User-Friendly IoT Data Management Framework

Hyun Bin Lee, Aravind Sagar, Zicheng Li
University of Illinois at Urbana-Champaign
Urbana, United States
{lee559,asagar3,zli135}@illinois.edu

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

I. Introduction

Smaller and more power efficient processors provide more options in our life to collect information and access internet. This gave rise to the idea of "Internet of Things" (IoT).

We have observed a rapid growth of IoT market [1] in recent years, as people started to adopt this new technology. On the other hand, rapid expansion of IoT devices also brings new technical and privacy related challenges. Apart from the technical challenges like storing and processing large volume of data, and building scalable communication channels, we also face a new era of privacy-related challenges arising from devices collecting a multitude of sensitive data. Assuring privacy of these data to users is an essential step towards making IoT a commercial success [2].

There exists several issues with currently available commercial IoT ecosystems. For instance, the access control policies are enforced by a central entity and the policies themselves are often too coarse-grained. The first issue forces users to trust a single party with all their data, and at the same time this centralized design inherently leads to a single point of failure. The second issue exposes more potentially sensitive data to data requesters. Several frameworks have been proposed to address these challenges [2], [3]. These systems aim to address the privacy concerns by putting the user directly in control of their data. Decentralizing access control mechanisms, fine-grained permissioning, and explicit user consent from the user to share just the right amount of data with third parties are common themes found in such frameworks.

While these privacy-preseving frameworks can drastically improve the privacy protection of user data in IoT systems, these theoretical models overlook some key aspects that must be addressed before the implementation. One such aspect is a gap between the expectation and reality of the end user's technical knowledge. We cannot assume that a normal user can comprehend cryptographic primitives, can set up back-end components correctly, perform data aggregation and summarization, and etc. Our work directly addresses this gap.

We propose a user-friendly IoT framework, composed of front-end mobile application and a corresponding server component. This framework can help even non-tech savvy users to utlize privacy-aware, distributed, and cryptographically secure

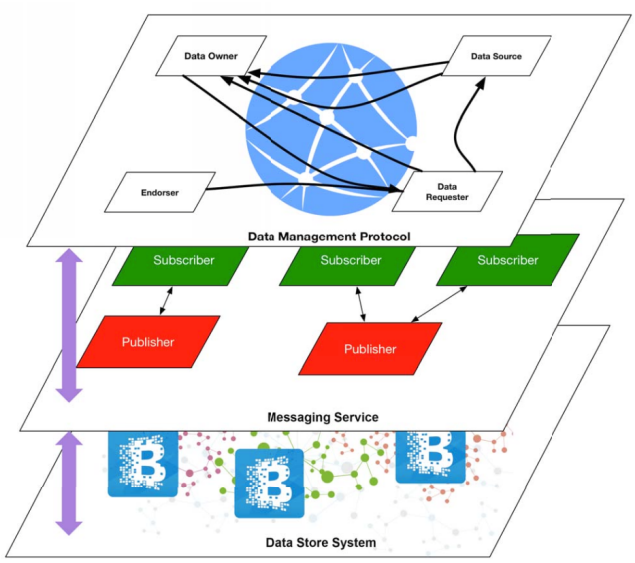


Fig. 1: Structure of the a privacy-aware IoT framework

IoT systems. We first investigate a transparent, decentralized Blockchain-based IoT data-sharing model. [3] Our app will help the users to visualize all the data owned by them, and manage fine-grained data access controls to data requesters. Each user will have a dedicated instance of the server component, which will act as a privacy-mediator [2], that enforces the data access control policies.

Our main contributions include:

1) We aim to build a user friendly application that puts the user in control of their data, and lets them understand and allow just the right amount of data to be available for third parties.
2) For the server component, we aim to make it secure from external attacks, and also design it in such a way that users have the freedom to run their own instance of the server or get an instance of it in commercial servers.

II. Background

A. Internet of Things

The term Internet of Things (IoT) originated more than 15 years ago, when it was used to describe the work of the Auto-ID Labs at the Massachusetts Institute of Technology (MIT) on networked radio-frequency identification (RFID) infrastructures [10]. The definition has since expanded to beyond the

scopr of RFID technologies. A lot of modern definitions have been proposed, one of them being "a global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [11].

The applications of IoT are diverse, ranging from smart industries to smart homes. In smart home area, thermostats, security systems, and energy management systems are particularly growing fast. In this paper, we focus on IoT technologies related to individual users, and these mostly fall under the smart home category, though the general principles are applicable in all areas of IoT.

Recent years have seen a rapid growth of smart devices available commercially. Ecosystems like Samsung SmartThings, Google Home and Apple Homekit are competing to become the primary player in personal IoT devices space, and for good reason - it is estimated that IoT could grow into a market worth $7.1 trillion by 2020 [1].

However, the rapid expansion of IoT market also means that many issues have been overlooked, and still need resolution. Some of the relevant issues are presented in 'Related Works'.

B. Privacy-Aware IoT Framework

IoT data security is crucial in protecting user privacy. Hashemi et al. [3] has proposed a scalable, privacy-aware IoT framework.

Fig. 1 shows the overall structure of the framework [3]. It is composed of three components: a Data Store System that stores the transaction of data in a distributed way (using blockchain); a Massaging Service through which data owners, data sources and data requesters communicate; and a Data Management Protocol that defines the procedures and policies of data access.

Our work focuses on the Data Management Protocol because the users do not directly interact with with the Messaging Service and the Data Store System. In the data management protocol, user data is stored in trusted entities, known as "data sources". IoT users are labelled as the "data owners". Entities who are trying to get access to user data are known as "data requesters".

The data owner is the only party who can grant data requesters the right, called "capability", to access their proprietary data. A data source needs to always allow the owner to access their data, and this is done by having the data source send an access ticket to the data owner whenever new data objects are created. The access ticket contains the data ID, the public key of the data owner, metadata and how the data can be accessed. To prove its identity, this message is encrypted with the private key of the data source.

When a data requester needs access to user data, it has to request a 'capability' to do so from the data owner through the Messaging Service. The data requester needs to send the Request ID, the its own public key and other conditions like access duration to the user. Before the information reaches the owner, the data requester can increase the credibility of its request by finding third-party endorsers to endorse this request. The endorsers would sign the request in series with their private keys, and include their public key at the end.

If the data owner decides to grant the access to its data, it would send a package to the requester. Part of the package which contains the data ID, the data access path, the capability and the public key of the data source is signed with the private key of the data owner. In addition, the request ID and a special version of the owners public key is also included in the package. The entire package is then encrypted with the public key of the data requester.

After the requester acquires a capability from a data owner, it uses this capability to fetch the corresponding data from the data source. Before the data requester sends the capability to data source, the capability is encrypted with the private key of the data requester, and the requesters public key is appended. Finally, the entire message is encrypted with the public key of the data source.

After sharing data with a data requester, the data source informs the data owner about the data access by sending a message containing the public key of the data source, the original data access ticket it sent to the data owner earlier, and part of the capability that was signed by the data owners private key. This entire message is encrypted with the public key of the data owner.

Finally, to ensure data transparency, all data transactions are recorded using Blockchains. After a successful data transaction, the data server broadcasts the transaction and the publishers update their Blockchain ledger accordingly.

C. Related Works

Davies et al. points out that privacy concerns could be a major factor preventing wide-spread adoption of IoT devices [2]. Over-centralization of IoT systems is identified as a critical obstacle to eliminating concern over privacy. They propose a privacy mediator which sits in between IoT devices/data and outside world, and validates permissions according to the access control policies before any data is sent out. This model includes a trusted 'cloudlet', which could be a local hub or a trusted server at the edge of the cloud. They also emphasize some important characteristics that a privacy-aware IoT system should have, like exposing summarized data, user anonymity, control over inferred sensors, and ease of use. However, the model's limitation is that data storage has to happen in the sensor or the cloudlet - this could be limiting when the number of sensors increase. The model also does not discuss any framework for how to identify legitimate third party applications, and how to make sure that the request originates from the same application as it is claiming to be from.

Data mining and clustering of IoT related articles expose major problem areas in IoT [4]. App over-privilege, environment mistrust, LAN mistrust and weak authentication are identified as some of the problem areas that needs work. They also conclude that permissioning needs to be more

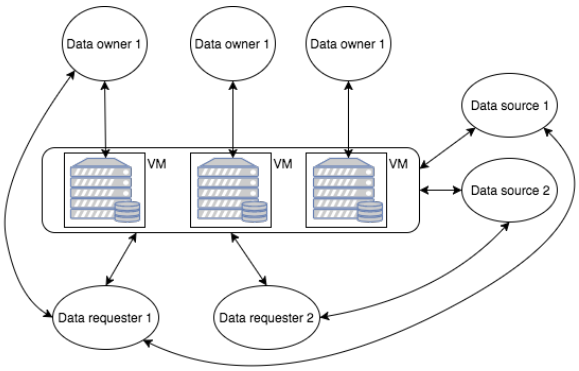


Fig. 2: Structure of the prototype framework

fine-grained, and recommend better standards and widely-applicable systems.

Security analysis of existing IoT systems show that serious vulnerabilities exist currently [14]. An analysis of Samsung Smartthings ecosystem in particular, shows that apps in the ecosystem are overpriveleged, and vulnerabilities like event spoofing can lead to privacy violation and even more serious consequences like loss of property. This goes even further in showing that a centralized proprietary system is probably not the right way to deal with IoT.

III. Problem and solution

As evident from the previous sections, effectively addressing privacy concerns about the data generated by IoT devices is essential for the success of IoT. There are some proposed architectures which tackles this problem using user-oriented and decentralized systems. However, there's a lack of a user-facing component in such systems, which can effectively make the complex architectures usable to everyone. Our work is aimed at tackling this problem, which is "to build a user-facing component for a privacy-aware, distributed, secure and user-centric IoT system (such as the blockchain based system described in [3]), which improves the user friendliness of the system while maintaining strong security and privacy guarantees."

To accomplish this, we build the user-facing component for the blockchain based IoT architecture described in "World of Empowered IoT Users [3]". Since that system is still in development, we concentrate on the user facing components of the system, and simulate some of the external components.

The proposed work consists of two components:

1) A smartphone app to manage IoT devices and data: Smartphones are becoming the preferred devices for internet access [5]. Hence we build a smartphone app with which users can manage their IoT devices and data. The functionality provided by the app includes:
   a) A centralized view of IoT generated data owned by them.
   b) A detailed view including the data source, access permissions and other details of this data.
   c) Permission request notifications from data requesters, and a way to accept or deny them.
   d) View, add, remove trusted parties.
   e) Provide user's (data owner's) public key to data requesters that provide personalized services to the user. This is not applicable for users providing anonymous data for purposes like analytics, where a different public key of the user is exposed to the data requesters.

   This is not a comprehensive list of the functionality, as we expect to add more as the project progresses. It also hides away some complexity from the users, like managing secure connection with the server described below, and managing user's private key etc. We also focus on the user-friendliness of the app, and expect to conduct user-research to find out what capabilities do users expect to find in such an app.
2) A trusted server which manages the access control: This is the component which actually manages access control and capability issuance in accordance to the user's wishes. This is similar to cloudlets described in [2], the key difference being that this server does not store data within it; it simply interacts with the data owner, data sources and data requesters in accordance to the protocol outlined in [3]. Interaction between server and data owner happens via the app described above. Interaction with other parties happen through a messaging layer, similar to the one outlined in [3]. The server ensures security in its communications using cryptographic primitives as outlined in the communication protocol.

   The server will also convert the technical specification of the data model and aggregation methods of each IoT device into user friendly texts, and uses these user friendly texts while pushing data access requests to the data owner's mobile app. We assume that the data model is agreed upon by the device and data sources. Creating required drivers for IoT devices is out of scope of this work.

   As we are focusing on user friendliness of the solution, and setting up a server may not be forte of everyone, a commercial implementation of our solution should ideally provide the users an option to automatically allocate a VM for the user in a commercial server. The users will of course have an option to use their own server instead. Another option is to have a 'hub' for each user, which will run the server process, but this solution may not be scalable for the user. We do not focus on this aspect, and instead focus on an implementation of the server which can be used for any of these purposes.

IV. Threat Model

We implement our solution based on Hashemi et al.'s [3] work for sharing IoT data objects. We particularly focus on their Data Management protocol to model communication between data requesters and data owners. Meanwhile, our work

is not limited to this protocol as we can extend our framework to any equivalent multi-party cryptographic communication protocol. In addition to four major parties involved in their protocol, we also add one additional party named Cloud Service Providers in our model. These providers are responsible for running our server-side implementation that enforces decisions made by users to regarding approvals/denials of data object requests.

1) **Data Owners**
   Our work primarily focuses on households with a small number of IoT devices. These individuals may interact with IoT devices using IoT applications written by IoT device manufacturers or other third-party programmers. Our framework does not trust these applications as they may leak sensitive data and violate Data Owner's privacy. Users grant or deny operations related with IoT data objects using our Android front-end application. Securing our application against Android malwares or compromised Android OS is out of the scope of our paper, but solutions that enforce Android application security and OS security [6] can be applied as orthogonal solutions to our work.
2) **Data Source**
   Although a Data Owner has a possession of data created by his Data Sources, the Data Owner may or may not manage Data Sources. We assume Data Sources are managed by trusted parties such that they honestly follow the data sharing protocol. On the other hand, enforcement of protocol on Data Sources is one of our future works.
3) **Data Requesters**
   We assume data requests coming from Data Requesters to be one of the major privacy threat vectors. Data Requesters query and find data objects using the Messaging service and request those data objects to Data Owners by following the Data Management protocol. Since we assumed Data sources to be honest, Data Requesters can only obtain data objects by following the protocol. Nonetheless, Data Owners may not fully understand the risks associated with approving or denying data requests. As Hashimi et al. has mentioned, the system they have proposed is user-centric such that user has full control of access control of data. In other words, Data Owners are solely responsible for granting access to all resources. This can be a very burdensome task such that it may not be scalable as we envision IoT devices to be more ubiquitous in the future. Thus, our solution primarily focuses on providing concise and accurate information for each data request and guiding Data Owners to make autonomous decisions.
4) **Endorsers**
   In our framework, trusted Endorsers are responsible for validating identity of Data Requesters and authenticity of Requesters' requests. Supplementary Information provided by Endorsers regarding the data requests may help user to make correct decisions.
5) **Cloud Service Providers**
   Trusted third party Cloud Service Providers host back-end server applications for each Data Owner such that each front-end Android application has a corresponding back-end application. Each server application is isolated from another as each runs on a separate virtual machine. For our future work, we would like to have a stronger threat model and assume Cloud Service Providers to be untrusted as well. In such case, we could enforce security and privacy of our solution using Intel's SGX [12] enclaves. Data Owner trusts server-side code run in the enclave, and they can validate integrity of the code using measurements provided with a cryptographic hash. While, our solution provides all security guarantees that are enforced by the enclaves, attacks that target security limitations of the Intel SGX processors are out of scope in our paper. These include Cache timing side-channel attacks [7] and Denial of Service attacks. Meanwhile, we believe orthogonal approaches [8], [9] that mitigate such threats can be applied to our solution.

With guidances of our user-friendly user interface, our work protects Data Owners against data requests that may violate user privacy. We believe this is the most likely threat against ordinary IoT users as we envision complexity of IoT system to increase in the future. Our ultimate goal is to provide concise yet accurate information to users such that they make intended and autonomous decisions.

V. Implementation

The implementation consists of 3 main parts: an Android app, a server manages communications with 3rd parties, and a simulator for Data sources and Data requesters according to the model in [3]. The implementations have been open-sourced under Apache 2.0 licence, and can be found in github:

- App: https://github.com/aravindsagar/cs523-iot-ui-app
- Server and simulator: https://github.com/ww785612/REST_API_encryption

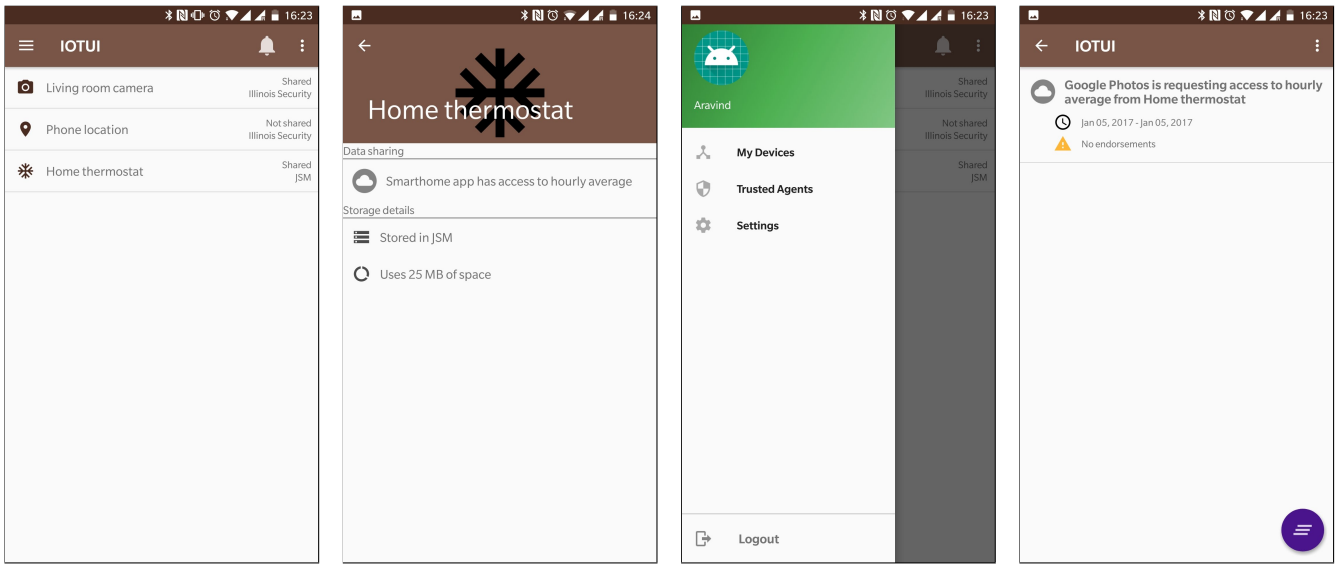
The repository used for collaborating on the reports: https://github.com/aravindsagar/cs523-docs

Each component is described in detail below.

A. Android app

The component of our solution that a user actually sees and interacts with is an Android app. The reason for going mobile is that mobile devices are now the primary gateway to computing and internet for a vast number of users [5]. Android has emerged as the most popular smartphone OS in recent years [15], so we decided to build our application for Android.

The application fulfils all the requirements noted down in the Problem and Solution section. The various screens that enable users to accomplish these functions are shown in Fig. 3 (not an exhaustive set). Fig. 3a shows the main entry point into the app. This shows an overview of the various IoT devices owned by the user, and whether the data collected by those devices are shared or not. Clicking on any of these entries



(a) Overview of user's data (b) Details of a particular device (c) Navigation drawer (d) Notifications screen

Fig. 3: UI of the user facing application

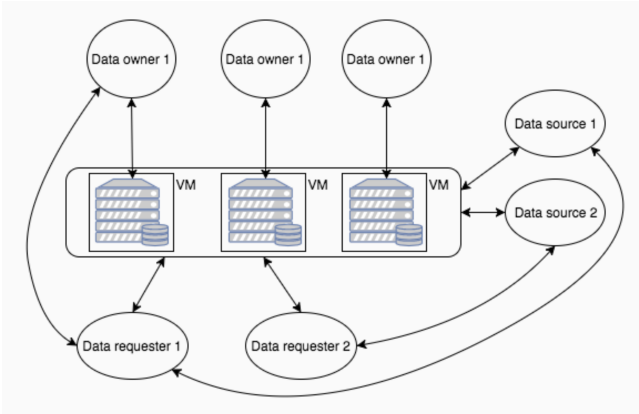


Fig. 4: App-server communication

shows the details corresponding to that device (Fig. 3b). Users can revoke granted permissions by long pressing an entry in the 'Data sharing' section. Fig. 3d shows the screen which lists the pending data requests. Users can accept or deny each request individually, or accept or reject all requests in one go using the button at bottom-right. Users also have an option to automatically accept/reject all future requests from a particular data requester for a particular device.

The app hides the complexities of collecting data from various sources and communications with the remote server. The basic communication scheme between the app and server is shown in Fig. 4. The communication happens over https, and the app sends authentication details to the server with each request.

Additionally, the app also stores certain metadata, which is collected by the server by coordinating with data sources and requesters, to present a better description of the third-parties to the user, instead of just using the public key identity of third parties. The various entities in the metadata are defined below:

1) Device: This entity represents a physical device owned by the user. The associated details like device name, type, data source used to store data etc are stored here.
2) Device data summary: This entity exists because each device can expose various temporal and/or spatial summaries of the data generated by it. It is the responsibility of each device to define its own summaries and make the data source aware of the different summaries. The data requesters are also made aware of the possible summaries so that they can request a suitable level of summarization.
3) Data source: This entity exists to identify a data source by a name, so that it becomes familiar to the user.
4) Data requester: Similar to a data source, this entity stores a name for a data requester.
5) Trusted agent: Similar to a data source, this entity stores a name for a trusted party.
6) Data access request/permission: Each access request or granted permission must link together a device data summary (the data for which request is being issued/permission is granted), and a data requester.
7) Endorsement: This keeps track of the endorsements which each data requester has.

The development of the app took place in 3 stages.

1) UI mockups and initial user study: We initially created low fidelity mockups of the app screens. This was done by borrowing app design ideas from Material design principles, as well as Rico [13]. We then ran a small scale user study where we showed the mockups to a 3 users, explained the tasks that the interface is supposed

to accomplish. We then collected their feedbacks on the strengths and weaknesses of the interface.
The study showed that the following design choices enhance the experience.

- Overall view of the devices and data is very helpful.
- Notifications screen is well done because users can see whether the requester is endorsed and act on the notification from the same screen.
- Users also feel more comfortable with IoT devices when a privacy-control mechanism like this app is present.

We also identified some major areas of concern.

- Users need more control over what data is stored. In particular, they might want to delete data between certain time periods.
- Users want to minimize the data shared with third parties, and hence each device should expose various spatial and temporal summarizations.
- Guarantee of anonymity is a must when sharing sensitive data with third parties, especially for analytics purposes.

2) Implementation: The next phase was to develop the actual app, taking into consideration the feedback that we received in the initial phase. Two of the major concerns identified in the previous phase were addressed during the implementation. Firstly, we introduced Device data summaries to make only the bare essential data available to the data requesters. Second, data requesters cannot link collcted data to data owner by default. This is a property of the protocol that we are following. However, when the user wants the data requester to provide them personalized services based on the data, they can authorize the data requester to get their public key from the app. We started off the implementation with mock data initially and hooked it up with the server later.
3) Evaluation of the implemented interface: User evaluation at this stage was done at a larger scale using Amazon mechanical turk. Details can be found in the 'User Study' section.

B. Server

We set up our server on Amazon Web Services EC2. We run a micro-instance with Ubuntu 16.04 to emulate a VM environment provided by a third-party cloud service provider. Our server-side code is written in Python Flask and is supported by Apache2 for HTTP server configuration and MySQL for implementation of the database backend. we used PyCrypto library for encryption/decryption tasks during the Data Management Protocol.

When our Android application communicates with the server, the app must provide credentials (username and password combination) via POST request. This process is processed via HTTPS connection such that app user's credentials are protected against attacks like MITM attacks. On the other hand when Data Sources and Data Requesters interact with our server, they would not require authentication as their messages are protected by cryptographic primitives set up by Hashimi et al. The server is consisted of following interfaces.

1) **/pk**
Our Android application retrieves Data Owner's public key from the server via this interface. We enforce authentication process to ensure anonymity of Data Owner. Note that Data Owner is allowed to have multiple set of public/private key pairs such that he uses different key for different purposes. This interface provides the default public key (labeled as K_O in Hashimi et al.'s work) to the application.
2) **/notify**
This interface provides information regarding status of data objects and data requests. After authentication, the server makes SQL queries to backend database and dumps the data into JSON format. Then, the query is returned as a return value of the POST request made by the application.
3) **/actions**
Data Owner makes decision on granting/denying data request through this interface. The Android application makes a POST request to this interface with authentication credentials. After verifying identity of the owner, the server validates if the decision made on a request is valid or not. Once this verification has ended, the server processes the action declared by the Data Owner and processes the result to respective Data Requester only when the request is granted. When the request is denied, no further action is performed.
4) **/dmp**
Data Sources and Data Requesters interact with our server via this interface. When message arrives with a POST request, our server first parses the message to label message type for the Data Management Protocol. Note that Data Owner can receive message 1 or 5 from Data Source and message 2 from Data Requester. Accordign to Hashimi et al., The message 2 is transferred via Publisher-Messenger Protocol for the Messaging Service layer. Unfortunately, implementation of Messaging Service is out of scope for this project, so we decided to emulate this service by letting Data Requester directly making requests via HTTP POST request. Incorporating our server with Messaging Service layer is a future work for the project.
Depending on the type of message, respective cryptographic operations are performed and metadata are parsed from the message. The metadata is written in JSON format, and the server parses the JSON string into SQL query so that a new data object or a new data request is stored into our backend MySQL database.

We assumed our frontend Android application to be honest in our threat model, so our server currently does not have defense against attacks from Application. These attacks include injection attacks via POST request such as SQL injection

attacks. Fortifying our server-side implementation against such malicious application is a future work for our project.

C. Data source and requester simulation

VI. User study

VII. Challenges and future work

There exists several challenges to implement our solution. We would like to address these challenges in this section.

1) **Simulation-based Implementation**
   Since our work is based on a theoretical model, we had to simulate each entity for simulation and implementation. During this simulation process, we ran into some implementation problems regarding details that are not explained in the literature. For instance, the paper does not specify how to incorporate multiple layers of protocols. Message 2 of Data Management Protocol is transmitted via Messaging Service, but this is also a part of Data Management Protocol. We went around this problem by allowing Data Requester to directly send requests via POST requests, but we would like to extend our framework such that it covers multiple components of the protocol. Furthermore, we were not able to prepare accurate performance benchmarks for our evaluation as we work on a simulated environment for testing our framework. Since our primary concern of the project was on effectiveness of user-interface, we did not allocate our resources on performance evaluation for this project.
   While Hashimi et al.'s core contribution on IoT data management framework was incorporating blockchain technology to the framework, we did not have an opportunity to implement blockchain onto our solution as our work only involved Data Management Protocol.
2) **Designing User Interface**
   Effective application design is an active area of research. Designing our front-end application was the most challenging task for our project. We explored Deka et al.'s [13] large repository of mobile app design to achieve our goal for the project. Unfortunately, our application did not fit with any of the category that was defined by the repository. We have examined Davies et al.'s [2] work which lists key challenges and concerns regarding how to design user policies that mediate between user and application. While their work did not directly provide a clear-cut solution to the problem, authors provide a list of possible approaches to improve effectiveness of user policies. We created our user interface based on their advices and evaluated our user interface to workers from the Amazon Mechanical Turk.
3) **User Study for App Interface**
   The challenges related to conducting a user research for the mobile app are two fold.
   First, we built an interface for which the backend is unfamiliar, and not available to interact. Users lacked some background information about the interface for which we seek feedback on. In particular, most of the IoT systems that exist now are centralized, and provides coarse-grained permissioning. By contrast, the system that we are targeting is distributed, user-centric, and fine grained in terms of third party data access permissions. While we attempted our best effort to bridge this gap, we felt that users need to know more about this fact to provide effective feedback about the interface.
   Second, IoT is still in its infancy, and it's hard to find enthusiast users of IoT devices. We overcame this by treating smartphone sensors as basic IoT devices in addition to commonly adopted smarthome IoT devices. We built a story for user interaction with the application in terms of sensor data available from smartphones and smarthome IoT devices for our user servey.

VIII. Conclusion

We present a User-Friendly IoT Data Management Framework based on Hashimi et al.'s IoT Data sharing model. We successfully implemented our solution and deployed our framework on a Android application and a VM hosted by Amazon EC2 cloud. We simulated the Data Management Protocol proposed by Hashimi et al. and successfully interacted with Data Sources and Data Requesters. Our User Study shows that our solution effectively provided summarized information to end-users to assist their decision making procedures for each data request. While our solution builds on a theoretical data sharing model, we believe our solution provides a useful insight to solving future IoT data sharing problems.

References

[1] (2015, Jun) "Internet of Things Market to Reach $1.7 Trillion by 2020". The Wall Street Journal. Available: https://blogs.wsj.com/cio/2015/06/02/internet-of-things-market-to-reach-1-7-trillion-by-2020-idc/ (Accessed Oct 25, 2017)

[2] Davies, Nigel, et al. "Privacy mediators: Helping iot cross the chasm." Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications. ACM, 2016.

[3] Hashemi, Sayed Hadi, et al. "World of Empowered IoT Users." Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on. IEEE, 2016.

[4] Zhang, Nan, et al. "Understanding IoT Security Through the Data Crystal Ball: Where We Are Now and Where We Are Going to Be." arXiv preprint arXiv:1703.09809. arXiv, 2017

[5] (2016, Nov.) "Mobile and tablet internet usage exceeds desktop for first time worldwide." Statcounter global stats. [Online]. Available: http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide (Accessed Oct 24, 2017)

[6] Ahmed M. Azab, et al. 2014. "Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World." In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14). ACM, New York, NY, USA, 90-102. DOI: http://dx.doi.org/10.1145/2660267.2660350

[7] Wang, Wenhao, et al. "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX." ACM Computer and Communications Security (CCS 17), October, 2017.

[8] Yan, Mengjia, et al. 2017. "Secure Hierarchy-Aware Cache Replacement Policy (SHARP): Defending Against Cache-Based Side Channel Atacks." In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17). ACM, New York, NY, USA, 347-360. DOI: https://doi.org/10.1145/3079856.3080222

[9] Rane, Ashay, et al. “Raccoon: Closing digital side-channels through obfuscated execution.” In USENIX Security Symposium (2015).
[10] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. “The internet of things: A survey.” Computer networks 54.15 (2010): 2787-2805.
[11] (2012, Jun) “Overview of Internet of Things.” ITU-T Recommendations. Available: http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060 (Accessed Oct 25, 2017)
[12] Costan, Victor, et al. “Intel SGX Explained.” IACR Cryptology ePrint Archive 2016 (2016): 86.
[13] Deka, Biplab, et al. 2017. “Rico: A Mobile App Dataset for Building Data-Driven Design Applications.” In Proceedings of the 30th Annual Symposium on User Interface Software and Technology (UIST ’17).
[14] E. Fernandes, J. Jung and A. Prakash. “Security Analysis of Emerging Smart Home Applications.” 2016 IEEE Symposium on Security and Privacy (SP) (2016).
[15] “Smartphone OS Market Share, 2017 Q1”. idc.com. Available: https://www.idc.com/promo/smartphone-market-share/os (Accessed Dec 14, 2017)