Security-Aware Multi-User Architecture for IoT

Marcus Birgersson
KTH Royal Institute of Technology
marbir@kth.se

Cyrille Artho
KTH Royal Institute of Technology
artho@kth.se

Musard Balliu
KTH Royal Institute of Technology
musard@kth.se

***Abstract*—IoT systems, such as in smart cities or hospitals, generate data that may be subject to different security classifications, privacy regulations, and access rights. However, popular IoT platforms do not consider data classification and security-aware data analysis.**

In this paper, we present a novel architecture based on open-source solutions that handles the issue of collecting and classifying data at the source and presents the data analysis to users at different authorization levels. Our architecture consists of three layers: a layer for exposing collected and classified data to a middleware, the middleware to handle storage and analysis of the data and expose it to a dashboard, and the dashboard responsible for authenticating users and visualizing data according to the users' classification level. Our solution distinguishes itself by focusing on data classification rather than data collection, supporting fine-grained access control and declassification. Our implementation, using the Web of Things API, Node-RED and Grafana, demonstrates the security benefits of our design on use cases in the smart city and healthcare domains.

***Index Terms*—secure IoT architecture, user-centric data classification, decentralized label model, fine-grained access control, multi-user IoT platform, data-centric architecture**

I. Introduction

IoT is a fast-growing industry with an estimated 75 billion devices in 2025, and an estimated yearly revenue of \$1.1 trillion by 2026 [1]. The large amount of data that is produced by IoT systems provides unprecedented opportunities for intelligent data analysis to boost innovation and market potential, and ultimately improve people's lives. A prime example is a *smart city*. It is expected that 70% of the world's population will live in cities and surrounding regions by 2050 [2]. The growing number of citizens requires delivering services in an efficient and effective manner, hence a large amount of data needs to be collected and processed in real time.

A key challenge lies in the security and privacy of sensitive data produced by the IoT ecosystem. Multi-user settings such as smart cities are subject to different security classification [3] and privacy regulations [4].

Furthermore, data generated by a device may not always be directly associated with its owner; for example, a smart meter may belong to an electricity service provider, but the data belongs to the consumer. Also, modern data analytics require means to control the release of data and share aggregate information; for example, a car owner may share location information as long as this information is used for the purpose of calculating traffic congestion and not for tracking the car's location. Unfortunately, current IoT platforms such as IFTTT[1], Zapier[2], and Microsoft Power Automate[3] provide all-or-nothing solutions, requiring that either all data or no data are shared, thus preventing the data owner from control of the data once it leaves the system.

We therefore need a data-centric classification model that can assign different classification labels to different kinds of data and prevent that sensitive data reaches unauthorized users. We argue that such model benefits both the data owners and data collectors. While the former would reclaim control over their data by attaching explicit policies, the latter can leverage these policies to perform useful computations in accordance with data owners' security requirements and privacy regulations. Existing solutions, as discussed in Section VII, provide coarse-grained isolation for single users and do not account for the sensitivity of the data.

We propose a novel architecture to distribute the classification of data to the devices themselves and to use this classification in a central system to automatically determine access rights on specific data, even if the system has no prior knowledge of the specific devices. This moves the capability for classifying data to the party making the data available and allows the central system, the middleware, to ensure that agreements related to data confidentiality are upheld.

Drawing on information flow control [5], [6], we develop and implement a method that builds on the decentralized label model to specify data confidentiality and prevent accidental leakage of data to unauthorized users. Our centralized middleware uses that labeled IoT data to enforce fine-grained access control policies, while allowing for user-controlled release (declassification) of sensitive data aggregates. In contrast to fully-fledged information flow control, our label-driven enforcement is flexible and lightweight, and it ensures, by construction, that only authorized users have access to the data associated with their classification label. We develop an open-source prototype [7] to implement our solution and evaluate its feasibility on different use cases.

In summary, this paper offers the following contributions:

1) We propose a data-centric model for assigning security classifications to IoT data.
2) We adopt the decentralized label model to enforce data confidentiality in a multi-user IoT setting, including controlled sharing of sensitive information.

[1]https://ifttt.com/
[2]https://zapier.com/
[3]https://flow.microsoft.com

3) We design an architecture that delegates the capability of managing data access to a central middleware.
4) We evaluate our design on an implementation based on Node-RED, Grafana, and Web of Things.

The paper is organized as follows: Section II motivates our approach and defines the problem setting. Section III presents the classification model, and Section IV presents our architecture. Section V describes our implementation, and Section VI evaluates it on common use cases. Section VII overviews related work, and Section VIII concludes.

II. Background and Motivation

A. Overview of IoT platforms

IoT platforms such as IFTTT, Zapier, and Microsoft Power Automate provide robust application support for automating the interaction and communication between Internet-connected services and smart physical devices. Rather than reinventing new protocols and standards for IoT, the Web of Things reuses well-known web standards to create an application layer for heterogeneous IoT applications [8]. The interaction is enabled by simple reactive programs running on a cloud-based IoT platform to sense and actuate data from services and devices on behalf of a single user. These programs are triggered by external information sources, as in "if the motion sensor detects movement", to perform actions on external information sinks, as in "turn on the security camera". By exposing IoT devices such as a motion detector and security camera to the IoT platform via, e.g., REST APIs, reactive programs can be used to implement user automation like "if the sensor detects movement in the apartment then turn on the security camera". A prominent standard for enabling such interactions is Web of Things [8].

B. The need for security-aware IoT platforms

Existing IoT platforms exhibit severe limitations in the areas of security and privacy. Once data leave an IoT device and reaches an IoT platform, data owners essentially lose control over their data. Data collectors, on the other hand, lack the means of identifying the sensitivity of the data, since this task is ultimately at the discretion of the data owners. Consequently, existing solutions are either overly restrictive, considering all data as sensitive and thus limiting the benefits that come with intelligent data analysis, or insecure, ignoring the sensitivity of the data and thus violating the security and privacy requirements of end-users and organizations.

In current IoT platforms, it is not possible to combine data from different users to create data aggregates without also sharing the raw data, which might be considered confidential.[4] IFTTT, the most popular IoT platform, is subject to such limitations, and so are both Microsoft Power Automate and Zapier. Consider the use case of a smart city where users would like to get statistics on the position of taxi cabs for different zones in a city. This information could benefit different stakeholders: The city could use this data to plan for infrastructure, the cab companies could send their cabs to zones with fewer cabs, while the cars' distribution in general could be used to measure traffic congestion. At the same time, no cab company wants to make the real-time position of their cabs available to the public. In current platforms, either the real-time position for each cab needs to be exposed to some third party, or the distribution cannot be computed.

The problem is further exacerbated by the multi-user nature of this setting. For example, data collected in a military context can be subject to different security classifications and should only be visible to users with the appropriate security clearance.

In this paper, we seek to address the above-mentioned challenges and propose a security-aware multi-user architecture for IoT. Our focus is on the confidentiality aspects in a large IoT system where multiple different parties contribute with data to a central system maintained by a trusted party.

We require that an authorized user is able to retrieve and visualize data from specified devices, and in some cases the ability to send commands to these devices. Ultimately, we aim to design a system that prevents unauthorized users from observing data and manipulating the system.

Our main concern is the confidentiality of the data. Collected data shall not be exposed to anyone not authorized to view data with the specified classification, neither an attacker nor a user with a lower classification level. Moreover, the system should be able to release (declassify) aggregates of sensitive data whenever all users agree on such release.

C. Secure multi-user IoT system

We assume that at least two parties are involved in collecting and distributing data. The former is responsible for installation of at least one IoT device, and that data of these devices is available to the latter. We can leverage the *Web of Things* standard [8] to create a virtual representation of an IoT device, thus providing the bridge between the IoT device and the Internet. A *Web Thing* is responsible for making data from the device available on the Internet, as well as forwarding actions to the devices when so is necessary.

The other party collects data from at least one device and is responsible for keeping data confidential according to some agreement with the first party, and based on that agreement makes the data available to authorized users. We assume a *middleware* layer positioned in the cloud and responsible for collecting data from web things representing physical devices. The middleware has the ability to both store and make computations on incoming data, which subsequently should be exposed and visualized to authorized users via a dashboard.

In general, we assume that all communication is performed on encrypted channels such as HTTPS and do not focus on the details related to how a specific IoT device communicates with the Internet. We assume that a single trusted party is responsible for the middleware and the dashboard, and hence these layers trust each other. Conversely, we allow that each smart device is maintained and installed by independent third parties with no direct relation to the middleware.

[4]Here 'raw data' refers to the data that are exposed by a device, without further modification by the network or middleware.

In summary, we target the following research questions:

1) *How can one model data classification properties in a multi-user IoT system?* In many scenarios in an IoT setting, data come from diverse sources, and users do not trust each other. All data from devices belonging to the same user may not be subject to the same level of confidentiality, which is why a user-centric access control model is not sufficient.
2) *How can the proposed architecture uphold the classification properties?* We want to ensure that the architecture restricts data flow on the network such that any access to data is authorized and follows given security policies.
3) *How can we map our classification model to existing software?* An implementation of the proposed classification model shows that the model is applicable in practice and not just a theoretical framework.

III. Classification model

To handle the above concerns, we need a decentralized fine-grained classification model as well as a valid architecture to uphold that model. The classification model needs to be flexible enough to model common cases for IoT data sharing, as well as rigorous enough to verify that data are not leaked to unauthorized users. On top of this, it should be possible to define declassification policies whenever the classification is too strict and the users agree to release aggregate information over the data without revealing the actual data.

To fulfill these requirements, we adopt the well-established decentralized label model to define and enforce security policies on IoT data [5]. This model is data-centric, and it allows to specify fine-grained access control policies consisting of the *owners* of data as well as the *readers* of that data. In a nutshell, our enhanced model combines discretionary access control (enabling flexible user-centric policy specification) and mandatory access control (enabling fine-grained policy enforcement by the middleware) in the IoT setting.

A. Data classification model

We propose a data classification model based on the decentralized label model [5]. The model associates users and data with security labels specifying which *principals* are allowed to access the data. The principals correspond to the different roles in an IoT system, e. g., a single user or a group of users, and the security labels are used to decide who can access specific data. For simplicity, we assume that a data item never changes its security label, and that the mapping between users and roles is static. In this model, principals have the ability to act as another principal according to a predefined relation modeled by an *acts-for* relationship. We assume that the relations defining what principals act for other principals are known statically. For simplicity, we assume that principals are single users, noting that the model can be generalized to roles, i. e., sets of users, in the expected manner.

We use security labels $\ell = \{o : [r_1, \cdots, r_n]\}$ to specify a security policy over a piece of data d, written d^ℓ. The policy consists of the owner o of the data and the set of data readers $r_1, \cdots, r_n$ that act for the data owner o. The owner is the principal that owns or manages the data d generated by an IoT device, while the readers are the principals that the owner allows to read the data d. Intuitively, this means that the data d can *flow to* the owner o and the readers $r_1, \cdots, r_n$. We use a function readers to extract the set of readers of a label ℓ, here $\text{readers}(\ell) = \{o, r_1, \cdots, r_n\}$. The owner is also a reader of the data, hence the security labels $\ell = \{o : [o]\}$ and $\ell' = \{o : [\,]\}$ are equivalent.

Security labels can be used to track the flow of information when computing over labeled data. For example, consider the data $d_1^{\ell_1}$ with security policy $\ell_1 = \{Alice : [Bob, Charlie]\}$ allowing $Alice, Bob$ and $Charlie$ to read d_1, and the data $d_2^{\ell_2}$ with security policy $\ell_2 = \{Bob : [Alice]\}$ allowing Bob and $Alice$ to read d_2. Then, the security policy of a computation over the data d_1 and d_2 corresponds to the security label $\ell = \{Alice : [Bob, Charlie], Bob : [Alice]\}$, namely the set union $\ell = \ell_1 \cup \ell_2$ of ℓ_1 and ℓ_2, allowing only $Alice$ and Bob to read the result of the computation, but not $Charlie$. Formally, this can be interpreted as the set intersection of the respective readers, i. e, $\text{readers}(\ell) = \text{readers}(\ell_1) \cap \text{readers}(\ell_2)$.

This model provides a powerful fine-grained mechanism for specifying security polices in a decentralized fashion.

A principal owning or managing an IoT device can attach a security label to each piece of data generated by the device at their discretion. These labels can subsequently be used when exposing the data to the centralized middleware to enforce *mandatory* security policies for the whole system. The advantage of this approach is two-fold: First, a security classification on the (virtual) device layer makes it possible for the user to decide on the security classification of their data rather than leaving that to the middleware. Second, a standardized classification system makes it possible for anyone to understand which principals can access what data and leverage such information to enforce policies globally.

B. Security policy enforcement

Using our data classification model, the middleware is now aware of the confidentiality of the data, as specified by the polices of the principals. It can leverage this information to control the propagation of data to authorized principals. This is achieved by comparing the security label of the data with the security label of the principals that observe that data. Specifically, the data $d_1^{\ell_1}$, possibly resulting from a computation over some other data, with security label ℓ_1 can be sent to a set of principals $P = \{p_1, \cdots, p_n\}$ if the readers of ℓ_1 include the set of principals P, i. e., $P \subseteq \text{readers}(\ell)$. At the same time, our classification models enables arbitrary computations over labeled data in a security-preserving manner.

We propose a fine-grained enforcement mechanism that considers a *computation function* g_P defined by a set of principals P over our dataset of labeled data; the result of the computation function is subsequently exposed to the principals in P. We can securely execute this computation function over the dataset by first extracting the data that all principals in P are allowed to read via a *trusted filtering function* f_P, and then using this data as input to the computation g_P. This procedure is secure by construction, since the computation f_P will access only data that all principals in P are allowed to read.

C. Declassification

The above-mentioned enforcement mechanism can be too restrictive in settings where principals agree to release aggregates of sensitive data in a controlled manner, also known as *declassification* [9]. The data stored in the middleware can be processed and used to create new data, which may not necessarily require the same classification level of the original data. For example, data owners may consider the location information generated from their tracking devices as sensitive, however, they might accept that such information is used to calculate the number of people in a given area and make the result available to the public. To accommodate such computations in a way that relaxes the confidentiality of data in a controlled manner, we need a mechanism that allows the data owners to explicitly authorize the use of their sensitive data in well-defined declassification functions. We will discuss this aspect further in Section IV.

Consider the example of an auction between two principals $Alice$ and Bob making secret bids $bidA^{\ell_A}$ and $bidB^{\ell_B}$ with security labels $\ell_A = \{Alice : [Auctioneer]\}$ and $\ell_B = \{Bob : [Auctioneer]\}$. The auctioneer (in our setting, the middleware) implements the following computation $f(bidA^{\ell_A}, bidB^{\ell_B})$ to determine the auction's winner:

```
if (bidA > bidB) {
   Winner = Alice;
} else {
   Winner = Bob;
}
```

The result of the auction, which is stored in variable $Winner$, should be revealed to Alice and Bob. Note that our enforcement mechanism would require to label the result of the computation as $\ell = \ell_A \cup \ell_B = \{Alice : [Auctioneer], Bob : [Auctioneer]\}$, hence the result cannot be revealed to either $Alice$ nor Bob since $\text{readers}(\ell) = \{Auctioneer\} \not\supseteq \{Alice\}$ and $\{Auctioneer\} \not\supseteq \{Bob\}$. On the other hand, we cannot use a trusted filtering function with data from only $Alice$ (or only Bob) since the auction computation $f(bidA^{\ell_A}, bidB^{\ell_B})$ requires the bids on both Alice and Bob to function correctly.

Hence, we need a way to specify a set of *trusted declassification functions* that implement computations that violate the security policies in a controlled manner, by explicitly setting the security level of the result of a computation. For our example, a declassification function $f(bidA^{\ell_A}, bidB^{\ell_B}) : \ell$, where $\ell = \{Auctioneer : [Alice, Bob]\}$, stipulates that the result of the auction is owned by the auctioneer, i. e., the middleware, and it can be read by both $Alice$ and Bob, since now $\text{readers}(\ell) = \{Auctioneer, Alice, Bob\} \supseteq \{Alice\}$ and $\{Auctioneer, Alice, Bob\} \supseteq \{Bob\}$. Moreover, we deploy such a policy with a *declassification filtering function* that feeds the declassification function f only data *owned* by either $Alice$ or Bob. As before, the filter function ensures that only data that can be declassified can be extracted from the dataset.

IV. Architecture

We propose an architecture to uphold the security requirements of the classification model described in the previous section. The architecture consists of the following layers: (1) Data collection, which comprises the set of all smart devices and services; (2) the middleware; (3) the dashboard/application layer. Figure 1 illustrates the different layers and the workflow of our architecture.

In the context of classification, the layers have the following responsibilities:

1) **Data collection:** Providing data to middleware with an initial classification of the data.
2) **Middleware:** Collecting all data and enforcing the classification model.
3) **Dashboard:** Visualizing data from middleware.

In this architecture, we abstract from the differences between devices and present a general framework for transporting data from any device to the user via the cloud.

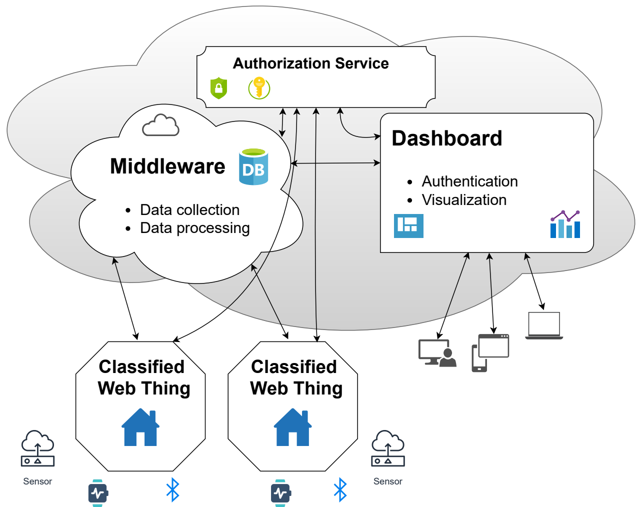

Fig. 1. Architecture for a secure IoT dashboard. The data collection layer consists of the classified Web Things, which expose HTTP endpoints to the cloud-based middleware, enabling the middleware to collect, analyze, and expose the data according to the policy. The dashboard retrieves the data and presents it to the authorized users. The authorization service verifies that each request has the correct permissions.

A. Data collection

Data can be collected by the middleware in many ways. As we will see, our implementation of this architecture collects data from the source via a REST API, of which the endpoint has been made known to the middleware beforehand. In general, the principle of the data-centric label model presented in this paper does not discriminate against other methods such as pushing data to the middleware via, for example, MQTT. We highlight REST because it is in line with current solutions adopted by the major cloud-based IoT platforms like IFTTT as well as the Web of Things model. It should also be known that many devices cannot communicate directly to the Internet, and we will in such cases assume that there exists an intermediate device which collects data from the smart device and expose this data to the middleware. We also remark that many IoT applications rely on edge nodes that transmit all or some of the data to the cloud. Securing data in transmission to the middleware or data only present on the edge nodes is handled by orthogonal techniques and is not the focus of our work. This includes the design of an optimal network architecture to connect the devices with the middleware. To collect data without a detailed specification of the components a priori, we use the Web of Things standard. It defines a structure for the REST endpoint using metadata to describe the type of data a specific endpoint is serving. We propose an extension of the Web of Things interface with a new property, *classification*. By doing so, the endpoint will not only describe what kind of data it serves, but also how sensitive the data are.

B. Middleware

The middleware is responsible for retrieving data from smart devices and services, storing[5] and analyzing the incoming data, and making it available to a dashboard for visualization. We assume the middleware is located in the cloud, and all traffic between the user and the data sources passes via the middleware. Therefore, we assume that devices do not broadcast any data, but always send it to the middleware instead.[6]

All incoming data must be analyzed based on their classification label and handled accordingly, and each classification label should be tamper-proof (this can be achieved by cryptographic signatures for the labels).

For each data point, the policies are completely independent, and thus any policy combinations are allowed. Misconfigurations need to be prevented by policy validation.

When data are aggregated, the new data must classified according to the decentralized label model. If any data point is declassified, i. e., gets downgraded to a lower classification level, it needs to be clear when this happens and where.

In our design, the main responsibility for the middleware is to collect data from the devices, perform computation over the data, enforce the classification model, and make the data available to the dashboard.

Figure 2 provides an overview of the security-aware functionality provided by the middleware. In short, two main paths are available when requesting data. The top path provides any principal with a way to request any available function to be used on data that they have access to. When the principal triggers this flow, a *trusted filtering function* ensures that only data that the principal have access to are extracted from the database. No matter what the *computation function* does, it cannot leak data that the principal are not allowed to read.

[5]We assume that the middleware has access to some form of data storage. Exactly how data are stored is orthogonal to our architecture.

[6]Reliability concerns with respect to the middleware and authentication servers as single points of failure can be addressed by replication; this is outside the scope of this paper.

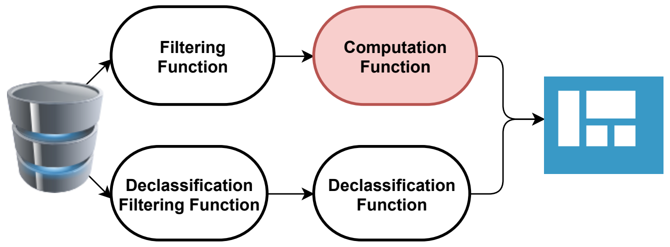
Filtering Function
Computation Function
Declassification Filtering Function
Declassification Function

Fig. 2. Overview of the two types of computation in the middleware: custom computation and data declassification. Trusted functions (in white) are provided by the middleware, while custom computation functions (in red) are untrusted. In the top path, the filtering function permits only accessible data to be used in the custom computation; in the bottom path, a declassification function receives only data owned by the principals that have agreed upon the declassification policy.

In the case when a principal wants to extract raw data, the function corresponds to the identity function. The bottom path visualizes the process of triggering a declassification procedure. This is defined by some declassification policy, and the *declassification filtering function* extracts only data that are owned by principals that have agreed to the specific declassification policy, and only data that are specified by the policy. These data are provided to the trusted *declassification function*. Before the result is returned, the declassification function verifies that the requesting principal has the correct access rights for the declassified data.

a) Custom computation: Since it is of key importance that the classification model is respected, and since the classification model provides a simple way of knowing which principals that are allowed to access what information, we use a trusted filtering function that provides data from the database. This function ensures that only data accessible by a specific principal are served, and hence only this function has to be trusted to prevent data leakage. Given this function, we can use it to serve data to principals without risking that any principal would access data that they should not be allowed to access.

In addition, filtering functions also open up the possibility to let any principal deploy untrusted (potentially malicious) functions running in a sandbox in the middleware. These functions do not need to be trusted, since they would be placed after the filtering function and hence only be allowed to operate on data that the invoking principal already can access.

b) Declassification: The architecture provides a mechanism for performing computations over data with different classification labels without revealing the actual data used in these computations. We achieve this by describing precisely which principals need to agree to contribute with their sensitive data to some computation, thus assigning a new (less sensitive) classification label to the result of the computation. In this setting, the principals have to explicitly give permission that their (owned) data are used in a predefined declassification function. If they do, the middleware has the ability to execute these functions and declassify the result according to the declassification policy associated with the declassification function.

The declassification policy consists of a specification of what dataset that should be used, what function (the associated declassification function) should be able to use the dataset, and finally what the resulting classification label should be. Note that all owners of a specific data point need to agree on releasing the declassified information. For example, a set of principals may agree to a policy stating that the principals' heights can be used in a declassification function that computes the mean value and makes it available publicly, without revealing the height of a specific principal. The middleware uses a declassification filtering function to first extract the *height* from the dataset only for the principals that have agreed to the policy, and then feeds this data to the declassification function to compute the mean value.

C. Dashboard

The dashboard is the entry point for a user and is a separate service from the other layers. This could be either a web interface accessed by a browser, or an app on a user's phone.

The user accesses the system by the use of login credentials. When a user is authenticated, the main responsibility for the dashboard is to visualize data that the user is authorized to view, and allow the user to perform authorized actions on the system.

D. Authentication/Authorization service

To connect users with the permissions set by the labels, the authentication/authorization service needs to map individual users to principals. Role-based access control [10] fulfills this requirement. In our implementation, authentication and authorization are currently hard-coded. Keycloak [11] provides an open-source solution that manages user accounts and their roles; we plan to adopt it in the future.

In this paper, we assume the existence of such a service, but will not cover it in detail, and neither key/identity management.

E. Workflow

Figure 3 visualizes the workflow of an authorized user connecting to the dashboard to retrieve data from the system or to perform an action. In general, the process works as follows:

1) *Data collection:* Data from smart devices with classification label are provided to the middleware.
2) *Login:* The user logs in to the dashboard.
3) *Request data:* The dashboard requests data that are accessible to the current user from the middleware.
4) *Process request:* The middleware checks that the user is authorized to retrieve the requested data and returns them to the dashboard.
5) *Visualization*: The dashboard visualizes the data.

V. Implementation

To demonstrate and validate our design in a real environment, we have built a secure IoT system using a modified

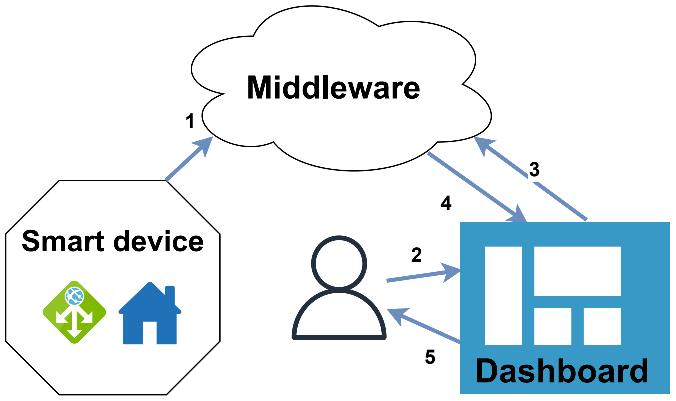

Fig. 3. Example flow for an authorized user. Data are provided to the middleware from the smart devices and services. The user logs in to the dashboard. The dashboard authenticates the user and requests data from the middleware. The middleware verifies the authorization for the user and returns the data, while enforcing the classification model.

open-source implementation of the Web of Things standard[7], Node-RED[8] to act as the middleware, and Grafana[9] for authentication and as the dashboard. The complete source code is available on GitHub [7].

A. Data collection

Our implementation creates the data collection layer using instances of Python services serving a Web of Things API. This implementation has then been modified to add an extra parameter, *classification*, on each device. The classification parameter is implemented as a dictionary and initialized to $\ell = \{o : \mathbf{r}\}$, where o is the name of the principal owning the data from the device, and $\mathbf{r}$ is a list of readers allowed to access the data from that device. Each Web Thing has its own endpoint and continuously exposes its data via a REST endpoint.

B. Middleware

The middleware is implemented using an instance of Node-RED, "a programming tool for wiring together hardware devices, APIs and online services" [12]. Node-RED is an open-source JavaScript-driven IoT platform for customizing complex data flows from IoT devices and services. This layer contains most of the application logic and is responsible for upholding the classification model. The middleware addresses four key tasks:

1) Fetching Data: The middleware is configured with the root URLs for each Web Thing and uses the standard to fetch devices and exposed data in a regular manner.

The fetched data are sanitized and stored as JSON objects in an instance of the NoSQL database MongoDB[10]. MongoDB supports queries to its database in JavaScript [13], which

[7]https://github.com/mozilla-iot/webthing-python
[8]https://nodered.org/
[9]https://grafana.com/
[10]https://www.mongodb.com/

allows us to implement advanced constructs on which entries should be returned.

2) Transforming Data: Pre-configured trusted functions have been deployed to perform common computations. The implemented functions can perform two main type of computations, mean value computation and computation of distributions of numbers. These functions are implemented as separate *nodes* in Node-RED and written in JavaScript.

3) Exposing raw data: The middleware exposes an API to give principals the possibility to request data from the database. In each request, one needs to include the requesting principal that will be used as a key to extract data authorized for that user. To make sure that only data accessible by the requesting entity are available, a filtering function extracts only data that the principal are allowed to access. MongoDB evaluates the function on each object in the database and gives the function the predefined properties as arguments. In our example, the query is specified as follows:

```
query = {"$expr": { "$function": {
      "body": func,
      "args": [ "$classification" ],
      "lang": "js"
} } }
```

We specify that query is a *function* and that MongoDB should evaluate the function using the property *classification* as argument for each object in the collection. The function is defined as follows:

```
func = function(classification) {
    for (var owner in classification){
        r = classification[owner];
        isReader = r.includes("$principal");
        isOwner = ("$principal" == owner);
        if (!isOwner && !isReader){
            return false;
        }
    }
    return true;
}
```

This function uses the classification property on each object and checks that for each policy in the property that the requesting principal is either an owner or a reader. If the function returns true, the object is returned, otherwise not.

By always using this function to query data, we ensure that the requester will only access data that it is allowed to view, independently of what other functions would do with these data.

We remark that in a real system, it is of great importance to sanitize the inputs in these queries, and the raw user inputs should never be included as is in any query to the database. On the same note, the keys for identifying entities should be handled using a real authentication service rather than the name of a principal, and these keys should only be valid for a limited period of time.

4) Exposing declassified data: Declassification requests are made towards a REST endpoint and consist of an argument including the requesting principal.

The middleware stores a pre-configured list of principals which have agreed to the specific declassification policy. We can then use this list to create a function query which will only extract data where all policy owners have agreed on the declassification policy.

```
function(classification) {
    for (var owner in classification){
        if (!$owners.includes(owner)){
            return false;
        }
    }
    return true;
}
```

In this function, variable *$owners* is substituted with the actual list of principals that have agreed on the declassification policy. This query is then applied on the specified dataset from the declassification policy. This ensures that only data belonging to the principals that have agreed to the declassification policy are returned from the database, and only data from the specified dataset are used. The resulting data are then sent to the specified function and given the new classification label. Before data are released to the requester, it is checked that the requesting principal actually has the right to access the data according to the new label.

C. Dashboard

The dashboard that is used is an instance of Grafana, which is open-source and handles authentication out of the box. After a user logs in to the dashboard, predefined graphs show the data fetched from the middleware depending on what principal that user belongs to. By using a plugin, we are able to configure the REST endpoints towards the middleware, including different principals as keys in the query. These queries then result in different datasets in pre-configured graphs that visualize the data for the user in a suitable way.

The data that are available in the dashboard hence only consist of computations where the current user is included as a reader in the corresponding policies.

VI. Evaluation

We have used our implementation described in the previous section to evaluate different use cases, which are inspired by real-world applications and developed to show different types of capabilities of our classification model.

A. Position sharing for taxis

In this scenario, a smart city wants to measure traffic congestion by using position sensors on taxi cabs. These sensors are managed by the companies themselves but transmit the data to a middleware managed by the city. The middleware should collect data from individual taxi cars and compute how many cars there are in each zone of the city at the current time. Figure 4 shows how data are allowed to flow in this aspect.

The dashboard is responsible for providing this information to authorized users. Depending on the user, the available information will be different. If the user is a representative from a cab company, that user can access real-time position data from each car of that company, and can hence determine where more cars are needed. A member of the public, on the other hand, should only be allowed to access the distribution to know how many taxi cars there are in a certain zone.

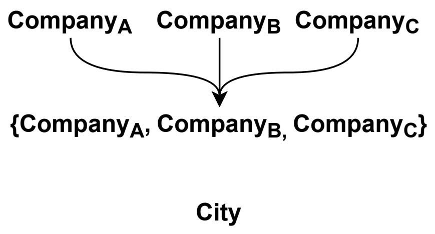
Company_A Company_B Company_C
{Company_A, Company_B, Company_C}
City

Fig. 4. Allowed information flow in the taxi use case

1) Data collection: Each taxi car is represented as a Web Thing endpoint and exposes a simulated position value produced by simulated 'location sensors' using a value between 1 and 4 to represent their zone. The devices are managed by three different cab companies. In this implementation, each company adds a classification label consisting of a single policy with itself as owner:

$$\ell_i = \{Company_i : []\}, i \in \{A, B, C\}$$

This policy makes the data readable only by the specific cab company. In this way, each cab company can access their own data from the cloud, while keeping it secret from anyone else.

2) Middleware: The middleware collects data from all cab cars via their Web of Things endpoint. The standard allows the middleware to automatically find all devices and data properties exposed by knowing only the root address of the endpoint. The data are collected together with their classification label and stored in the cloud.

To compute the distribution of cab positions, we use a predefined trusted function in the middleware and feed it the position data from all companies. By the means of the decentralized label model, this distribution gets the following label:

$$\ell_{dist} = \{Company_A : [], Company_B : [], Company_C : []\}$$

This label is in the general case not readable by anyone.

To be able to expose the distribution to the public, each cab company has agreed upon a declassification policy for this purpose. In this context, the classification policy specifies the dataset *position*, and a trusted function f_{dist} which computes the distribution of the taxi cabs position. The new label is defined as $\ell'_{dist} = \{City : [Public]\}$. The declassification endpoint can now use the list of principals that have agreed on the policy, which in this case are all three cab companies, and only query position data where they are the only policy owners. The function will compute the distribution and apply the new label ℓ'_{dist} which will make the principal *City* the owner of the data, and allow anyone access to them.

Since the declassifier always queries *only* data owned by *only* these companies, and only for this specific dataset, potential other principals that have not agreed will not get their data leaked. It is up to the principals to confirm that f_{dist} computes (aggregated) data in a way that is consistent with the intended declassification. We currently assume that the implementation of a declassifier is trusted. This assumption can be lifted by employing orthogonal techniques such as quantitative information flow [14] or differential privacy [15], which we defer to future work.

In our current implementation, the list of principals that have agreed on the policy is predefined in the middleware. In a real setting, this would be supported via a web interface where the users can agree on policies to release this kind of data. The middleware can then use data from all users that have agreed upon the policy.

3) Dashboard: Depending on which user logs in, different graphs will be shown. Each graph is predefined for each user and requests data based on that user. For simplicity, our current implementation uses the current principal as the key in the query. In a real setting, this key will be the authentication token one gets when logging in. For a cab company, a graph will show the exact position of each cab. The request fetches all position data that belong to that principal, and the middleware ensures that the correct data are served. For a user not associated with a cab company, a graph will show the aggregated distribution of the cabs for each zone.

Figure 5 shows how the dashboard looks like for the principal CompanyA. Here, one panel is configured to visualize the public view of the distribution of all cabs in each zone, and another panel shows the real-time position of cab A1.

B. Hospital Smart Portal

The health sector is a growing market for different smart products where many actors such as Fitbit, Withings, Apple, etc. are marketing products that measure pulse, blood pressure, workouts, and sleep metrics. In this aspect, we see a potential for the hospitals to use these data and at the same time store their own related patient data. Such a system would enable medical personnel (even between different hospitals) to use data that an individual has collected on their own; at the same time, the patient could access their latest measurements collected by the hospital via an online portal.

Another scenario entails the use of such data for research in different projects, without participants needing to go through similar processes to allow data sharing, which would make it both more time- and cost-efficient.

We here present a simplified scenario for how to collect and share different health-related data. Note that in a real-world scenario, the number of stakeholders for specific data might be a lot larger than presented here, and there might be local regulations that complicate sharing specific personal

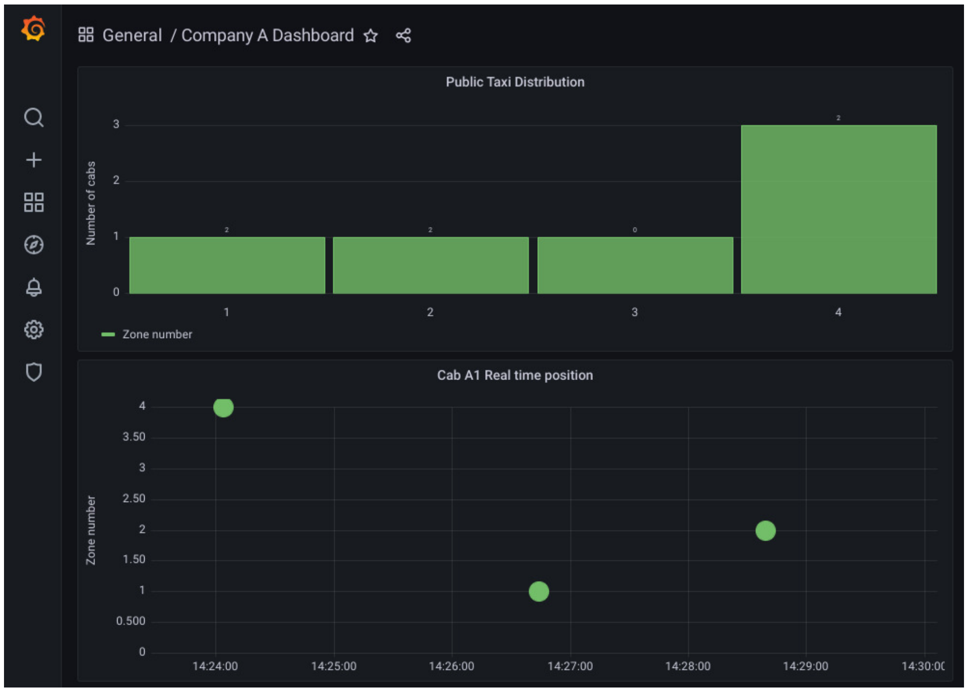

Fig. 5. Example view of dashboard for company A. Here, the company has chosen to see the publicly available distribution of all cabs and the real-time position of their cab A1. We can see that there is one cab in each of the Zones 1, 2 and 3, and three cabs in Zone 4, and the latest reported zone for cab A1 was Zone 2.

data. Here, we want illustrate the potential of our model in this security-sensitive context, where complex use cases can be implemented because of the fine-grained control that comes with the decentralized label model.

1) Data collection: In this use case, we have two patients which act as principals $Patient_A$ and $Patient_B$. Both patients measure their pulse data continuously during the day and make these data available to the middleware as Web of Things endpoints. $Patient_A$ wants their doctor to be able to directly access the pulse data. The doctor is part of the $Doctors$ principal, and hence the classification label will be set as:

$$\ell_A = \{Patient_A : [Doctors]\} \tag{1}$$

This label sets the patient as owner of the data but allows the $Doctors$ principal to read the data. In this example, the patient has the ability to access the data from a central system managed by the hospital. At the same time, the personnel that can act as the $Doctors$ principal can access the raw pulse data and then use them to diagnose the patient or call in the patient for additional tests if anything in the data looks wrong.

$Patient_B$, on the other hand, does not want anyone else to access the pulse data, and hence configures the following label:

$$\ell_B = \{Patient_B : []\} \tag{2}$$

The patient mainly uses the central system managed by the hospital as a secure way of storing the data in the cloud with the ability to access it via a dashboard. No one except for $Patient_B$ should have access to these data.

2) Sharing data: When data are managed in the central system, it should be possible to securely share the data with trusted parties. This is managed by using *declassification policies*, where new datasets can be created based on existing data and declassified if the owners of the current policies of the data have granted permission.

In Figure 6, we can see examples of how the data in general is allowed to flow in the system.

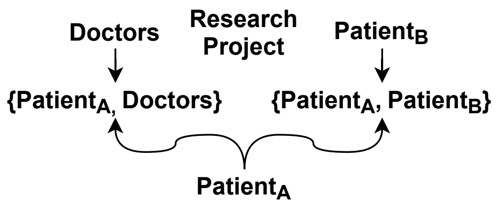

Fig. 6. Example of allowed information flow

a) Sharing the mean value of a patient: $Patient_A$ wants to share the mean value of their measurements with a trusted research company. To do so, the trusted company has defined a declassification policy stating that they need the principal's *pulse* data; they want to feed it to the trusted function f_{mean} in order to apply the new label $\ell_R = \{ResearchProject : []\}$ in the end.

When the user acting as principal $Patient_A$ accepts this policy, the declassification endpoint for the mean data will include the patient's data in the query, compute the mean value by feeding the data to f_{mean}, and finally apply the new label ℓ_R on the result. This dataset will only be available to the research company and no one else. Note that it is only the policy owner $Patient_A$ that needs to approve the policy. While the principals in the readers list have access to the data, they are not in control of the policy itself.

b) Sharing the mean value of multiple patients: Both $Patient_A$ and $Patient_B$ agree to share the mean value of their respective data with a trusted research project, but neither wants any other party to see their raw data. In this case, the research project has deployed a trusted function f'_{mean} in the middleware that computes the mean value over multiple datasets. Note that if the data owned by $Patient_A$ and $Patient_B$ were used in this function according to the decentralized label model, the label for the aggregated data would be

$$\ell_{AB} = \{Patient_A : [Doctors], Patient_B : []\} \tag{3}$$

which in the general case is not readable by anyone. To be able to release these data to the research company, both patients have agreed to a policy stating that their *pulse* data will be fed to the function f'_{mean}. Label $\ell_R = \{ResearchProject : []\}$ is then applied to the result. When both patients have agreed to this, the middleware will be able to query all *pulse* data owned by only $Patient_A$ or $Patient_B$, feed that data to f'_{mean}, and apply the label ℓ_R to the result. This enables the research company to access the data, but neither $Patient_A$ nor $Patient_B$ are allowed to do so. This makes the data protected from the other parties, since if any of the principals $Patient_A$ or $Patient_B$ were allowed to access the result, they would be able to infer the other principal's raw data.

C. Discussion

By using the decentralized label model for fine-grained access control, it is possible to automatically determine access when aggregating data, while avoiding accidentally releasing information to unauthorized parties.

Our model prevents the release of confidential data to any users that try to extract data they are not allowed to read. It allows any party to deploy any function to the middleware without risk of leaking information, since the system provides such a function only with data that the requesting principal already can access.

A user can only access data that they are allowed to read given its labels. Inadvertent data sharing outside our platform is impossible to prevent (as external channels can always leak data) and does not affect data that other users keep confidential. To this end, we rely on the correctness of the trusted filtering function and the declassification functions, as well as on the correct usage of the labels.

In general, our model can be used for fine-grained decentralized access control to protect leakage of data in an environment where datasets are aggregated and originate from multiple different parties. The model has the ability to protect data while at the same time making it possible to release information in a controlled way for secure data sharing. The model is flexible because of the fine-grained access control provided by the decentralized label model. It decentralizes the access control of the data to the owners, rather than delegating it to the central system as in many other RBAC models [16] today. It is lightweight with regard to using the decentralized label model as a means for dynamic access control rather than information flow control, which is computationally more expensive.

We assume that users send correct data (or else withhold data) but not data that are designed to skew aggregated statistics. Techniques that detect anomalies in data sets or prevent statistical methods from extracting individual data from aggregated values are orthogonal to our work.

1) How can one model data classification properties in a multi-user IoT system? When multiple parties are involved in contributing data in a larger system, every party is responsible for labeling the data before submitting. When data are included in the larger system, the classification labels must be respected based on predefined rules. By forcing every device in the system to have a classification label before the data are submitted to the cloud, or using the most restrictive label as a default setting, it is possible to automatically add and remove devices from the system while still enforcing the classification of each individual data point.

2) How can the proposed architecture uphold the classification properties? We design our architecture (see Section IV) such that it respects information flow restrictions outlined earlier:

1) Every data point must have a label and hence also defines a set of its owners.
2) We use labels to express what level of restriction data are subjected to.
3) Every request to the middleware is properly authenticated.
4) Aggregated data get a new classification label based on the decentralized label model.
5) Every policy owner has to agree before data can be declassified.

3) How can we map our classification model to existing software? To show that the model is valid in a real setting, we have developed an implementation using existing open-source technologies. The architecture and the classification model have then been evaluated on real-world-inspired use cases. More case studies, especially with respect to the adaptability of our framework to different data security policies, are future work.

VII. Related work

We discuss closely related work on securing IoT platforms, cloud security, and IoT security architectures. We refer to overview papers on securing IoT platforms [17], [18], information flow control for cloud computing [6] and a survey on IoT security [19] to help the reader navigate on these topics.

Popular IoT platforms include If This Then That (IFTTT), Microsoft Power Automate, and Zapier. IFTTT, the most popular of these platforms, supports more than 550 types of Internet-connected IoT devices and services, 11 million users, and 54 million apps; one billion instances run per month [17]. These platforms leverage centralized cloud-based systems to connect IoT services and devices. However, they lack support for data classification and security-aware data sharing. Moreover, they enforce data access on a per-user basis, thus making it challenging to define flexible multi-user policies. Recent works study security and privacy risks in IoT platforms [20], [21] and propose access control [22], [20] and information flow control [21] to prevent such risks. By contrast, our work focuses on the data-centric multi-user setting and enforces security via fine-grained access control, rather than information flow control, which is more lightweight and flexible.

Our middleware relies on the Node-RED platform to automate the interaction between the data collection layer and the dashboard. Although our implementation of computation functions, filtering functions, and declassification functions does not rely on third-party Node.js packages, Ahmadpanah et al. [23], [24] show that Node-RED can be subject to attacks from third-party code. They propose SandTrap, a JavaScript-based monitor that enables fine-grained access control policies. SandTrap can be helpful in our setting to prevent attacks from malicious code.

Ancona et al. [25] propose runtime monitoring of trace expressions to ensure the correct API usage of Node-RED functions. They study a rich policy language including temporal patterns over sequences of API calls. Kleinfeld et al. [26] develop an extension of Node-RED called glue.things, focusing on usability of trigger and action nodes. These works are complementary and can be leveraged to accommodate more convoluted use cases in our middleware, if needed.

A wide array of works propose decentralized information flow control (DIFC) to enforce end-to-end security policies at the level of applications [27], [5], [28] and operating systems [29], [30]. Notably, the line of work by Bacon et al. [6], [31], [32], [33] applies and develops DIFC in the context of secure cloud computing with applications to the IoT domain. They build a kernel module providing information-flow abstractions to enforce security of processes running in the cloud. Our focus is on lightweight fine-grained access control, aiming at securing data in SaaS (Software as a Service) cloud-based IoT platforms. Recent work by Islam et al. [34] proposes the use of Trusted Execution Environments (TEEs) such as Intel SGX to perform IoT data analytics in the cloud. While TEEs avoid trusting the cloud-based middleware with users' sensitive data, enforcement mechanisms like ours are still needed to ensure security in a multi-user setting.

Existing works proposing IoT architectures distinguish between three to five layers [35], [36], [37]. These layers can be classified as: a *perception layer* to collect data from sensors, the *Internet layer* to expose the data to the Internet, a *middleware* consisting of one or two layers to store and compute over data in the cloud, and an *application layer* to present the data. For security, much focus has been on the perception layer and how data can be transmitted to the cloud securely [38]. Other works focus on security of transporting data from the cloud to the users [39]. These works are complementary to our work, which focuses mainly on the data itself, on how multiple (potentially mutually distrusting) parties can distribute security-labeled data to a shared cloud, and on how to compute over this data without exposing it to unauthorized parties.

VIII. Conclusion

Data from IoT devices may be subject to different levels of restrictions, which do not necessarily follow the identity of the user who owns a device. Popular IoT platforms do not consider such a data classification.

We propose to extend the data of Web Things with a label that represents its classification. We also describe and implement an architecture where data transmission is restricted to communication between Web Things and central middleware, as well as middleware and a dashboard that provides the interface to the user. All communication is subject to authentication and authorization, which is managed by a dedicated service.

We implement this architecture using existing open-source components and demonstrate its functionality in the context of a smart city and a hospital.

Future work will consider more complex use cases and technical methods to address the needs that arise when using our architecture in the field. These include validation of the labels and their enforcement in the middleware using information flow analysis, and the use of machine learning to detect anomalous behavior of compromised devices. Finally, support for updating labels and user permissions dynamically is also left as future work.

References

[1] Maayan, G.D.: The IoT rundown for 2020: Stats, risks, and solutions. Security Today

[2] Jin, J., Gubbi, J., Marusic, S., Palaniswami, M.: An information framework for creating a smart city through Internet of Things. IEEE Internet of Things Journal **1**(2) (2014) 112–121

[3] National Institute of Standards and Technology. In: Department of Defense Trusted Computer System Evaluation Criteria. Palgrave Macmillan UK, London (1985) 1–129

[4] GDPR: General Data Protection Regulation. https://gdpr.eu/ Accessed: 2021-08-31.

[5] Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Trans. Softw. Eng. Methodol. **9**(4) (October 2000) 410442

[6] Bacon, J., Eyers, D.M., Pasquier, T.F.J., Singh, J., Papagiannis, I., Pietzuch, P.R.: Information flow control for secure cloud computing. IEEE Trans. Netw. Serv. Manag. **11**(1) (2014) 76–89

[7] Birgersson, M.: secure-iot-architecture prototype. https://github.com/marbirg/secure-iot-dashboard-prototype Accessed: 2021-08-31.

[8] W3C: Web thing model (April 2021)

[9] Sabelfeld, A., Sands, D.: Dimensions and principles of declassification. In: 18th IEEE Computer Security Foundations Workshop (CSFW 2005). (2005) 255–269

[10] Sandhu, R.S.: Role-based access control. In: Advances in Computers. Volume 46. Elsevier (1998) 237–286

[11] Keycloak: Open source identity and access management (August 2021)

[12] OpenJS Foundation: About. https://nodered.org/about/ Accessed: 2021-08-31.

[13] MongoDB Inc: $function (aggregation). https://docs.mongodb.com/manual/reference/operator/aggregation/function/ Accessed: 2021-08-31.

[14] Smith, G.: On the foundations of quantitative information flow. In: Foundations of Software Science and Computational Structures, Springer Berlin Heidelberg (2009) 288–302

[15] Dwork, C.: Differential privacy. In: International Colloquium on Automata, Languages, and Programming, Springer (2006) 1–12
[16] The Apache Software Foundation: Apache Atlas (August 2021)
[17] Balliu, M., Bastys, I., Sabelfeld, A.: Securing IoT apps. IEEE Security & Privacy Magazine (2019)
[18] Celik, Z.B., Fernandes, E., Pauley, E., Tan, G., McDaniel, P.D.: Program analysis of commodity IoT applications for security and privacy: Challenges and opportunities. ACM Computing Surveys (2019)
[19] Alaba, F.A., Othman, M., Hashem, I.A.T., Alotaibi, F.: Internet of Things security: A survey. Journal of Network and Computer Applications **88** (2017) 10–28
[20] Fernandes, E., Jung, J., Prakash, A.: Security analysis of emerging smart home applications. In: 2016 IEEE Symposium on Security and Privacy (SP). (2016) 636–654
[21] Bastys, I., Balliu, M., Sabelfeld, A.: If This Then What?: Controlling flows in IoT apps. In: CCS. (2018)
[22] Fernandes, E., Rahmati, A., Jung, J., Prakash, A.: Decentralized Action Integrity for Trigger-Action IoT Platforms. In: NDSS. (2018)
[23] Ahmadpanah, M.M., Hedin, D., Balliu, M., Olsson, L.E., Sabelfeld, A.: SandTrap: Securing JavaScript-driven trigger-action platforms. In: 30th USENIX Security Symposium (USENIX Security 21), USENIX Association (August 2021) 2899–2916
[24] Ahmadpanah, M.M., Balliu, M., Hedin, D., Sabelfeld, A.: Securing Node-RED applications. Protocols, Strands, and Logic: Essays Dedicated to Joshua Guttman on the Occasion of his 66.66th Birthday (2021)
[25] Ancona, D., Franceschini, L., Delzanno, G., Leotta, M., Ribaudo, M., Ricca, F.: Towards Runtime Monitoring of Node.js and Its Application to the Internet of Things. In: ALP4IoT@iFM. (2017)
[26] Kleinfeld, R., Steglich, S., Radziwonowicz, L., Doukas, C.: glue.things: a Mashup Platform for Wiring the Internet of Things with the Internet of Services. In: WoT. (2014)
[27] Myers, A.C.: JFlow: Practical mostly-static information flow control. In: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL '99, New York, NY, USA, Association for Computing Machinery (1999) 228241
[28] Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications (2003)
[29] Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazières, D., Kaashoek, F., Morris, R.: Labels and event processes in the Asbestos operating system. In: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles. SOSP 2005, New York, NY, USA, Association for Computing Machinery (2005) 1730
[30] Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M.F., Kohler, E., Morris, R.: Information flow control for standard OS abstractions. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles. SOSP 2007, New York, NY, USA, Association for Computing Machinery (2007) 321334
[31] Pasquier, T.F.J., Bacon, J., Eyers, D.M.: Flowk: Information flow control for the cloud. In: IEEE 6th International Conference on Cloud Computing Technology and Science, CloudCom 2014, Singapore, December 15-18, 2014. (2014) 70–77
[32] Singh, J., Pasquier, T.F.J., Bacon, J., Ko, H., Eyers, D.M.: Twenty security considerations for cloud-supported Internet of Things. IEEE Internet Things J. **3**(3) (2016) 269–284
[33] Pasquier, T.F.J., Singh, J., Eyers, D.M., Bacon, J.: Camflow: Managed data-sharing for cloud services. IEEE Trans. Cloud Comput. **5**(3) (2017) 472–484
[34] Islam, M.S., Özdayi, M.S., Khan, L., Kantarcioglu, M.: Secure IoT data analytics in cloud via Intel SGX. In: 13th IEEE International Conference on Cloud Computing, CLOUD 2020, Virtual Event, 18-24 October 2020. (2020) 43–52
[35] Sethi, P., Sarangi, S.R.: Internet of Things: Architectures, protocols, and applications. Journal of Electrical and Computer Engineering **2017** (Jan 2017) 9324035
[36] Sreeram, M., Sreeja, M.: A novel architecture for IoT and smart community. In: 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC). (2017) 486–491
[37] Fremantle, P.: A reference architecture for the Internet of Things. Technical report, WSO2 (2015)
[38] Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M.: Internet of Things for smart cities. IEEE Internet of Things Journal **1**(1) (2014) 22–32
[39] Bokefode, J., Bhise, A., Satarkar, P., Modani, D.: Developing a secure cloud storage system for storing IoT data by applying role based encryption. Procedia Computer Science **89** (12 2016) 43–50