HomePad: A Privacy-aware Smart Hub for Home Environments

Igor Zavalyshyn*†, Nuno O. Duarte*, Nuno Santos*
*INESC-ID / Instituto Superior Técnico, Portugal
†Université Catholique de Louvain, Belgium
igor.zavalyshyn@student.uclouvain.be
{nuno.duarte,nuno.m.santos}@tecnico.ulisboa.pt

***Abstract*—The adoption of smart home devices is hindered today by the privacy concerns users have regarding their personal data. Since these devices depend on remote service providers, users remain oblivious about how and when their data is disclosed and processed. In this paper we present HomePad, a privacy-aware smart hub for home environments. Our system aims to empower users with the ability to determine how applications can access and process sensitive data collected by smart devices (e.g., web cams) and to prevent applications from executing unless they abide by the privacy restrictions specified by the users. To achieve this goal, HomePad applications are implemented as directed graphs of *elements*, which consist of instances of functions that process data in isolation. By modeling elements and the flow graph using Prolog rules, HomePad allows for automatic verification of the application's flow graph against user-defined privacy policies. Homepad incurs a negligible performance overhead, requires a modest programming effort, and provides flexible policy support to address the privacy concerns most commonly expressed by potential smart device consumers.**

***Keywords*-smart home; privacy; internet of things; data flow analysis; smart home devices.**

I. Introduction

One of the biggest barriers to the widespread adoption of smart devices for home environments involves concerns about privacy. Today, smart devices compatible with IoT frameworks such as Samsung SmartThings [1] or Apple HomeKit [2] depend on Internet connectivity in order to provide useful service. Numerous smart home devices, from smart lights and locks to thermostats and cameras constantly stream their sensor data to service providers' remote servers for processing, backup, and remote access and control. However, end-users have little knowledge or control about how much or what kind of data is collected by service providers, nor do users know for what purpose the collected data will be used or with whom it will be shared.

Moreover, the terms of use of IoT services tend to be extremely aggressive. For example, the following can be read from Samsung's SmartThings terms of use, effective since April 2017: "*you [the user] hereby do and shall grant SmartThings a worldwide, non-exclusive, perpetual, irrevocable, royalty-free, fully paid, sublicensable and transferable license to use, modify, reproduce, distribute, share, prepare derivative works of, display, perform, and otherwise fully exploit the User Submissions and Device Data in connection with the SmartThings's Services*" [3]. In practice, users have to yield full control of their data if they want to benefit from SmartThings' services.

In this work, we aim to revisit the design of current smart home platforms so as to provide end-users with greater transparency and control over how their data is collected and used. While this seems to be disadvantageous for service providers, we argue that such is not the case for two main reasons. First, it would allow them to reach a large market of potential privacy-wary IoT consumers. In fact, a recent study [4] reported that 87% of US consumers "are concerned about their personal information being collected and used in ways they were unaware of"; 27% mentioned this concern as the "main reason they do not currently own a smart device". Unfortunately, such fears are all too well justified, backed up by anecdotal cases of stealthy data theft [5], [6], [7] or undisclosed data sharing [8], [9]. Second, service providers face increasing pressure from many countries to uphold strict personal data handling policies. Notably in Europe, since May 2018, the GDPR regulations [10] require service providers to pay formidable penalty fees in case of personal data misuse or user privacy violations. However, their common practice of aggressively collecting raw sensor data and shipping it down to their servers can only increase such risks.

This paper presents HomePad, a privacy-aware hub for home environments. Similarly to recently proposed systems [11], [12], [13], HomePad extends the architecture of current smart home platforms with the ability to execute IoT applications at the edge. This is achieved by relying on a trusted hub device that can both manage the local devices and provide a local platform for executing IoT apps without necessarily depending on the service provider's centralized services. As a result, whenever the functionality of an app does not strictly require the shipment of data onto the cloud, the sensor data can be collected and processed locally by the applications, therefore reducing the risks of data exposure and misuse at the service provider's backend.

To provide end-users with fine-grained control over the way these untrusted applications access and process sensor data, HomePad introduces two novel features. First, HomePad forces applications to make all information flows explicit. It employs a technique that allows to keep track

not only (1) of how the sensor data flows within the applications, but also (2) of the *semantics* of the data as it gets processed inside the application. This allows us, for example, to determine if an application that takes a photo from a camera sends that photo to the service provider in its raw form, or has instead sanitized the photo by running it through a privacy-preserving blurring algorithm. Our technique consists of exposing a programming model in which HomePad applications are implemented as *directed graph of elements*, which consist of interlinked instances of special functional units that can be put together in order to build applications.

A second noteworthy feature is that HomePad provides a mechanism that allows users to easily examine whether a given application has the ability to violate specific privacy concerns expressed in a user-defined policy. For example, the user can verify whether an application with access to a webcam has the ability to send raw image data to the cloud or not. This verification is performed at install time so that the user can refuse to install the application if it violates such conditions. To implement this feature, HomePad generates a formal model of the application's element graph in Prolog rules, and then issues a set of queries to determine the existence of data flows that violate the user's privacy policy.

We implemented HomePad and used it to test several use case applications. From our evaluation of the system, we found that HomePad was able to effectively detect illegitimate data flows and incures low performance overheads.

Next, we provide an overview of our motivation, approach and goals. In Section III, we introduce the HomePad hub and its API. We explain the concepts of dataflow programming model and the verification techniques in Sections IV and V. We then introduce the design concepts and implementation details of the HomePad hub in Sections VI and VII. We continue with the evaluation results in Section VIII. Finally, in Section IX we provide a security analysis of HomePad and describe its current limitations. We conclude with Section XI.

II. Goals and assumptions

The main goal of this work is to build a programmable hub system to allow for controlling smart home devices in a privacy-aware manner for homeowners. Following the current trends of mobile and web platforms, our hub system must provide an "appified" platform enabling third-party developers to write applications that can access the smart home devices (both for collecting sensor information or issuing control commands), process sensor data, or even issue requests to Internet services.

From a privacy perspective, our system must be able to make users aware of how their sensor data is accessed and processed by the apps, and eventually prevent the installation of apps that the homeowner may deem to be too privacy-invasive. For example, a homeowner may be comfortable

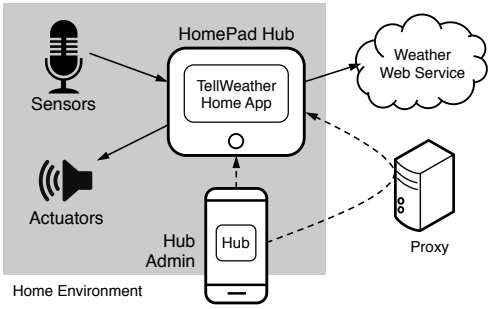

Figure 1. HomePad deployment.

with a notification to the cloud when motion is detected by her front door camera, but not with sending raw image data extracted from that camera. Our hub system must provide mechanisms to differentiate flows tolerable by the homeowner from those that are not.

Note also that we are not worried about preserving the compatibility with existing IoT platforms, such as Samsung SmartThings [1] or Apple HomeKit [2]. Nevertheless, we envision that these platforms can be easily integrated with our hub device by exposing REST APIs accessible to the hub. It is also not our goal to be fully compatible with existing IoT devices. Nevertheless, we assume that the IoT devices managed by HomePad have a public interface that allows for the communication between them and the hub.

Threat model: In designing our "appified" home hub, our main adversary consists of potentially buggy or malicious applications aiming to extract privacy-sensitive information from home sensors. An application may try to attain this goal by leveraging legitimate operations provided by the hub API. However, we assume that the hub platform itself is part of the trusted computing base. In particular, we do not focus on attacks which try to exploit bugs in the hub software or hardware, or attacks aimed at leveraging existing vulnerabilities in the smart devices themselves. We assume that the hub hardware is correct, that the software that implements the hub system is correct, and that potential software updates to the hub have been implemented and signed by trustworthy entities. We focus only on attacks that aim to exfiltrate sensitive data extracted from smart devices connected to the hub. Consequently, we do not prevent privacy breaches from rogue devices deployed at home that can connect to the Internet bypassing our hub. In this paper, we do not protect against low-bandwidth side-channels.

III. Overview

Figure 1 represents a HomePad deployment in a home environment. HomePad consists essentially of a smart hub that controls access to all smart devices at home and provides a platform for the execution of apps, called *home apps*. The HomePad hub provides an administration interface through which the homeowner can access the hub directly

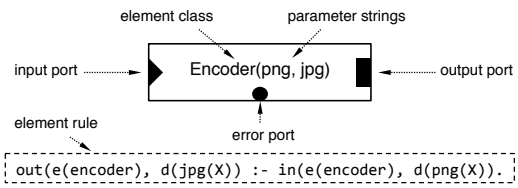

Figure 2. Element that converts a PNG image to JPG.

or tunneled through a proxy and manage it, e.g., to install or uninstall home apps, register new smart devices, supervise the execution state of home apps, install hub software extensions, or set up privacy policies.

Through the HomePad API, a home app can perform numerous operations, such as collecting data from sensor devices (e.g., audio from microphones, images from cameras), sending data to actuator sensors (e.g., audio signal to speakers, or video streams to displays), accessing Internet services, and performing various data computations (e.g., voice or face recognition, voice synthesis, or data anonymization). Figure 1 represents a simple home app—TellWeather—which listens for an audio command (e.g., "Tell weather in LA"), issues an HTTP request to a weather service, converts the response into audio signal, and forwards it to a speaker.

HomePad was designed from scratch with privacy-awareness in mind. There are two main features that contribute to attaining this goal. First, the HomePad API adopts a dataflow programming model that forces applications to make explicit both all internal data flows and all internal data transformations. As a result, HomePad contributes to making applications more transparent with respect to how they access and process user data. Second, HomePad includes a mechanism to automatically analyze how and what information flows within a home app and check the presence of invalid flows as specified in the privacy policy defined by the homeowner. This end is achieved by generating a Prolog model of the home app's information flow graph and performing relevant queries based on the privacy policy, which is also specified as simple Prolog rules. Next, we describe both these features in more detail in independent sections, and then present the HomePad design in Section VI.

IV. Dataflow programming model

This section presents how HomePad applications are written, and shows this programming model's expressiveness, allowing home apps to be built in a privacy-aware manner.

A. Elements

A HomePad application consists of a directed graph whose nodes are called *elements*. An element is a functional unit that can be executed on the hub. The graph edges represent a possible path for data transfer between the connected elements. An element has four important properties:

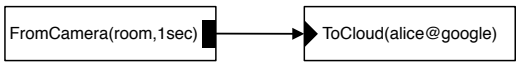

Figure 3. Sample motion detector app (version 1).

- **Element class**: Each element has an associated piece of code which determines its behavior. On the hub, each element is an instance of that code (e.g., of a Java class).
- **Ports**: An element can have any number of input or output ports, which can have different semantic meanings. A message at an input port causes the element code to execute; at the same time, a message may or may not be fired to the output port. Elements can optionally have an error port which is used by the hub to output internal exceptions and stack traces. Each port type is denoted with a different notation (see Figure 2) and is statically typed.
- **Parameter string**: Element classes may optionally support parameters to initialize per-element state and configure the element behavior.
- **Element rules**: An element must be accompanied by Prolog rules that specify the (abstract) types of data that can be sent as output.

Figure 2 represents a simple element, named Encoder, which converts images from PNG to JPG. The input port receives the PNG image and the output port sends the resulting JPG image. The file format is predefined in the parameter string. The output rule reflects this transformation as a Prolog rule. These rules are used in HomePad for privacy verification purposes, and are distributed along with the element class package. Also note that our element notation was inspired by the notation used for programming the Click modular router [14]. We adapted and extended Click's notation accordingly, by adding error ports and output rules.

B. Flow graph

Elements can be coupled together to form a directed graph, which we call *flow graph*. They can be connected by linking compatible input and output ports of matching data type. A flow graph makes information flow explicit across elements and can be used to fully describe a HomePad app.

Figure 3 represents the flow graph of a simple application aimed at detecting movement in Alice's bedroom. Element FromCamera takes photos from a stationary camera in Alice's bedroom, at the rate of one picture per second. Then it sends these pictures to element ToCloud, which internally issues HTTP requests to upload them to a cloud service hosted by Google. This cloud service runs a motion detection algorithm that analyzes differences between consecutive frames and fires a notification to the user if differences occur. (For the sake of simplicity, assume that the camera has no built-in motion detection capability.)

To execute this application inside a running hub, HomePad instantiates each element of the flow graph as a single

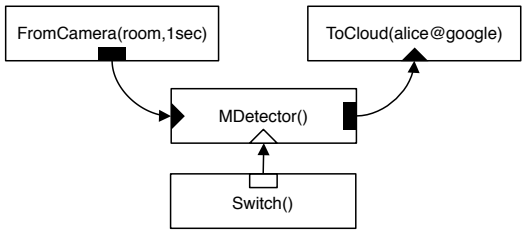

Figure 4. Sample motion detector app (version 2).

object and establishes an internal communication channel responsible for forwarding the messages between these elements according to the element connections as specified by the flow graph; no other data flows are allowed between elements other than those explicitly declared in the flow graph. Thus, from Figure 3 we see that camera frames are produced by FromCamera, which acts as a *data source*, and from there flow down to ToCloud, which acts as a *data sink*. To interact with the camera(s) and network interfaces, element objects use internal hub functions provided by HomePad drivers.

C. Push and pull connection types

In the type of element connection described so far, an element pushes data from its output port to the input port of the downstream element (see Figure 3 in which FromCamera sends incoming camera pictures to ToCloud). We say that elements are linked by a *push connection*. In other cases, elements must retrieve data synchronously from another element and process it before issuing an output. To address this need, elements can be connected by *pull connections*.

Figure 4 illustrates both connection types in the flow graph of a more complex version of the motion detection app presented previously. One major change is that instead of uploading all pictures to Google, the motion detection function is implemented locally at the hub by element MDetector. This element executes every time it receives a new frame from the input push connection, and keeps comparing incoming frames in order to detect any differences; if differences exist it outputs a "motion" event which is forwarded to element ToCloud, which uploads a notification to the cloud service.

This version also introduces element Switch, which allows the homeowner to turn the motion detection service on/off. Switch keeps track of an option ("on" or "off") selected by the hub administrator. MDetector must be able to retrieve the Switch state after receiving a new frame and decide on the output based on that result: if the state is "on", then MDetector sends "motion" events to cloud, otherwise no events are sent. To read the switch state synchronously, MDetector links the output port of Switch to its input port using a pull connection. To differentiate push from pull connections, we color them in black and white, respectively (see Figure 4).

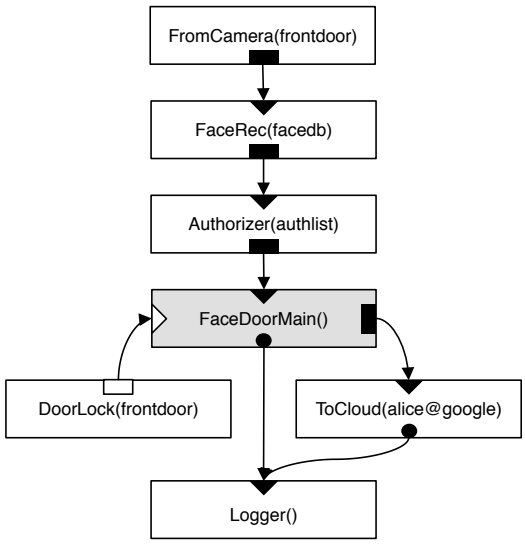

Figure 5. Flow graph of the FaceDoor app.

D. Untrusted elements

Each element implements a specific functional unit and can be considered to be part of the HomePad platform. HomePad supports the third-party development of new elements that can be incorporated into the hub as platform extensions (plugins). Since built-in elements are part of the trusted computing base, we call them *trusted elements*. Application developers can also write app-specific elements to be shipped along with the flow graph and instantiated on the hub. Because code of such elements cannot be deemed to be correct, we call them *untrusted elements*.

Figure 5 illustrates an example of an application that uses both trusted and untrusted elements, colored respectively in white and grey. This application is named FaceDoor and aims to automatically unlock the front door if the presence of a family member is detected through the camera installed at the main entrance. It starts by reading pictures from the camera (FromCamera), and sending them to a face recognition element (FaceRec). If the face recognition is successful, FaceRec sends the feature vector of the identified person to the Authorizer which checks if that person is a family member. If not, no output is generated. Otherwise, Authorizer sends a "match" event (without including the identity of the person), to the untrusted element provided by the app. This element— FaceDoorMain—sends an unlock command to the DoorLock element, waits for a response, and notifies the cloud service. If an error occurs in either FaceDoorMain or ToCloud, the app logs it using Logger.

The untrusted element FaceDoorMain is necessary to implement a piece of logic specific to the app. To enforce proper protection against buggy or malicious untrusted elements, HomePad instantiates them inside individual sandboxes such that they can communicate with the outer world only through the element's input and output ports.

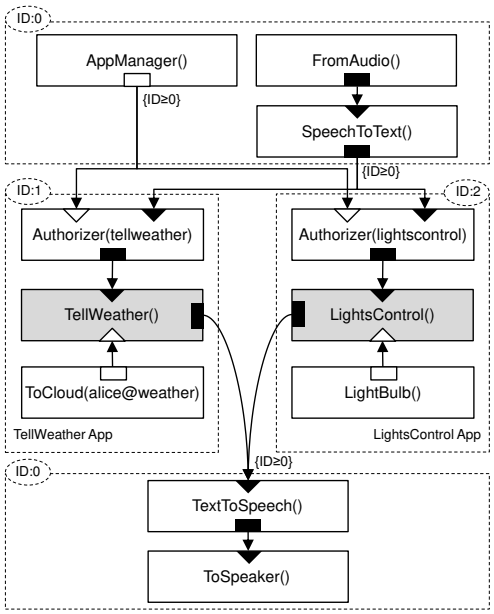

Figure 6. Hub configuration with multiple apps.

E. Hub configurations

The previous sections have discussed in detail how to implement a single application as a single flow graph. In order to host multiple applications, HomePad must not only allow multiple independent flow graphs to coexist, one for each application, but also allow applications to interact with each other and with global system services. Furthermore, it is necessary to ensure that applications cannot interfere with each other, e.g., by modifying each other's flow graphs. Moreover, HomePad must enforce strict compliance with the homeowner's privacy preferences when (un)installing apps.

To address these requirements, first, we extend the notion of flow graph to comprise not just a single application but the entire *hub configuration*. The hub configuration is represented by a fully connected flow graph that can be decomposed into two types of subgraphs: *system subgraphs* and *application subgraphs*. The former implement system-wide functions (e.g., event bus), and the latter represent installed applications. Second, installing (or removing) an application consists of patching the hub configuration so as to connect to it (or disconnect from it) the respective application subgraph. To ensure correct behavior, the connection of an application subgraph cannot be performed arbitrarily, but requires linking specific elements of both application subgraph and system subgraphs. Third, for security reasons, HomePad assigns principal IDs to subgraphs and defines *connection permissions* to restrict modifications to the structure of subgraphs (e.g., to prevent the installation of an application from tampering with the flow graph of another application). HomePad assigns the principal ID 0 to the system subgraph, and a new principal ID (>0) to each application subgraph. As a general protection rule, HomePad

```
1  import homepad.*
2  import homepad.elements.trusted.*
3
4  class FaceDoorMain extends UntrustedElement {
5      def initialize() {
6          // specify all ports of the FaceDoorMain element interface
7          port name:"auth", type:Authorizer.class, io:"in", connector:"push"
8          port name:"doorlock",type:DoorLock.class,io:"in",connector:"pull"
9          port name:"cloud", type:ToCloud.class, io:"out", connector:"push"
10         port name:"logger", type:Logger.class, io:"out", connector:"push"
11
12         // specify the handler function associated to the push input auth
13         inputhandler("auth") {
14             if (ref("doorlock").command("unlock"))
15                 ref("cloud").command("post", ref("auth").evt("id") +
16                     + "arrived!")
17             ref("logger").command("write", "Door unlock event.")
18         }
19     }
20 }
21
22 flowgraph {
23     // specify all seven element instances of the flowgraph
24     def fromcamera = new FromCamera("frontdoor")
25     def facerec = new FaceRec("facedb")
26     def auth = new Authorizer("authlist")
27     def doorlock = new DoorLock("frontdoor")
28     def facedoor = new FaceDoorMain()
29     def tocloud = new ToCloud("alice@google")
30     def logger = new Logger()
31
32     // specify the connectors of the flowgraph
33     from fromcamera to facerec
34     from facerec to auth
35     from auth to inport: "auth" facedoor
36     from doorlock to inport: "doorlock" facedoor
37     from outport: "cloud" facedoor to tocloud
38     from outport: "logger" facedoor to logger
39 }
```

Listing 1. Sample code of the FaceDoor app.

does not allow to interconnect elements of subgraphs with different IDs. However, connection permissions can override this rule.

To illustrate these concepts, consider the hub configuration depicted in Figure 6. It provides a common voice-activated user interface, through which the user can send instructions to applications installed on the hub (elements FromAudio and SpeechToText). Applications can provide audio feedback to the user by leveraging elements TextToSpeech and ToSpeaker. The hub relies on a global AppManager element to route the voice commands to each application. Figure 6 shows two applications installed: TellWeather, which issues a web service request to determine the current weather, and LightsControl, which can turn on/off the lights.

F. Programming model

HomePad provides a simple programming interface for writing applications. The programming interface is based upon a domain specific language (DSL) implemented in the Groovy programming language. To illustrate it, Listing 1 shows the pseudocode of the FaceDoor application represented by the flow graph depicted in Figure 5. Essentially, to implement this application, the developer must declare the element instances (lines 24-30) and element connectors (lines 33-38) of the application's flow graph. The trusted

elements are instantiated based on built-in classes or extensions to the HomePad API (e.g., FromCamera, ToCloud, etc.) which must be imported into the program. Untrusted elements must be written by the developer as independent classes, in this example the FaceDoorMain element is implemented by a homonym class. To specify an untrusted element, it is necessary to declare its ports (lines 7-10) and input port handlers (lines 13-18). From our experience, the effort of writing HomePad applications is comparable to the effort of writing applications for the popular SmartThings IoT framework, whose API is also based on a DSL developed in Groovy.

V. Verification of privacy properties

Section IV presented two simple versions of a motion detector app that implement a similar functionality, but offer different levels of privacy protection. In version 1, since the camera pictures are streamed to the cloud, the cloud provider has access to the raw picture data (see Figure 3). Version 2 performs most of the processing at the hub endpoint and simply notifies the cloud service about the occurrence of motion events (see Figure 4). As a result, version 2 is more privacy friendly for the user because less information is released to the cloud provider. The question is then how can users determine whether a given app satisfies their privacy requirements in a user-friendly manner.

To address this question, HomePad allows for the automatic verification of an application's privacy properties. In particular, it allows users to (1) determine what type of information is released by a given application, and (2) assess beforehand whether the type of information released by the application is acceptable to the user. The verification is performed by first creating a model of the application flow graph in Prolog (named *flow graph model*), and then issuing queries to determine the existence of illegitimate data flows. A data flow is illegitimate if it violates the conditions specified in a *privacy policy* provided by the homeowner. A privacy policy consists of one or more Prolog rules that specify disallowed flows of specific data types (e.g., a camera frame) to specific data sinks (e.g., the ToCloud element). Next, we explain how the flow graph model is generated, and in Section V-B we cover the main steps for the verification process.

A. Flow graph formal modeling

To create a flow graph model, HomePad only needs to analyze the flow graph of the application. From that analysis, it generates a set of facts and rules describing the elements the application depends on, their functions, connections, and the data types they operate with. These facts and rules are then written to a file, which will be provided to Prolog in the verification process. The generation of this file entails three steps:

```
1 el(fromcamera).
2 el(tocloud).
3 el(switch).
4 con(el(fromcamera), el(mdetector)).
5 con(el(mdetector), el(tocloud)).
6 con(el(switch), el(mdetector)).
```

Listing 2. Model of flow graph version 2.

1. Model the flow graph structure: HomePad begins to model the flow graph of an application by generating a set of facts in Prolog that represent the elements and the connections of the application's graph. The general format of these facts is represented by facts F_1 and F_2:

$$F_1 \rightarrow \quad \mathsf{el}(x).$$
$$F_2 \rightarrow \quad \mathsf{con}(\mathsf{el}(x), \mathsf{el}(y)).$$

Fact F_1 declares x to be an element of the graph, and fact F_2 declares a connection from element x to element y. When applied to version 1 of the motion detector application (see Figure 3), HomePad generates two F_1 facts ("el(fromcamera)." and "el(tocloud).") and one F_2 fact ("con(el(fromcamera), el(tocloud))."). On the other hand, version 2 is modeled by three elements and three connections as shown in Listing 2 (see Figure 4).

2. Model the behavior of trusted elements: The next step is to model how each element generates its outputs. Typically, an output is a function of the element's inputs and / or of the element's internal behavior. Since this function is dependent on the specific implementation of the element, to model element behavior, HomePad requires that each element is associated with its corresponding Prolog rules. These rules express how the element outputs are produced and the possible dependencies of these outputs from the element inputs. They are termed *element rules* and are provided to HomePad as part of the application package (see Section IV-A). When the application is installed, the HomePad hub keeps a repository of all the element rules declared in the system. When creating a flow graph model, HomePad retrieves the rules of the elements used by the application and includes these rules in the model file. Table I lists simple output rules for the elements used in the application examples presented in this paper. In general, element rules take the following form:

$$R_1 \rightarrow \quad \mathsf{out}(\mathsf{el}(x), \mathsf{data}(t_{out}(y))) \text{ :- } \mathsf{in}(\mathsf{el}(x), \mathsf{data}(t_{in}(y))).$$

This rule states that the output data of element x is defined as $t_{out}(y)$ and depends on the input data $t_{in}(y)$. Informally, R1 indicates that an element will produce a declared output as long as a given input is provided. The operator ":-" is used to define Prolog clauses. The more formal declarative interpretation of this Prolog clause is: "Given X and Y, an output typed $t_{out}(y)$ is produced from

Table I
SAMPLE RULES OF HOMEPAD ELEMENTS.

Element name	Element output rules
FaceRec	out(el(*facerec*), data(frfunc(X))) :- in(el(*facerec*), data(img(X))).
FromCamera	out(el(*fromcamera*), data(img(frame))).
MDetector	out(el(*mdetector*), data(mdfunc(X))) :- in(el(*mdetector*), data(img(X))), in(el(*mdetector*), data(state(on))).
Switch	out(el(*switch*), data(state(on))). out(el(*switch*), data(state(off))).

element *X*, if an input typed $t_{in}(y)$ reaches that element". Note, however, that each element may have a variant of this rule, or may even require more than a single rule. Consider now the examples in Table I. Regarding FromCamera and Switch, out rules indicate the type of data returned by the element: an image representing a frame from the camera (img(frame)), and the state of the switch element (state(on) and state(off)). In the switch element, since two outputs are possible, it is necessary to specify two rules, one for each output. Elements MDetector and FaceRec produce an output that is a function (mdfunc, frfunc) of their respective inputs (frame). The element ToCloud introduced in Figures 3 and 4 (and omitted in the table) does not require specific rules because it has no output ports.

3. Model the behavior of untrusted elements: Just like in the case of trusted elements, untrusted elements must also be accompanied by Prolog rules that characterize the type of data output by the element. However, the application programmer cannot be relied upon to write these rules. As a result, HomePad defines a common element rule for all untrusted elements. We take a conservative approach in modeling such elements by assuming that an untrusted element will try to forward all input data to the output ports in an attempt to leak as much data as possible. Thus, we can model an untrusted element using two rules:

$$R_2 \rightarrow \quad \text{bad(el}(x)\text{).}$$
$$R_3 \rightarrow \quad \text{out(el(X), data(Y)) :- bad(el(X)), in(el(X), data(Y)).}$$

Rule R_2 is used by HomePad do declare a specific element x as "bad". This operation is performed at the time when the application is installed on the hub. At this time, HomePad sweeps the flow graph, detects additional untrusted elements, and extends the flow graph model with R_2 in which x is replaced by the name of the element. This operation is performed for every untrusted element in the flow graph. HomePad then adds rule R_3 which, when read from right to left, says that if an element X is untrusted (denoted by "bad") then all its inputs might be forwarded to the outputs. In the FaceDoor app, HomePad would automatically mark FaceDoorMain as bad.

To prevent a malicious untrusted element from bypassing rules R_2 and R_3 by memorizing stale inputs in internal memory and forwarding them to the output in the future, HomePad forces each untrusted element to be *stateless*, i.e., each input is processed completely independently and the element cannot internally store state that can be passed over across different invocations of the element. If the app needs to store state persistently, it requires the incorporation of specific storage elements (see Section VI-B) in order to make all information flows appear explicitly on the flow graph.

4. Model the connection behavior: Now that the structure of the flow graph and the behavior of each element has been modeled, the last missing piece is to model the behavior of the graph's connections, which are responsible for propagating the outputs of upstream elements to the inputs of downstream elements. To model this behavior, HomePad adds rule R_4 to the flow graph model:

$$R_4 \rightarrow \quad \text{in(el(X), Y) :- con(el(Z), el(X)), out(el(Z), Y).}$$

Again, this rule should be read from right to left. It says that if an element Z outputs a data item Y and there exists a connection between Z and an element X, then X receives data Y as input. With R_4 the behavior of the flow graph is now completely specified. It is then possible to proceed with the automatic verification of the app's privacy properties as explained next.

B. Information flow tracking and privacy policies

The flow graph model automatically generated by HomePad allows the user to verify the privacy properties of the respective application. Essentially, our approach for privacy verification entails issuing specific Prolog queries to the flow graph model in order to track how information flows within the application. To this end, HomePad relies on a simple but powerful rule:

$$R_5 \rightarrow \quad \text{flows(X,Y) :- in(el(Y), data(X)).}$$

This rule allows to determine if a specific piece of data X will ever be able to flow within the application until it reaches element Y. To assess if this statement is true, R_5 checks if data X can eventually appear at the input of element Y. This rule can be used in HomePad for two main purposes: *application profiling* and *policy enforcement*.

Application profiling: This operation allows the user to analyze the flow graph of an application and learn how the information can flow within the application by determining, (1) what kind of information can be accessed by the application, (2) where information can be obtained from, and (3) where information can be propagated to. The flows rule can be instrumental for this purpose. For example, considering the motion application version 2 (see Figure 4), in order to

determine if the raw frame data from FromCamera reaches MDetector and ToCloud, we can issue two R_5 queries:

1. ?- flows(img(frame),mdetector).
2. **true.**
3. ?- flows(img(frame),tocloud).
4. **false.**

The results of these queries mean that the raw frame data can arrive at the motion detector element (line 2), but not at the cloud upload element (line 4). From here we see that this rule can help query whether a particularly sensitive piece of data arrives at a given element. But it can also be used for performing more powerful queries:

5. ?- bagof(X, flows(X,mdetector), L).
6. **L = [img(frame), state(on), state(off)].**
7. ?- bagof(X, flows(mdfunc(frame),X), L).
8. **L = [tocloud].**

In the first query (line 5), we aim to determine all data types that can arrive at a particular element, namely the motion detector element. A Prolog bagof() predicate is used for this query in order to obtain a list, or a "bag", of data types that satisfy a given condition. The result is shown in line 6, and consists of a list that comprises the raw frame data, and the state of the switch (on or off). By observing both Figure 4 and Table I, we can confirm that this is the expected result. The second query (line 7) queries for all elements that can have access to a particular data type, in this case which elements can access the motion event mdfunc(frame) returned by MDetector. The result is a list that includes a single element, ToCloud, as expected.

Policy assessment: While application profiling allows for generating "privacy reports" of applications, policy assessment aims to ensure that an application can be installed only if it satisfies the privacy restrictions as specified in a policy. A privacy policy is provided by the homeowner and consists of a list of flows facts which must hold true when checked against an application's flow graph. The HomePad hub runs this check at application installation time: if the test fails, installation aborts. Consider, e.g., the privacy policy P_1:

$$P_1 \rightarrow \quad \text{flows(img(frame),tocloud).}$$

This policy states that raw camera data (img(frame)) is not allowed to flow to the cloud (i.e., element ToCloud). As a result, if the homeowner attempts to install the version 1 of the motion detection application (see Figure 3), then P_1 evaluates to true, and the installation will be aborted. In contrast, version 2 of this application satisfies this requirement, and therefore can be safely installed on the hub. To facilitate the writing of privacy policies and profiling queries, HomePad provides hub administrators with a user-friendly interface to compose them out of predefined Prolog rules. Although more intuitive solutions can be applied to the specification of user privacy policies, we currently allow for users to specify such policies only through a mobile application.

Policy enforcement: Policy enforcement aims at addressing user privacy preference changes. More specifically, the cases when a user modifies his initial privacy preferences and by doing so enables a data flow which was not allowed at the time of the app's installation. In this case, HomePad automatically checks if all currently installed applications are compliant with the new privacy preferences. HomePad flags the apps failing this check and halts their execution temporarily. The user will then have to decide what to do: either modify the privacy preferences or uninstall the app.

Taking the first version of the motion detector app as an example, consider the case where Alice wishes to continue receiving motion notifications from 5pm to 7pm, before arriving from work and after her son arrives from school, but this time she wishes the raw video data to be blurred, so that no video of her son is sent to the cloud. In order to support such policies, the specification of privacy policy is extended from P_1 to P_2:

$$P_2 \rightarrow \quad \text{rtime(flows(img(frame),tocloud),}$$
$$\text{excep(mode(blur),time(17,19))).}$$

This enhanced policy, or runtime rule, joins flow rules with exception rules (rtime(flows(...), excep(...))). Exception rules complement flow rules with both timing constraints that restrict the period when a given data flow can occur, as well as the data mode, i.e., a transformation needed to be performed on the data before it is sent to the cloud. In this particular policy, video data would only be sent to the cloud if it was blurred (mode(blur)), and only during a specific time period (time(17,19)). This is achieved by modifying the data flow graph of the application and adding restricting trusted elements (FaceBlur and a TimeSwitch) to the data flow path (see Section VI-B for more details on these elements). HomePad applies such constraints at runtime each time the application is executed. Both flow and exception rules allow for a great flexibility in the specification of policies that express the current users' privacy concerns. In Section VIII-E we show how such an approach can be successfully used in common smart home scenarios.

VI. Design

This section presents architectural details of the HomePad hub, which allows for the execution of home applications and verification of their privacy properties.

A. Architecture

In HomePad, there are several involved parties. *Users* interact with home apps via apps' own interfaces. The *hub administrator* (typically the homeowner) maintains the hub, e.g., by installing or removing apps and elements, setting up privacy policies, performing privacy verification operations.

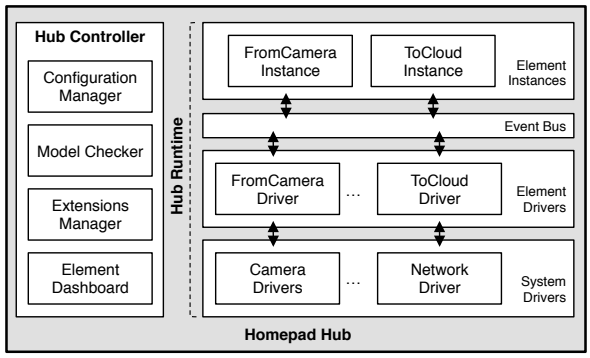

Figure 7. HomePad hub architecture.

Application developers create HomePad applications, which involves writing a manifest file specifying the flow graph and the code of untrusted elements, compiling, and packaging the code binaries and manifest. *Element developers* implement new elements. To this end, they must write the element code and the output Prolog rule. *Platform developers* write and maintain the code of the HomePad core system installed in the hub. We envision HomePad core and elements to be developed by an open source community.

Figure 7 represents the main software components of the HomePad system running on the hub. The model checker manages a repository of Prolog rules and exposes a management interface to allow for privacy verification of applications. This repository contains: the output rules of elements installed in the system, the models of all applications installed in the system, and the privacy rules defined by the hub administrator. Before installing an application, the administrator can upload the flow graph of the application to the system, and check if it is compliant with the privacy policy. If not, the installation aborts, otherwise, the application package is deployed into the system.

The configuration manager maintains the installed applications, offers a management interface to the hub administrator, and supervises the execution lifecycle of applications. When a home application is installed on the hub, the configuration manager instantiates element objects on the kernel runtime and sets up connections between instances so as to reflect the flow graph specified in the home application package. Each element object can interact with a local driver which serves the specific requests of that particular element. As for untrusted elements, the runtime kernel runs the respective app code inside individual sandboxes. The sandboxing mechanism prevents the use of shared memory and thus leaking information across elements.

Elements and drivers together implement the app functionality by firing events and routing them internally through the event bus. Figure 7 illustrates how this works for the simple motion detector application represented in Figure 3. The driver of the FromCamera element is configured to read a new frame (one frame per second) from any camera in Alice's bedroom, and forwards that frame to the respective element instance that belongs to the application. This event is eventually forwarded to a driver responsible for the cloud invocation operation. The HomePad hub architecture is general, allowing for future extensions with new element / drivers. Next, we describe several trusted element categories that can be used as app building blocks.

B. Functionality of trusted elements

HomePad includes a library of trusted elements which allows for highly flexible hub configurations. Trusted elements can be further incorporated into this library in order to provide new functions to application developers. We present several functions, grouped into broader categories:

Interaction with sensors and actuators: A crucial functionality is to enable home apps to access sensors (e.g., thermostats, cameras, and microphones) or actuators (e.g., locks, or light bulbs). While some elements can implement low-level functions such as simply reading / writing from / to a device, others can be more sophisticated and high-level. For example, element FromCamera can read data from multiple devices, know camera location(s), and can take pictures at predefined frame rates.

Communication with remote endpoints: Trusted elements can also enable communication with a remote party. The ToCloud element, e.g., allows an application to issue HTTP requests to a web service. Specific trusted elements can be implemented to communicate with a mobile application. To authenticate the mobile application's endpoint, the trusted element can be configured with a public key, whose private part is maintained by the mobile application only.

System control and management: Some trusted elements can be devised specifically to help control and manage the system. For example, Switch (see Figure 4) provides an interface for the administrator to enable or disable a given function. This element implements a widget which is integrated into the control dashboard in the HomePad hub. Logger provides a logging functionality enabling the application to maintain a record of operations. Additional trusted elements can be used to control the system, e.g., the AppManager element (see Figure 6).

Error handling and debugging: Specific trusted elements can help handle application errors and debugging, e.g., for sending bug reports to an application provider. To preserve anonymity, there can be instances of such elements that, in addition to packaging memory dumps or exception related data, can first anonymize that data so as to prevent exfiltration of sensitive user information.

Storage of persistent data: As mentioned in Section V-A, untrusted elements where application code is executed are stateless. However, the application may need to keep state persistently, either for sharing context information between

untrusted elements, exchanging data between multiple applications, etc. To this end, HomePad includes trusted elements for persistent data storage. Such elements can be instantiated with a given capacity. The following example shows the output rule of a key-value store element:

```
out(el(kvstore), data(get, K), data(get, V)) :-
    in(el(kvstore), data(put, pair(K, V))).
```

This rule states that the output value of a get request for key K will return value V, which corresponds to the key-value pair provided when issuing the request get(K,V). More sophisticated storage elements can be devised, e.g., to keep anonymized user data, implementing restricted data access control through differential privacy, etc.

Data transformation: A class of trusted elements aims at implementing data transformation functions. Examples include audio / video codecs, compression algorithms, encryption algorithms, etc. The face recognition element FaceRec presented in Section IV-D is one concrete example. In general, the output rule of such elements can be written as:

$$\text{out(el(dtf), data}(f(\text{X}))) \text{ :- in(el(dtf), data(X)).}$$

This rule states that the output of a data transformation element (dtf) is the result of applying function f to the input data X. Naturally, the function to be applied is element-specific. It is also possible to implement data sanitization functions, e.g., by anonymizing user identifiers, filtering sensitive data, replacing certain data with mockup information.

Time-based dataflow control: Finally, HomePad provides a set of trusted elements that allow to enforce time constraints on the application's data flow. For instance, with the TimeSwitch element, users are able to specify time windows when data flows are allowed or denied. As an example, consider an IP camera which is allowed to record the video when the user is not at home, and denied to do so otherwise. Alternatively, a RateLimit element provides a way to specify the maximum rate for data transmissions. Following the IP camera example, the video stream might be restricted to one frame per second, when an input of '1 sec' is provided to RateLimit element. This is important when using expensive mobile network connections or battery-powered devices. Overall, this type of elements allow for the enforcement of various time-constraining rules by just modifying the application graph accordingly.

C. Correctness of trusted elements

HomePad relies on multiple elements provided as part of the platform. The behavior of these elements and the operations they perform on sensor data are described by their accompanying Prolog rules. However, there can be a mismatch between the elements' internal implementation and their corresponding Prolog rules. Such a mismatch must be avoided otherwise the correctness and safety of trusted elements could be compromised. To overcome this problem we propose an approach inspired by the Linux OS success. We envision trusted elements' software to be maintained and scrutinized by an open-source developer community. Several techniques can then be used in order to provide assurances with regards to the correctness of trusted elements code, such as software testing and verification tools.

VII. Implementation

We implemented HomePad to run on top of Debian 8 OS on a dedicated computer. We implemented home apps' untrusted element sandboxes using Java Security Managers to restrict access to network and underlying file system. As for the System Drivers, e.g., Camera Driver, we used custom Python scripts to interface low level communication between devices and HomePad's system drivers wrapping Java classes. Sensor data is received by system drivers and forwarded to element drivers, which then serve it to apps' element instances through the event bus (see Figure 7). This dataflow is event-based and is fully implemented in Java.

To simulate device communication, we used Arduino Yun boards [15] and implemented simple device drivers in C++ and Python, to interact with the HomePad hub. These boards communicate over Wi-Fi through AES-256 secure channels. To support application scenarios with different sensors, e.g., cameras, microphones, we established simple APIs to facilitate the management of these boards via the HomePad hub. For the same reason, the boards were equipped with a Sony USB webcam [16] and electret microphones [17].

At install time the Model Checker analyzes the application's DSL code in order to validate the application's privacy properties. This validation involves the generation of the application's corresponding Prolog model followed by a set of Prolog queries. The Model Checker component was implemented as a Java class with SWI-Prolog version 6.6.6 engine stubs. To provide the user with a visual representation of the applications' structure and privacy properties, we implemented an HTML report generator using the Graphviz tool. This report shows the results of dataflow analysis from the Prolog queries. In order for users to specify their own privacy policies we developed a simple Android application offering a simple API that allows users to pick data sources and sinks, as well as exception rules such as time constraints or data modes (e.g., encrypted, anonymized). The app then sends the Prolog rules to Homepad through HTTPS.

VIII. Evaluation

We present an evaluation of HomePad regarding performance, application programming effort, and verification effectiveness. We then evaluate HomePad's privacy policy specification mechanism and its ability to model and express the variety of users' privacy concerns.

Table II
USE CASE EXECUTION TIMES.

Execution	Lights Controller			Spotify Controller			Tide Pooler			FaceDoor		
	Baseline	HomePad	Over	Baseline	HomePad	Over	Baseline	HomePad	Over	Baseline	HomePad	Over
Recognition	2.2s (99%)	2.3s (98%)	5.8%	2.3s (99%)	2.4s (99%)	5.6%	2.4s (63%)	2.5s (61%)	4.1%	1.07 (89%)	1.11s (87%)	4.7%
Actuators	34ms (1%)	37ms (<2%)	7.7%	0.7ms (1%)	1.1ms (<1%)	63%	0.7ms (1%)	0.9ms (<1%)	32%	30ms (2%)	35ms (3%)	15%
Network	–	–	–	–	–	–	1.4s (36%)*	1.5s (38%)*	2.6%	117ms (9%)	119ms (9%)	1.8%
Core	–	3.6ms (<1%)	–	–	3.5ms (<1%)	–	–	8.8ms (<1%)	–	–	5.7ms (<1%)	–
Total	2.2s (100%)	2.3s (100%)	6%	2.3s (100%)	2.5s (100%)	5.7%	3.8s (100%)	4.1s (100%)	5.4%	1.2s (100%)	1.3s (100%)	4.7%

Legend: Baseline = execution outside HomePad; HomePad = execution inside HomePad; Over = HomePad execution overhead; * = both Network times account for 63.7ms and 63.9ms to parse the results of the tide request outside and inside HomePad respectively.

A. Use-case applications

To demonstrate the richness of applications supported by HomePad, we developed four applications using technologies and devices available today in the smart home environment. Some applications rely on open-source software, i.e., Kaldi ASR [18] for voice recognition and OpenFace [19] for face recognition.

1) Lights Control application - voice based lights control. Implemented using Philips Hue API [20].
2) FaceDoor application - face recognition based door control. Implemented by custom device drivers.
3) Tide Pooler application - voice based tide information request service that performs text-to-speech conversion when informing the user. This app was ported from the Amazon Echo [21] skills collection.
4) Spotify Control application - voice based Spotify player control. Implemented by custom device drivers and leveraging Spotify's API.

The privacy risks associated with these applications come from the way they interact with the user. Voice and face recognition requires access to the camera or microphone feed which is a source of sensitive information constantly being analyzed and may be used without the user's knowledge.

B. Performance evaluation

To evaluate the performance of HomePad, we adapted the four home automation applications described above to run under two different configurations: on HomePad and as standalone Java applications. This setup allows us to compare the performance overhead introduced by HomePad. To test the execution of these apps and measure their performance, we specified voice commands and pictures as inputs, according to each use case. The values presented reflect the average of 40 tests per application, with 20 running inside and the other 20 running outside HomePad.

Figure 8 plots the execution time of our use-case applications when executed on HomePad (light grey) and on standalone mode (dark grey). HomePad introduces an overhead which varies between 4.7% and 6%. This overhead is caused by the containerized sandboxes implemented by HomePad. From our experience, considering that the total

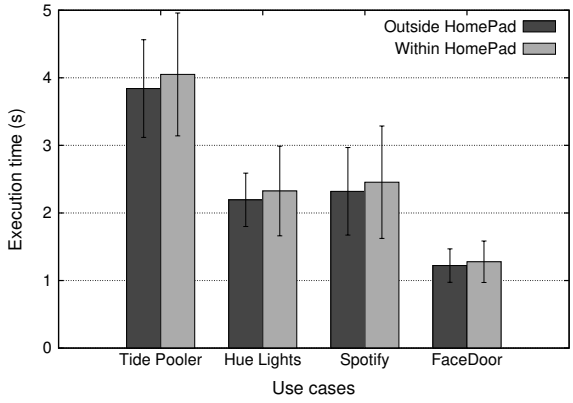

Figure 8. Use case performace.

execution time varies between 2.2 and 4.1 seconds, these overheads do not significantly hinder the user's experience.

To better understand the factors that contribute to the overall performance of each application, Table II displays the total application execution time broken down into: *recognition*, *actuators*, *network*, and *core*. Recognition time is associated with the execution of voice or face recognition and it measures the overhead of running these algorithms following our privacy preserving containerized approach. Actuators comprise the time spent on commands to turn on lights, play the next track on Spotify, output tide information as sound and unlocking a door. Network time involves the communication with the outer world, whether to fetch tide information, or notify a user someone just entered his home. Core refers to the time spent on the event based communication characteristic of HomePad's architecture.

Most of the execution time of these apps is spent on voice and face recognition (between 61% and 99%), which constitute the most CPU-intensive tasks. On the other hand, the time spent on actuators represents a very small percentage, never bigger than 3% or longer than 40ms. The network communication accounts for 9% of FaceDoor's total execution time, as it only features an API call to an online notification service so the user can know when someone enters his home. In Tide Pooler, networking cost amounts to 38% taking on average 1.5 seconds (due to the download

Table III
PRIVACY POLICY SPECIFICATION AND TRANSLATION.

Application Name	Description	Privacy Concern	Privacy Policy
Amazon Echo	Interactive voice assistant app that records and responds to user commands prepended with an "Alexa" wake word.	*Can the device record the conversations even when the wake word was not said?*	data source(Amazon Echo), transformation(wake word detection), data sink(Amazon cloud).
Nest Cam	Video surveillance and motion detection app with cloud backup.	*Is the camera active when the owner is at home?*	data source(Nest Cam), time restriction(8 PM - 8 AM), data sink (Nest Cloud)
Hello Barbie	Interactive doll app that records and responds to children's questions.	*What can a toy say to the child?*	data source(Hello Barbie App), transformation(word filter), data sink(Hello Barbie Doll).

and parsing of a large file containing tide information). Processing and routing internal messages within the HomePad core takes up only about 1% of the total execution time. The overall performance is largely influenced by specific app elements.

C. Application programming effort

To assess the programming effort needed to implement a HomePad app we ported our most complex use case application – Tide Pooler – for two additional platforms: Amazon Skill [22] and Google Speech API [23]. Amazon Skill leverages Amazon's backend to perform voice recognition and provides compatibility with Amazon's Echo device [21]. Google Speech API provides a voice recognition service that our Tide Pooler port uses when running as a Java desktop application. Keep in mind that our baseline HomePad implementation of Tide Pooler uses HomePad's native voice recognition system module based on Kaldi [18]. All Tide Pooler versions were implemented in Java. We assess the development effort in terms of the number of lines of code (LOC).

From our experience, we found that the development effort of implementing Tide Pooler across these platforms is quite comparable, requiring 331 LOC for Amazon Skill, 332 LOC for Google API, and 370 LOC for HomePad. In all cases, 35 LOC relate directly to the use case logic, 15 LOC relate to getting tide information from a server, and 250 LOC correspond to parsing the json file returned from the server. The remaining lines of code are specific to the API of each platform. In HomePad, specifically, 70 LOC are associated with adaptation to HomePad's module-element architecture.

D. Detection of privacy violations

To evaluate whether HomePad is able to detect policy violations by a malicious application, we crafted two of our use case apps, namely TidePooler and FaceDoor, by adding malicious untrusted elements into their data flow graph. These elements, named *Bad, aim to collect raw sensor data from camera or microphone and send it directly to the cloud without the user's knowledge. To implement this in TidePooler, we added a new element TidePoolerBad to the flow graph and connected it to FromMicrophone and CloudCall. Similarly, for FaceDoor, we introduced FaceDoorBad and added one upstream connection to FromCamera and one downstream connection to CloudCall. To perform the test, we also specified this privacy policy:

```
flows(img(frame),tocloud).
flows(mic(sample),tocloud).
```

This policy declares to be invalid any flow of raw audio or video data going to the cloud. We then executed the checker for each app. In both cases, HomePad has detected privacy violation and correctly identified the malicious element.

E. Flexibility of privacy policies

HomePad allows for flexible privacy policies. To demonstrate this flexibility, Table III presents three real-life use-case scenarios [24], [4] that can benefit from HomePad's rich privacy policy features. The first example covers major concerns regarding always-on voice assistants. Users worry that devices like Amazon Echo can silently record and analyze their conversations [25], [26], [27], [28], [29]. Such concerns can be expressed in a HomePad privacy policy with an exception rule requiring the wake word detection before delivering the audio recording to remote service providers.

The second example illustrates a common concern regarding Internet-connected home cameras. Users are essentially worried that their cameras are active when they are not supposed to thus violating user privacy [7], [30], [31], [32]. In this particular case, the user wishes for his bedroom's camera to be inactive from 8 PM to 8 AM. Within HomePad this restriction can be validated using an exception rule that specifies the time during which the restriction applies.

The last example regards common concerns over smart interactive toys. Parents worry that such toys might leave children vulnerable to stealthy advertising or offensive content [6], [33], [34]. In HomePad a swear words privacy policy can be modelled with an exception rule that performs word filtering on the data the application wishes to send to a toy's speaker. Note that this case shows that HomePad's privacy policy specification can not only handle outgoing data flows but also incoming data flows that may violate users' privacy.

IX. DISCUSSION

In this section we provide a brief security analysis of HomePad as well as its current limitations.

A. Security discussion

If we assume that the hub system and installed element software is correct, an attacker (i.e., a malicious application developer) may try to deploy malicious untrusted element code undeclared in the application manifest in an attempt to execute it on the hub. This attack, however, is prevented by HomePad, which only allows the execution of elements explicitly declared in the manifest.

An attacker may attempt to craft the flow graph in the application manifest, e.g., adding concealed connections between elements in order to bypass sensitive data to a data sink, or adding a large number of connections and elements in order to increase the complexity of the graph and obfuscate the flow of data. Such attacks can also be thwarted by HomePad, because it fashions a complete model of the flow graph which captures all elements and connections which can, therefore, be detected by the Prolog checker.

A malicious home app may try to exfiltrate information through implicit flows, e.g., by omitting or issuing a call to the ToCloud element (even if sensitive data is sent in the request). However, in HomePad's flow graphs, all information flows are made explicit, which means that by the time the user validates the policy, he is informed that such flows are possible and can decide consciously as to whether or not to proceed with the installation of the application.

A limitation of HomePad is that the privacy verification depends on the correctness of both the output rules of elements and the rules of privacy policies. If errors exist in rules, the flow graph will no longer reflect the app's implementation logic which may result in undetected breaches. This problem is alleviated by the fact that the Prolog rules of elements are very simple and relatively easy to analyze.

An additional limitation comes from the conservative approach used for data flow verification of untrusted elements. By default, HomePad assumes the output of untrusted elements to be the same as their inputs. Such a strict approach was selected in order to safeguard the user's privacy, even if it means to incur some false positives. Nevertheless, it is possible to refine the verification granularity, for instance, by using dynamic taint-tracking within untrusted elements to verify the input/output data types.

B. Operational considerations

A potential concern is that it might be complicated to manage HomePad hub for people with no computing background, especially to create the privacy policies. Moreover, the privacy policies can also grow in complexity depending on the number of installed apps. Creating and managing complex policies may cause the users to experience *decision fatigue*, a state in which a user gets overwhelmed by options and acts recklessly. To maintain the privacy policies more manageable, HomePad includes pre-defined rules that can be used as is according to the profile of the user and the smart home devices he or she owns. These built-in rules contain best-practice privacy policies as recommended by industry experts or other tech-savvy HomePad users.

Another concern is related to HomePad's backward compatibility with existing smart home systems. However, we argue that the market pressure for enhanced privacy and data protection may well justify a departure from existing IoT models in favor of alternative secure-by-design IoT platforms, such as HomePad. Nevertheless, we plan to investigate in the future whether it would be possible to automatically (or semi-automatically) extract the dataflow model from existing platforms applications, e.g. Samsung SmartThings, so as to enable developers and smart home owners to reuse existing applications on HomePad.

X. Related work

There is a large body of work addressing home automation and IoT-related issues, such as the privacy of sensor-generated data. Centralized approaches have been proposed to address user personal data storage access and management [35], [36], [37]. However, these contributions do not consider the issue of how apps use sensitive data once in their possession. At the same time, Privacy Capsules [38] processes raw sensor data only inside sealed containers without network access. While Privacy Capsules limit access to the network, we allow the user to decide if an app may access data and network resources dynamically.

Some recent works address these privacy issues from a network perspective. Davies et al. [13] propose the deployment of cloudlets to run applications and manage their access to raw sensor data. Yu et al. [12] suggested using routers to secure IoT devices by running micro network-security functions, acting as security gateways for each device. However, in both cases it is assumed the apps and functions are trusted respectively.

Fernandes et al. [39], [40] as well as Tian et al. [41] identified and addressed the problems of over-privileged apps in a popular smart home platform. However, all of these systems focus mainly on security implications of over-privileged apps and assume access to their source code. In ProvThings [42], Wang et al. perform IoT platform log analysis to detect malicious device actions. They, however, assume the smart home cloud platform execution environment to be trusted, which is at odds to HomePad assumptions.

A decentralized trigger-action smart home platform DTAP was proposed in [43]. While DTAP renders compromised OAuth tokens useless, it does not allow to track and control the flow of user data to legit token holders. In contrast, HomePad allows to do so for any third party involved.

There are several systems that perform information flow analysis through taint tracking [44], [45], [46], but also leveraging static code analysis [47], [48], [49]. These systems, however, do not address the smart home environment and its complex interaction model.

FlowFence [11] uses information flow control to manage sensor data accesses from applications. One of FlowFence's limitations is the inflexibility of its taint labeling mechanism. For instance, consider again the motion detector example where Alice wishes to send blurred video data of her bedroom to the cloud for motion detection. While Homepad's privacy policy specification and system trusted elements allow for the raw video data to be processed before being sent to the cloud, FlowFence's mechanism overtaints this data, preventing even for the blurred data to be sent. On the other hand, FlowFence offers no way to automatically verify the privacy properties of an application against users' preferences, resorting instead to a pure runtime mechanism, incurring in considerable performance overhead.

There's also a considerable amount of work in the field of formal verification [50], [51], [52], [53]. In particular, Deshotels et al. [54] use Prolog to model the policies of iOS container sandbox profiles and discover vulnerabilities in them. Still, this solution does not directly address our problem as that although it uses Prolog, it has a broader focus on assessing security rather than privacy properties.

Various software verification techniques have been proposed and used for quite some time [55], [56], [57], [58], [59]. State-based model checking methods, for instance, allow to verify the safety properties of a given software by checking all the possible states it can reach. Although these methods can provide high precision, their main shortcoming is a so-called *state-space explosion* – an exponential growth of system states which often makes a model checking ineffective. In HomePad we leverage some model checking ideas but operate with data operations instead of application states in order to fully model any IoT app. This allows to improve the verification performance dramatically and overall makes the model checking approach more practical. Furthermore, while classic model checking techniques are more suitable for control-flow analysis, HomePad's approach allows to perform sophisticated data-flow analysis, which is essential for user privacy guarantees.

XI. Conclusions

We presented HomePad a privacy-aware home hub that allows users to supervise how the data generated by smart devices is processed and used by home applications. In HomePad, applications are required to be modularized, and the data flows between those modules made explicit by the developers. By laying out applications in this fashion, HomePad can automatically leverage its Prolog-based data flow verification mechanism in order to assess these applications' compliance with users' privacy policies. Additionally, Homepad's expressive privacy policy specification supports a broad spectrum of privacy concerns users have. By combining these two capabilities, Homepad provides runtime data control to users.

Acknowledgments

We thank our shepherd Ardalan Amiri Sani and the anonymous reviewers for their comments and suggestions. This work was partially supported by national funds through Instituto Superior Técnico, Universidade de Lisboa, and Fundação para a Ciência e Tecnologia (FCT) via projects UID/CEC/50021/2013 and SFRH/BSAB/135236/2017, and by the LightKone project in the European Union Horizon 2020 Framework Program under grant agreement 732505.

References

[1] "Samsung SmartThings," https://www.smartthings.com. Accessed September 2018.

[2] "Apple HomeKit," https://developer.apple.com/homekit/. Accessed September 2018.

[3] "SmartThings Terms of Use," https://www.smartthings.com/terms. Accessed September 2018.

[4] TRUSTe, "US IoT Privacy Infographics," https://www.truste.com/resources/privacy-research/us-internet-of-things-index-2015/. Accessed September 2018.

[5] Forbes, "When 'Smart Homes' Get Hacked: I Haunted A Complete Stranger's House Via The Internet," http://www.forbes.com/sites/kashmirhill/2013/07/26/smart-homes-hack. Accessed September 2018.

[6] S. Gibbs, "Hackers can hijack Wi-Fi Hello Barbie to spy on your children," https://www.theguardian.com/technology/2015/nov/26/hackers-can-hijack-wi-fi-hello-barbie-to-spy-on-your-children. Accessed September 2018.

[7] D. Kerr, "FTC and TrendNet settle claim over hacked security cameras," https://www.cnet.com/news/ftc-and-trendnet-settle-claim-over-hacked-security-cameras/. Accessed September 2018.

[8] L. Fair, "What Vizio was doing behind the TV screen," https://www.ftc.gov/news-events/blogs/business-blog/2017/02/what-vizio-was-doing-behind-tv-screen. Accessed September 2018.

[9] C. Matyszczyk, "Samsung's warning: Our Smart TVs record your living room chatter," https://www.cnet.com/news/samsungs-warning-our-smart-tvs-record-your-living-room-chatter/. Accessed September 2018.

[10] "EU General Data Protection Regulation (GDPR)," https://www.eugdpr.org/. Accessed September 2018.

[11] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical Data Protection for Emerging IoT Application Frameworks," in *Proc. of USENIX Security*, 2016.

[12] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things," in *Proc. of HotNets*, 2015.

[13] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy Mediators: Helping IoT Cross the Chasm," in *Proc. of HotMobile*, 2016.

[14] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *Proc. of TOCS*, 2000.

[15] "Arduino Yun," https://www.arduino.cc/en/Main/ArduinoBoardYun. Accessed September 2018.

[16] "Sony PlayStation Eye," https://www.engadget.com/products/sony/playstation/eye/specs/. Accessed September 2018.

[17] "Low power, signle-supply, rail-to-rail operational amplifiers," http://www.ti.com/lit/ds/symlink/opa344.pdf. Accessed September 2018.

[18] "Kaldi ASR," http://www.kaldi-asr.org/. Accessed September 2018.

[19] "OpenFace," https://cmusatyalab.github.io/openface/. Accessed September 2018.

[20] Philips, "Hue Developer Program," https://developers.meethue.com. Accessed September 2018.

[21] "Amazon Echo," https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E. Accessed September 2018.

[22] "Alexa Skills Kit," https://developer.amazon.com/alexa-skills-kit. Accessed September 2018.

[23] "Speech API - Speech Recognition," https://cloud.google.com/speech/. Accessed September 2018.

[24] "Privacy and Information Sharing," http://www.pewinternet.org/2016/01/14/privacy-and-information-sharing/. Accessed September 2018.

[25] "How Amazon Echo Users Can Control Privacy," https://www.forbes.com/sites/tonybradley/2017/01/05/alexa-is-listening-but-amazon-values-privacy-and-gives-you-control. Accessed September 2018.

[26] "Goodbye privacy, hello 'Alexa': Amazon Echo, the home robot who hears it all," https://www.theguardian.com/technology/2015/nov/21/amazon-echo-alexa-home-robot-privacy-cloud. Accessed September 2018.

[27] "How Alexa, Siri, and Google Assistant Will Make Money Off You," https://www.technologyreview.com/s/601583/how-alexa-siri-and-google-assistant-will-make-money-off-you/. Accessed September 2018.

[28] "The FBI Can Neither Confirm Nor Deny Wiretapping Your Amazon Echo," http://paleofuture.gizmodo.com/the-fbi-can-neither-confirm-nor-deny-wiretapping-your-a-1776092971. Accessed September 2018.

[29] "How private is Amazon Echo?" https://www.slashgear.com/how-private-is-amazon-echo-07354486/. Accessed September 2018.

[30] "A Creepy Website Is Streaming From 73,000 Private Security Cameras," https://gizmodo.com/a-creepy-website-is-streaming-from-73-000-private-secur-1655653510. Accessed September 2018.

[31] "Webcam Maker Takes FTC's Heat for Internet-of-Things Security Failure," http://www.technewsworld.com/story/78891.html. Accessed September 2018.

[32] "Hacks to turn your wireless IP surveillance cameras against you," http://www.networkworld.com/article/2224469/microsoft-subnet/hacks-to-turn-your-wireless-ip-surveillance-cameras-against-you.html. Accessed September 2018.

[33] "German parents told to destroy Cayla dolls over hacking fears," http://www.bbc.com/news/world-europe-39002142. Accessed September 2018.

[34] "What did she say?! Talking doll Cayla is hacked," http://www.bbc.com/news/av/technology-31059893/what-did-she-say-talking-doll-cayla-is-hacked. Accessed September 2018.

[35] A. Chaudhry, J. Crowcroft, H. Howard, A. Madhavapeddy, R. Mortier, H. Haddadi, and D. McAuley, "Personal data: thinking inside the box," in *Proc. of Aarhus*, 2015.

[36] D. McAuley, R. Mortier, and J. Goulding, "The dataware manifesto," in *Proc. of COMSNETS*, 2011.

[37] N. Anciaux, L. Bouganim, B. Nquyen, I. S. Popa, P. Pucheral, and P. Bonnet, "Trusted cells: A sea change for personal data services," in *Proc. of CIDR*, 2013.

[38] R. Herbster, S. DellaTorre, P. Druschel, and B. Bhattacharjee, "Privacy capsules: Preventing information leaks by mobile apps," in *Proc. of MobiSys*, 2016.

[39] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash, "ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms," in *Proc. of NDSS*, 2017.

[40] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications," in *Proc. of the 37th SSP*, 2016.

[41] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Z. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 361–378.

[42] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Network and Distributed Systems Symposium*, 2018.

[43] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized action integrity for trigger-action iot platforms," in *Network and Distributed Systems Symposium*, 2018.

[44] J. Bell and G. Kaiser, "Phosphor: Illuminating Dynamic Data Flow in Commodity JVMs," in *Proc. of OOPSLA*, 2014.

[45] O. Ruwase, P. B. Gibbons, T. C. Mowry, V. Ramachandran, S. Chen, M. Kozuch, and M. Ryan, "Parallelizing dynamic information flow tracking," in *Proc. of SPAA*, 2008.

[46] Y. Xu and E. Witchel, "Maxoid: Transparently Confining Mobile Applications With Custom Views of State," in *Proc. of EuroSys*, 2015.

[47] A. C. Myers, "JFlow: Practical Mostly-Static Information Flow Control," in *Proc. of POPL*, 1999.

[48] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps," in *Proc. of PLDI*, 2014.

[49] F. Wei, S. Roy, X. Ou *et al.*, "Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps," in *Proc. of CCS*, 2014.

[50] M. J. May, C. A. Gunter, and I. Lee, "Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies," in *Proc. of CSFW*, 2006.

[51] M. Brusó, K. Chatzikokolakis, and J. Den Hartog, "Formal Verification of Privacy for RFID Systems," in *Proc. of CSF*, 2010.

[52] M. Kost, J.-C. Freytag, F. Kargl, and A. Kung, "Privacy Verification using Ontologies," in *Proc. of ARES*, 2011.

[53] M. Arapinis, L. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar, "New Privacy Issues in Mobile Telephony: Fix and Verification," in *Proc. of CCS*, 2012.

[54] L. Deshotels, R. Deaconescu, M. Chiroiu, L. Davi, W. Enck, and A.-R. Sadeghi, "Sandscout: Automatic detection of flaws in ios sandbox profiles," in *Proc. of ACM CCS*, 2016.

[55] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in cesar," in *International Symposium on programming*. Springer, 1982, pp. 337–351.

[56] Y. Resten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar, "Symbolic model checking with rich assertional languages," in *International Conference on Computer Aided Verification*. Springer, 1997, pp. 424–435.

[57] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[58] N. Halbwachs, Y.-E. Proy, and P. Roumanoff, "Verification of real-time systems using linear relation analysis," *Formal Methods in System Design*, vol. 11, no. 2, pp. 157–185, 1997.

[59] V. D'silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008.