, a user could add an email like `iot@google.com` to allow Google services to read the device data, while that data is still being restricted via granular permissions.

Users who change to the 'Readable' tab will be able view all of the devices that they have read access to (updating in real-time on the app). Readers, however, may have their data

restricted. The graphs are shown with as much or as little data as the owner grants for that particular reader, and the user reading data from the device is unable to see how the data has been manipulated for them. Given that they do not own the device, users with read-only access are unable to view further device properties (like the readers list or restrictions list) and are unable to manage the device.

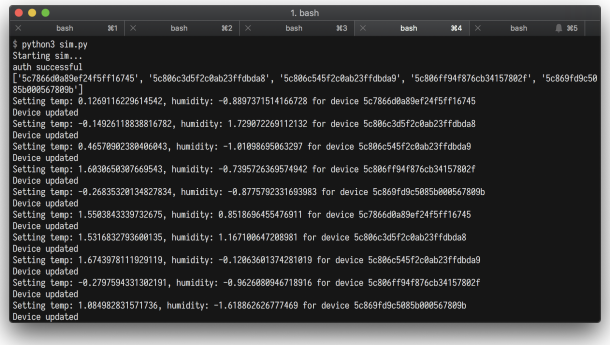

Fig. 3. Terminal capture of the device simulator

B. The Back End (built by Sarah and River)

The back end is the core of the project and required the most thought put into its design. Given the current state of Solid development, we decided that it would be more beneficial to create the framework for a permissioned IoT cloud using a traditional web technology stack in a way that does not tightly couple the system to the provider, so that when Solid development matures the project can easily be migrated to a Solid pod.

The back end is built using Node.js to run server-side JavaScript, and Express.js as a web framework to help build a REST API on top of Node [13], [14]. Currently, the API allows for the creation and authentication of users using JSON Web Tokens (JWT) which allow us to authenticate users by encoding their data in the token along with a secret key [15]. The tokens are then used for any REST endpoints that require authentication by placing them in the HTTP headers using Bearer authentication [16]. User account passwords are hashed and salted using bcrypt, which is specialized for hashing passwords [17].

All data is stored in a MongoDB database, which was chosen due to its flexibility for development [18]. The database is interacted with using Mongoose to model objects, which allows the project to easily transfer to other database systems using an Object-Relational Mapping library [19].

The REST API currently exposes methods to create and interact with devices, which are objects describing a sensor or other device that produces data and is owned by a user. Each device has a list of users who have read access (other than the device owner). Each of those readers will be able to view that device's data when they issue a GET request to /devices. The server includes a granular permissions layer wherein sensor data can be manipulated in order to hide a certain level of precision before being served as a response to the reader. So far, three forms of granular permissions can be applied to shared data. Users can chose to apply a round-to-nearest value to reduce the precision of the data shared. A fuzz value can also be applied to hide data precision. The user can also chose to block data from the reader entirely.

The granular permissions layer is implemented when the device's owner sets restrictions on an individual reader. As of right now, users are able to execute three forms of reader restrictions on shared data. Threshold based restrictions, when set by the user, specify a minimum to maximum threshold level of the incoming data that should cause the data to undergo granular deprecation before being sent to the reader. A time based restriction specifies the time of day when restrictions should be implemented on the data. The last form of restriction enables the user to specify what days of the week he or she would like data to be restricted.

Devices currently store the last payload sent to the server for that device, along with a timestamp and expected time of next payload (given as a timeout period). This allows for the client to cache any data they receive as they make requests for the device payloads in order to create a history of data. The timestamp and timeout period allow for the client to determine when the device has come online and gone offline.

C. Hardware (built by Sarah)

Our hardware setup includes a Raspberry Pi 3 model B running Raspberry Pi's Linux based operating system, Raspian, as well as an Adafruit Si7021 temperature/humidity sensor connected via I2C communication [20].

We utilized a number of Adafruit API libraries in connecting to the sensor device. These include the Adafruit CircuitPython library bundle, the Adafruit CircuitPython BusDevice library, and the CircuitPython driver for the Si7021 temperature and humidity sensor [21]. CircuitPython is a product of Adafruit, used to assist in controlling hardware devices in a Python environment [22]. The Bus Device library helps to handle I2C and SPI serial data transfers in Python as well [23].

The Pi runs a python program that takes sensor information from all devices connected to the account and sends that data to the server in individual payloads. Currently, the sensor setup is a relatively manual operation. The user email and password must be hard coded into the python program. It then fetches an array of all device ids owned by that user through a GET request to the server. Once all device ids have been fetched, the program cycles through each device, reading the sensor value and then sending a payload with these values out to the server. As the hardware is currently set up, all timeout values for a user's devices must be the same as they are read in a consecutive loop. Also, we have not yet established a way by which to determine a device's specific sensor type, thus all devices are assumed to be the same Si7021 temperature/humidity sensor.

Below are outputs of the server's response as the python program posts sensor data remotely from the Raspberry Pi.

Fig. 4. temperature/humidity sensor on POST to server

Initially, the program logs in to its user account with a password and email address. The server then responds with an authentication key. This key is added to the header in future server calls to ensure authentication. Next the program fetches the list of devices attributed to its account (in the example above there is only one device). Hereafter, the program perpetuates a loop that takes data from the sensor and sends it as a payload to the server in a post request.

This hardware setup is rudimentary and meant merely as a stand-in prototype for any commercial IoT device. Primarily, we have demonstrated the protocol by which a device would send and receive data to and from the server.

IV. Results

We have created a system that enables users to have much more control over their internet-of-things data. The platform we have created relies on a simple API that takes into account user authorization context in order to display relevant data in a secure way. We have supplemented the platform with a mobile app that allows user-friendly access and control over the API.

The platform, mobile client, and hardware client work together to allow users to store their IoT data flexibly, to restrict who has access to that data with as much granularity as desired, and to view data in a simple and useful way.

The platform is modular, and allows users to easily swap components however they desire. If someone wants to spin up an instance of the framework with a different database, they need only connect to the new database and find a suitable object-relation mapping or object-document mapping library to replace mongoose.

This flexibility in the core platform and API allow for the code to work well in a variety of situations, such as a local instance specifically for a house or office building, or as a managed instance in the cloud, allowing users to sign up and connect to the platform instantly.

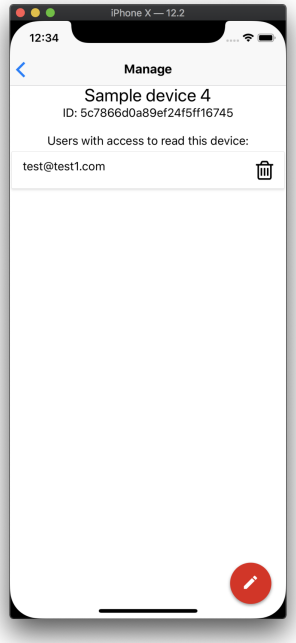

Fig. 5. Owner managing list of readers for 'Sample device 4'

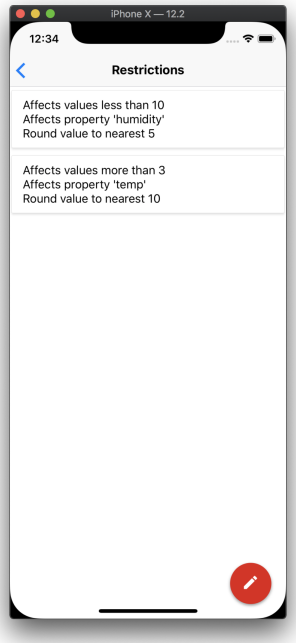

Fig. 6. Owner managing list of restrictions for 'Sample device 4' for 'test@test1.com'

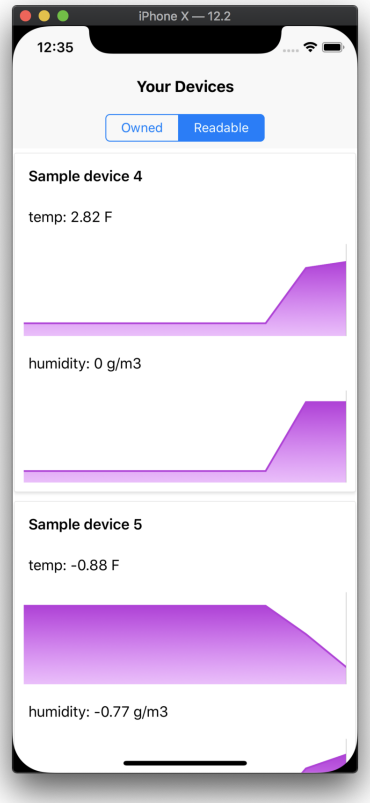

Fig. 7. Reader viewing restricted data for 'Sample device 4' (actual data: temp=2.82 and humidity=2.01)

The presented system enables further development of ubiquitous computing, as it gives the developers an easy-to-use platform to develop their IoT projects on. Additionally, the API and the inner workings of the platform are meant to be invisible. Users who have developed IoT systems in the past shouldn't have to learn anything new to develop with this system. Users who simply wish to read data or receive alerts can do so easily from the mobile client without having to think about how the data got there or how it was manipulated.

V. Future Work

While we have created a useful and functioning system, there are still many areas to improve upon. In the future, extending functionality already available in the API to the mobile app would be a great place to start. Currently, users are only able to create restrictions via the API directly. Adding a form to the mobile app that would allow users to create restrictions more easily. Additionally, the app should use the timestamp of the last received payload, along with the timeout value for the device, to determine if the device is online. This online-status should be shown to the user as a badge on each device card.

In addition to the mobile app, we would also like to provide an easy-to-use terminal client. Currently, using the API from the terminal requires manually creating the HTTP requests with `curl`. This isn't very user-friendly, so replacing it with an actual client would be very useful.

We would also like to add support for WebSockets. Currently, the API is designed around constant updates for querying the data. This means the user has to manually set up and keep track of a timer to make API requests. This is a cumbersome and inefficient process. Providing a WebSocket API in addition to the HTTP API would allow developers to send and receive data in real-time, opening up possibilities for new types of data, like streaming video or audio.

References

[1] https://breachlevelindex.com/
[2] Columbus, L. (2018, July 27). "IBM's 2018 Data Breach Study Shows Why We're In A Zero Trust World Now". Retrieved from https://www.forbes.com/sites/louiscolumbus/2018/07/27/ibms-2018-data-breach-study-shows-why-were-in-a-zero-trust-world-now/#18af22168ede
[3] Granville, K. (2018, March 19). "Facebook and Cambridge Analytica: What You Need to Know as Fallout Widens". Retrieved from https://www.nytimes.com/2018/03/19/technology/facebook-cambridge-analytica-explained.html
[4] Brannelly, K. (2016, November 4). "Trump Campaign Pays Millions to Overseas Big Data Firm". Retrieved from https://www.nbcnews.com/storyline/2016-election-day/trump-campaign-pays-millions-overseas-big-data-firm-n677321
[5] https://nest.com/support/article/What-is-Emergency-Contact-for-Nest-Protect
[6] https://nest.com/alarm-system/overview/
[7] https://nest.com/legal/privacy-statement-for-nest-products-and-services/
[8] Tilley, A. (2013, October 15). "Honeywell Is Giving Utility Companies Access To Your Thermostat (But Only If You Want)". Retrieved from https://www.forbes.com/sites/aarontilley/2013/10/15/honeywell-is-giving-utility-companies-access-to-your-thermostat-but-only-if-you-want/#394b9ecf679d
[9] Shankland, S. (2018, October 4). "Interest Surges in Tim Berners Lees Inrupt Startup to Remake the Web". Retrieved from https://www.cnet.com/news/interest-surges-in-tim-berners-lees-inrupt-startup-to-remake-the-web/
[10] Bansal, A. (2018, October 29). "An introduction to SOLID, Tim Berners-Lee's new, re-decentralized Web". Retrieved from https://medium.freecodecamp.org/an-introduction-to-solid-tim-berners-lees-new-re-decentralized-web-25d6b78c523b
[11] https://redux.js.org/
[12] https://github.com/N1ghtly/react-native-responsive-linechart#readme
[13] https://expressjs.com/
[14] https://nodejs.org/en/
[15] https://jwt.io/
[16] SmartBear Software. (2019, March 1). "Bearer Authentication". Retrieved from https://swagger.io/docs/specification/authentication/bearer-authentication/
[17] Provos, Niels; Mazires, David; Talan Jason Sutton 2019 (1999). "A Future-Adaptable Password Scheme". Proceedings of 1999 USENIX Annual Technical Conference: 81-92. Retrieved from https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node1.html.
[18] https://www.mongodb.com/
[19] https://mongoosejs.com/
[20] http://pi4j.com/pins/model-3b-rev1.html
[21] Dopieralski, Radomir. (2018, August 25). "Adafruit SI7021 Library Documentation: Release 1.0". Retrieved from https://media.readthedocs.org/pdf/adafruit-circuitpython-si7021/latest/adafruit-circuitpython-si7021.pdf.
[22] https://www.adafruit.com/circuitpython
[23] https://github.com/adafruit/Adafruit_CircuitPython_BusDevice