Analysing the Provenance of IoT Data

Chiara Bodei[1] and Letterio Galletta[2(✉)]

[1] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
chiara.bodei@unipi.it
[2] IMT School for Advanced Studies, Lucca, Italy
letterio.galletta@imtlucca.it

Abstract. The Internet of Things (IoT) is leading to a smartification of our society: we are surrounded by many smart devices that automatically collect and exchange data of various kinds and provenance. Many of these data are critical because they are used to train learning algorithms, to control cyber-physical systems or to guide administrators to take decisions. Since the collected data are so important, many devices can be the targets of security attacks. Consequently, it is crucial to be able to trace data and to identify their paths inside a network of smart devices to detect possible threats. To help designers in this threat reasoning, we start from the modelling language IoT-LySA, and propose a Control Flow Analysis, a static analysis technique, for predicting the possible trajectories of data in an IoT system. Trajectories can be used as the basis for checking at design time whether sensitive data can pass through possibly dangerous nodes, and for selecting suitable security mechanisms that guarantee a reliable transport of data from sensors to servers using them. The computed paths are also interesting from an architectural point of view for deciding in which nodes data are collected, processed, communicated and stored.

1 Introduction

We are living the Internet of Things (IoT) revolution: we are surrounded by many interconnected devices (smart objects) equipped with sensors and actuators that automatically collect and transmit huge amounts of data over the net. Actually, a typical IoT system is a production chain that starts from raw data collected by sensors, continues with intermediate devices that perform data aggregation and ends with servers that store and process these data using learning algorithms. The results of the computation made on servers are used to take decisions or to trigger actuator actions in some part of the system.

Partially supported by Università di Pisa PRA_2018_66 *DECLWARE: Metodologie dichiarative per la progettazione e il deployment di applicazioni* and by MIUR project PRIN 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems).

P. Mori et al. (Eds.): ICISSP 2019, CCIS 1221, pp. 358–381, 2020.
https://doi.org/10.1007/978-3-030-49443-8_17

Secure transmission of data becomes even more crucial in IoT systems where devices can be physically attacked and data can be eavesdropped or altered during their communication. Therefore it is important that designers and administrators of such systems are aware of the provenance and the trajectories of data, especially when they are sensitive or when they impact on critical decisions.

Usually, formal methods offer designers tools to support the development of systems. In practice, designers build a mathematical model that describes the system we want to implement at a certain level of abstraction and they use formal verification techniques to reason about properties of the model, and consequently of the system it represents.

In this paper, we follow this approach to enable designers reasoning on data trajectories in IoT systems. Technically, we start from the formal specification language IoT-LySa, a process calculus recently proposed for IoT systems [5,9]. IoT-LySa allows designers to define a model of a system and fosters them to adopt a *Security by Design* development model. Indeed, designers can exploit the language to describe the network architecture of the system and how its components (smart objects or nodes) interact with each other. Furthermore, they can reason about the system correctness and robustness by using the Control Flow Analysis (CFA) of IoT-LySa.

This static analysis without running the system predicts (safely approximates) how data from sensors may spread across the system and how objects may interact. Technically, it "mimics" the behaviour of the system, by using abstract values in place of concrete ones and by modeling the consequences of each possible action. By inspecting the results of this "abstract simulation", designers can detect possible security vulnerabilities and intervene as early as possible during the design phase.

Here, we extend the original IoT-LySa CFA [5] for performing a *data path analysis*. The goal of this analysis is to predict how data travel across the network from specific data source nodes to data consumer nodes, computing all possible paths. Using the analysis results, a designer can investigate whether the trajectories taken from a particular piece of information include nodes that are considered potentially dangerous or that do not have an adequate security clearance for the information they are receiving.

Moreover, the trajectories can be used to make decisions on the architecture of the system by detecting critical nodes where data are collected and stored. Consequently, the information computed by our analysis may help designers in making educated decisions, on the exposure of both raw and aggregated data. Since the CFA over-approximates the behaviour of a given system, if the predicted trajectories do not show dangerous situations, we can be sure that at run time they will never happen. If instead they do, this means that there is a (even small) possibility of these situations to happen, and it can be worthwhile for the designer to carry out further investigations.

A short and preliminary version of the above results have been previously presented in [11]. As new contributions, in this paper, we systematise the full formal development of our data path analysis by presenting all its inference rules

together with the formal proofs of its correctness. In addition, we introduce the notion of scored trajectories that enriches the previous notion of simple trajectories with quantitative information. Finally, we apply our analysis on a completely new example, a Closed Circuit Television system, based on a visual sensor network.

Structure of the Paper. The paper is organised as follows. We introduce our approach, in Sect. 2, with an illustrative example that we use as case study. We briefly recall the process algebra IoT-LySa, in Sect. 3. Section 4 defines the CFA for the data path analysis and we show how to compute the data trajectories from the analysis result. In Sect. 5 we enrich the trajectories with the security scores on the involved nodes. Conclusions are in Sect. 6.

2 A Visual Sensor Network

In this section, we illustrate our methodology through a simple yet realistic scenario similar to the ones introduced in [12,13], where we model the problem of tracking some targets moving in the sensing space in the visual sensor network of a building for surveillance aims.

2.1 The Scenario

A Visual Sensor Network (VSN) consists of a large number of interconnected sensor nodes endowed with an imager (photo or video camera) and an embedded processor that communicate via wireless interfaces. Nodes can be different in computing power, amount of memory and energy consumption. Each node (or camera) can directly communicate to the nodes lying in its radio range, here called *physical neighbours*. Moreover, since each camera covers a part of the 3-D space by its conic field of view (FOV), there is a further notion of proximity for nodes: two nodes can collaborate and work on the same data when their FOVs intersect, i.e. the corresponding cameras monitor a common part of the space. Note that they can also be distant from each other. They are here called *logical neighbours*. Many applications of VSN address event detection and estimation of some metrics, based on the combination of different sensor readings, such as light and temperature sensors, or microphones. Since information is more valuable when close to its source and sending it is expensive, distributed approaches are preferable to centralised ones, where visual surveillance tasks are performed by collaborative groups of one or more camera nodes.

We suppose to have, as illustrated in Fig. 1 and as in [12], a Closed Circuit Television system in a building like a university department, with 14 corridor nodes, called of *type 1* and 4 room nodes, called of *type 2*, in the room considered more sensible. Both kinds of nodes are equipped with cameras and, for the sake of simplicity, with just one sensor. Moreover, nodes of type 2 have also alarm buzzers as actuators. According to the given topology both physical and logical neighbours are statically known. In particular, only two corridor nodes

with cameras with intersecting FOVs are not physical neighbour, c_0 and c_{13} and cannot communicate directly. In our model we choose w.l.o.g. that the camera nodes 2 and 1 serve as forwarder between the two nodes. Furthermore there are 5 aggregator nodes that collect information on a specific area of the building.

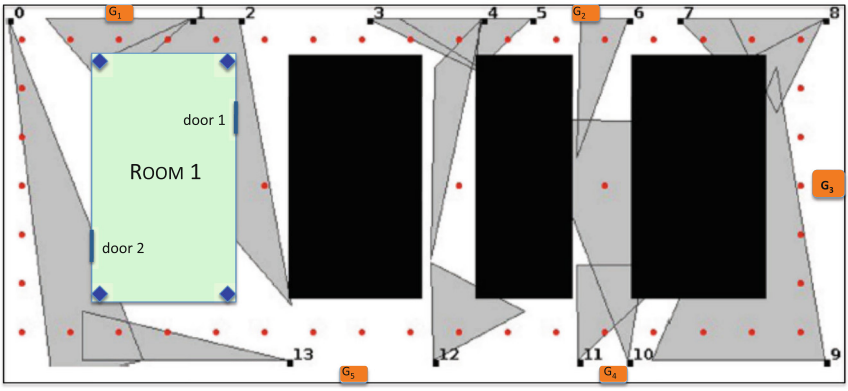

Fig. 1. The organisation of nodes in our Visual Sensor Network (modification of the Fig. 3 in [12]): little rectangles are the nodes with cameras of type 1, having each its FOV rooted in it and represented as a gray triangle, little diamonds are the nodes with cameras of type 2 and alarm actuators. Small orange rectangles are the aggregator nodes. Big rectangles are obstacles, the light green one is Room 1 and little circles are point of interest. (Color figure online)

In this application, corridor nodes detect intruders, in particular close to the sensible room ROOM 1. If one of the corridor nodes detects an intruder and checks that this one is close to one of the doors of the room, the corridor agent sends a warning message to the closest camera node inside the room.

2.2 The IoT-LySA Model

Here, we show how the scenario above can be easily modelled in IoT-LySa and what kind of information our CFA may provide to designers. In our model, the overall behavior of the network depends on the local processing at each node and on the inter-node communication, because the duty cycle of each camera involves only local computations and the exchange of partial approximations with logical and/or physical neighbors. Furthermore, we abstract away from the actual tracking algorithm used to reach a consistent view across nodes, and we model it as collaboration among nodes that exchange information (for further details see e.g. [12]).

The IOT-LYSA model, described in Table 1, consists of a finite number of nodes running in parallel (this is the meaning of the parallel composition operator | for nodes). Some of the terms are equipped with annotations (variables

Table 1. Visual Network System N.

$$
\begin{array}{rl}
 & \textbf{Whole Network} \\
N & = N_1^1 \mid ... \mid N_n^1 \mid N_1^2 \mid ... \mid N_s^2 \mid G_1 \mid ... \mid G_k \mid \\
 & \textbf{Node of type 1 with } i, r1, ..., rt \in [1,n] \\
N_i^1 & = \ell_i^1 : [P_i^1 \parallel C_i^1 \parallel S_i^1 \parallel B_i^1] \\
P_i^1 & = *[(z_{i1}^{v_{i1}} := 1^{a_{i1}}).(z_{i2}^{v_{i2}} := 2^{a_{i2}}).\langle\!\langle p(z_{i1}^{v_{i1}}, z_{i2}^{v_{i2}})^{p_i} \rangle\!\rangle \triangleright \{L_i^1\} \parallel \\
 & (;\, x_{ir1}^{vr1})^{X_i^{r1}} . ... (;\, x_{irt}^{vrt})^{X_i^{rt}} . \hat{P}_i^1 \\
 & detection_{1i}^{v1i} = d(z_{i1}^{vi1}, z_{i2}^{vi2}, x_{ir1}^{vr1}, ..., x_{irt}^{vrt})^{d_{1i}} . \\
 & detection_{1i}^{v1i} ? \langle\!\langle detection_{1i}^{v1i} \rangle\!\rangle \triangleright \{\ell_f^2\}] * \\
\hat{P}_{10}^1 & = (;\, x'_{13}) \\
\hat{P}_1^1 & = \langle\!\langle fw(x_1^0)^{fw_{11}} \rangle\!\rangle \triangleright \{\ell_{13}^1\} \\
\hat{P}_2^1 & = \langle\!\langle fw(x_2^{13})^{fw_{12}} \rangle\!\rangle \triangleright \{\ell_0^1\} \\
\hat{P}_{13}^1 & = (;\, x'_0) \\
\hat{P}_i^1 & = \tau \text{ for } i \neq 0, 1, 2, 13 \\
C_i^1 & = *[(\tau.1_i^1 := k_i^1).\tau] * \\
S_i^1 & = *[(\tau.2_i^1 := v_i^1).\tau] * \\
 & \textbf{Node of type 2 with } j, f \in [1,s] \\
N_j^2 & = \ell_j^2 : [Q_j^2 \parallel C_j^2 \parallel S_j^2 \parallel A_j^2 \parallel B_j^2] \\
Q_j^2 & = *[(;\, w_j^{v2j})^{W_j^2} . (w_{2j1}^{v2j1} := 1^{d_{2j1}}) . (w_{2j2}^{v2j2} := 2^{d_{2j2}}) . \\
 & confirm_{2j}^{w2j} = check(w_{det}^{v2j}, w^{v2j1}, w^{v2j2})^{c2j} . \\
 & confirm_{2j}^{w2j} ? \langle j, Alarm \rangle . \langle\!\langle 2j, confirm_{2j}^{w2j} \rangle\!\rangle \triangleright \{\ell_l\}] * \\
C_j^2 & = *[(\tau.1_j^2 := k_j^2).\tau] * \\
S_j^2 & = *[(\tau.2_j^2 := v_j^2).\tau] * \\
A_j^2 & = *[(\!|j, \{\mathsf{Alarm}\}).\tau] * \\
 & \textbf{Aggregator l with } l \in [1,k] \\
G_l & = \ell_l : *[R_l \parallel B_l] * \\
R_l & = (21; y_{l1}^{vl1})^{Y_1^l} . R_l^{21} \mid \cdots \mid (2s; y_{ls}^{vls})^{Y_s^l} . R_l^{2s}
\end{array}
$$

and function applications) and tags (input prefixes) that support the Control Flow Analysis in a way that will be clarified in the next section. Each node, uniquely identified by a label ℓ, consists of control processes and, possibly of camera, a sensor, and an actuator. Communication is multi-party: each node can send information to a set of nodes, provided that they are in the same transmission range. The communication patterns in the described scenario are not too complicate, so the example can serve the aim of illustrating our framework. Outputs and inputs must match to allow communication. In more detail, the output $\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L.\, P$ represents that the tuple $E_1, \cdots, E_k$ is sent to the nodes with labels in L. The input is instead modelled as $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_k)^X P$ and embeds pattern matching. In receiving an output tuple $E'_1, \cdots, E'_k$ of the same size (arity), the communication succeeds provided that the first j

elements of the output match the corresponding first elements of the input (i.e. $E_1 = E'_1, \cdots, E_j = E'_j$), and then the variables occurring in the input are bound to the corresponding terms in the output. Suppose e.g. to have a process P waiting a message that P knows to include the value v, together with a datum that is not known from P. The input pattern tuple would be: $(v; x)$. If P receives the matching tuple $\langle v, d\rangle$, the variable x can be bound to v, since the first component of the tuple matches with the corresponding value.

We first examine the camera nodes N_{1i} of type 1:

$$N_i^1 = \ell_i^1 : [P_i^1 \parallel C_i^1 \parallel S_i^1 \parallel B_i^1],$$

where ℓ_i^1 is the label that uniquely identifies the node, and B_i^1 abstracts other components we are not interested in, among which its store Σ_i^1. Each of these nodes is managed by a control process P_i^1, connected to a camera C_i^1 that covers a given FVO_i^1 and to a sensor S_i^1 that senses the environment in the area close to the node. They run in parallel (this is the meaning of the parallel composition operator $\parallel$ for processes). The node N_i^1 collects the data of its camera and its sensor, elaborates them with the help of a filter and pre-processing function p and then transmits its local result to its physical neighbours in L_i^1. In the meanwhile, the node collects all the local results of its neighbours and analyses them in order to detect a possible intruder in the observed corridors. If this is the case the node sends the camera node of type 2 closest to the intruder a warning message to inform that the intruder may enter the room. In the IoT-LySA jargon, the camera communicates the picture/video to the node by storing it in its reserved location 1_i^1 of the shared store, while the sensor stores the sensed data in the location 2_i^1. The action τ denotes internal actions of the sensor we are not interested in modelling, e.g. adjusting the camera focus. The construct $*[...]*$ denotes the iterative behaviour of processes and of sensors.

The control process P_i^1: (i) stores in the variables z_{i1}^{vi1} and z_{i2}^{vi2} (where $vi1$ and $vi2$ are the variable annotations) the data collected by the camera and the sensor, by means of the two assignments: $(z_{i1}^{vi1} := 1^{a_{i1}})$ and $(z_{i2}^{vi2} := 2^{a_{i2}})$; (ii) elaborates them with the help of a filter and pre-processing function p and (iii) then transmits its local result to its physical neighbours in L_{1i}, with the output $\langle\!\langle p(z_{i1}^{vi1}, z_{i2}^{vi2})^{p_i}\rangle\!\rangle \triangleright \{L_i^1\}$, where p_i is the label of the application of the function p. In the meanwhile the node collects all the local results of its neighbours, with the inputs $(; x_{ir1}^{vr1})^{X_i^{r1}} ... (; x_{irt}^{vrt})^{X_i^{rt}}$ (where inputs are enriched with tags X_i^{r1}, ..., X_i^{rt}) and analyses them in order to detect a possible intruder in the observed corridors, with the detection function $d(z_{i1}^{vi1}, z_{i2}^{vi2}, x_{1i}^{r1}, ..., x_{1i}^{rt})^{d_{1i}}$. If this is the case (if $detection_{1i}^{v1i}$ is true) the node sends the value to the camera node of type 2 closest to the intruder to inform that the intruder may enter the room: $\langle\!\langle detection_{1i}^{v1i}\rangle\!\rangle \triangleright \{\ell_f^2\}$. The part in blue in the pdf describes the communication for the special nodes N_{10} and N_{13}^1 that cannot communicate directly and that rely on the intermediation of the nodes N_1^1 and N_2^1. In particular, N_1^1 forwards the data received from N_0^1 to N_{13}^1, while N_2^1 forwards the data received from N_{13}^1 to $_0^1$.

In the node N_j^2, the process Q_j^2 waits for possible warning messages from corridor nodes. In case such a message arrives and its is bound to w_j^{v2j}, it collects the data w_{2j1}^{v2j1} and w_{2j2}^{v2j2} of its camera and its sensor, processes them with the help of the function *check* in order to verify the possible presence of intruder inside the room. In case the presence is confirmed, the node activates an alarm buzzer and sends an alarm to its aggregator node, with a label recalling its name. Each aggregator node G_l controls a subset of the camera nodes of both types. Again B_j^2 and B_l abstract other components we are not interested in, among which their stores Σ_j^2 and Σ_l. Some nodes can be attacked and therefore may alter or tamper data passing from there, thus potentially impacting on the whole system and making the building vulnerable.

Since our analysis identifies the possible trajectories of data in the system, we can analyse these trajectories in order to check which are more risky w.r.t. the involved nodes. To this aim we suppose that operators can provide a security score for each node that measures its risk of being attacked. Reasoning on the formal model of the system and on the possible trajectories of data can be exploited to determine possible countermeasures such as redundancy, by introducing some new components that can mitigate the impact of attacks.

3 Overview of IoT-LySA

We now present a briefly overview of IoT-LySA [5,9], a specification language recently proposed for designing IoT systems. It is an adaption of LySA [3], a process calculus introduced to specify and analyse cryptographic protocols and checking their security properties (see e.g. [16,17]).

Differently from other process algebraic approaches introduced to model IoT systems, e.g. [19–22], IoT-LySA provides a design framework that includes a static semantics to support verification techniques and tools for certifying properties of IoT applications.

3.1 Syntax

Systems in IoT-LySA consist of a pool of nodes (things), each of which hosts a store for internal communication, sensors and actuators, and a finite number of control processes that detail how data are to be processed and exchanged among the node. We assume that each sensor (actuator) in a node with label ℓ is uniquely identified by an index $i \in \mathcal{I}_\ell$ ($j \in \mathcal{J}_\ell$, resp). A sensor is an active entity that reads data from the physical environment at its own fixed rate, and deposits them in the local store of. Actuators instead are passive: they just wait for a command to become active and operate on the environment. Data are represented by terms. Annotations $a, a', a_i, ...$, ranged over by $\mathcal{A}$, identify the occurrences of terms. They are used in the analysis and do not affect the dynamic semantics in Table 3. The set of nodes and all the node components are defined by the syntax in Table 2, that completes the one in [11].

Table 2. Syntax.

$\mathcal{E} \ni E ::=$ *annotated terms*			$\mathcal{M} \ni M ::=$ *terms*		
M^a	annotated term with $a \in \mathcal{A}$		v	value ($v \in \mathcal{V}$)	
			i	sensor location ($i \in \mathcal{I}_\ell$)	
			x		
			$\{E_1, \cdots, E_r\}_{k_0}$	encryption with key $k_0 \in \mathcal{K}$	
			$f(E_1, \cdots, E_r)$	function on data	
$\mathcal{N} \ni N ::=$ *systems of nodes*			$\mathcal{B} \ni B ::=$ *node components*		
0	empty system		Σ_ℓ	node store	
$\ell : [B]$	single node ($\ell \in \mathcal{L}$)		P	process	
$N_1 \mid N_2$	par. composition		S	sensor (label $i \in \mathcal{I}_\ell$)	
			A	actuator (label $j \in \mathcal{J}_\ell$)	
			$B \parallel B$	par. composition	
$\mathcal{S} \ni S$	$::=$ *sensors*		$\mathcal{A} \ni A$	$::=$ *actuators*	
0	inactive sensor		0	inactive actuator	
$\tau.S$	internal action		$\tau.A$	internal action	
$i := v.\, S$	store of $v \in \mathcal{V}$ by the i^{th} sensor		$(\!\lvert j, \Gamma \rvert\!).\, A$	command for actuator j	
			$\gamma.A$	triggered action ($\gamma \in \Gamma$)	
h	iteration var.		h	iteration var.	
$\mu\, h\, .\, S$	tail iteration		$\mu\, h\, .\, S$	tail iteration	

$P ::=$ *control processes*	
0	inactive process
$\langle\langle E_1, \cdots, E_r \rangle\rangle \triangleright L.\, P$	asynchronous multi-output $L \subseteq \mathcal{L}$
$(E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_r)^X.\, P$	input (with matching and tag)
decrypt E as $\{E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_r\}_{k_0}$ in P	decryption with key k_0 (with match.)
$E?P : Q$	conditional statement
h	iteration variable
$\mu h.\ P$	tail iteration
$x^a := E.\, P$	assignment to $x \in \mathcal{X}$
$\langle j, \gamma \rangle.\, P$	output of action γ to actuator j

We assume as given a finite set $\mathcal{K}$ of secret keys owned by nodes, exchanged at deployment time in a secure way, as it is often the case [26]. Terms come with annotations $a \in \mathcal{A}$. The encryption function $\{E_1, \cdots, E_r\}_{k_0}$ returns the result of encrypting values E_i for $i \in [1, r]$ under the shared key k_0. We assume to have perfect cryptography. The term $f(E_1, \cdots, E_r)$ is the application of function f to r arguments; we assume given a set of primitive functions, typically for aggregating or comparing values. We assume the sets $\mathcal{V}$, $\mathcal{I}_\ell$, $\mathcal{J}_\ell$, $\mathcal{K}$be pairwise disjoint.

Each node $\ell : [B]$ is uniquely identified by a label $\ell \in \mathcal{L}$ that may represent further information on the node (e.g. node location). Sets of nodes are described through the (associative and commutative) operator $\mid$ for parallel composition. The system 0 has no nodes. Inside a node $\ell : [B]$ there is a finite set of components combined by means of the parallel operator $\parallel$. We impose that there is a *single* store $\Sigma_\ell : \mathcal{X} \cup \mathcal{I}_\ell \rightarrow \mathcal{V}$, where $\mathcal{X}, \mathcal{V}$ are the sets of variables and of values (integers, booleans, ...), resp.

The store is essentially an array whose indexes are variables and sensors identifiers $i \in \mathcal{I}_\ell$ (no need of α-conversions). We assume that store accesses are atomic, e.g. through CAS instructions [18]. The other node components are control processes P, and sensors S (less than $\#(\mathcal{I}_\ell)$), and actuators A (less than $\#(\mathcal{J}_\ell)$) the actions of which are in Act.

The prefix $\langle\!\langle E_1, \cdots, E_r \rangle\!\rangle \triangleright L$ implements a simple form of multi-party communication: the tuple obtained by evaluating $E_1, \ldots, E_r$ is asynchronously sent to the nodes with labels in L that are "compatible" (according, among other attributes, to a proximity-based notion). The input prefix $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_r)^X$ receives a r-tuple, provided that its first j elements match the corresponding input ones, and then assigns the variables (after ";") to the received values. Otherwise, the r-tuple is not accepted. As in [2], each input in the syntax of processes P has a tag $X \in \mathbf{X}$, which is exploited to support the analysis and does not affect the dynamic semantics. A process repeats its behaviour, when defined through the tail iteration construct $\mu h.P$ (h is the iteration variable), intuitively rendered with $*[...]*$ in the motivating example. The process $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_r\}_{k_0}\ \mathsf{in}\ P$ tries to decrypt the result of the expression E with the shared key $k_0 \in \mathcal{K}$. Also in this case, if the pattern matching succeeds, the process continues as P and the variables $x_{j+1}, \ldots, x_r$ are suitably assigned.

A sensor can perform an internal action τ or put the value v, gathered from the environment, into its store location i. An actuator can perform an internal action τ or execute one of its actions γ, received from its controlling process. Sensors and actuators can iterate. For simplicity, here we neither provide an explicit operation to read data from the environment, nor to describe the impact of actuator actions on the environment.

Operational Semantics

Our reduction semantics is based on the following *Structural congruence* $\equiv$ on nodes and node components. It is standard except for rule (4) that equates a multi-output with no receivers and the inactive process, and for the fact that inactive components of a node are all coalesced.

(1) $(\mathcal{N}/_\equiv, |, \mathsf{0})$ is a commutative monoid
(2) $(\mathcal{B}/_\equiv, \|, \mathsf{0})$ is a commutative monoid
(3) $\mu\, h\, .\, X \equiv X\{\mu\, h\, .\, X/h\}$ for $X \in \{P, A, S\}$
(4) $\langle\langle E_1, \cdots, E_r \rangle\rangle : \emptyset.\ \mathsf{0} \equiv \mathsf{0}$

The two-level *reduction relation* $\rightarrow$ is defined as the least relation on nodes and its components satisfying the set of inference rules in Table 3. For the sake of simplicity, we use one relation. We assume the standard denotational interpretation $[\![E]\!]_\Sigma$ for evaluating terms.

Table 3. Reduction semantics (the upper part on node components, the lower one on nodes), where $X \in \{S, A\}$ and $Y \in \{N, B\}$.

$$\text{(S-store)}\quad \frac{}{\Sigma \parallel i^a := v^{a'}.\, S_i \parallel B \rightarrow \Sigma\{v/i\} \parallel S_i \parallel B} \qquad \text{(Asgm)}\quad \frac{[\![E]\!]_\Sigma = v}{\Sigma \parallel x^a := E.\, P \parallel B \rightarrow \Sigma\{v/x\} \parallel P \parallel B}$$

$$\text{(Cond1)}\quad \frac{[\![E]\!]_\Sigma = \texttt{true}}{\Sigma \parallel E?\, P_1 : P_2 \parallel B \;\rightarrow\; \Sigma \parallel P_1 \parallel B} \qquad \text{(Cond2)}\quad \frac{[\![E]\!]_\Sigma = \texttt{false}}{\Sigma \parallel E?\, P_1 : P_2 \parallel B \;\rightarrow\; \Sigma \parallel P_2 \parallel B}$$

$$\text{(A-com)}\quad \frac{\gamma \in \Gamma}{\langle j, \gamma\rangle.\, P \parallel (\!|j, \Gamma|\!).\, A \parallel B \;\rightarrow\; P \parallel \gamma.\, A \parallel B} \qquad \text{(Act)}\quad \frac{}{\gamma.A \;\rightarrow\; A} \qquad \text{(Int)}\quad \frac{}{\tau.\, X \;\rightarrow\; X}$$

$$\text{(Decr)}\quad \frac{[\![E]\!]_\Sigma = \{v_1, \cdots, v_r\}_{k_0} \;\wedge\; \bigwedge_{i=1}^{j} v_i = [\![E'_i]\!]_\Sigma}{\Sigma \parallel \texttt{decrypt}\ E\ \texttt{as}\ \{E'_1, \cdots, E'_j; x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r}\}_{k_0}\ \texttt{in}\ P \parallel B \rightarrow \Sigma\{v_{j+1}/x_{j+1}, \cdots, v_r/x_r\} \parallel P \parallel B}$$

$$\text{(Ev-out)}\quad \frac{\bigwedge_{i=1}^{r} v_i = [\![E_i]\!]_\Sigma}{\Sigma \parallel \langle\!\langle E_1, \cdots, E_r\rangle\!\rangle \triangleright L.\, P \parallel B \;\rightarrow\; \Sigma \parallel \langle\!\langle v_1, \cdots, v_r\rangle\!\rangle \triangleright L.0 \parallel P \parallel B}$$

$$\text{(Multi-com)}\quad \frac{\ell_2 \in L \wedge\ Comp(\ell_1, \ell_2) \wedge\ \bigwedge_{i=1}^{j} v_i = [\![E_i]\!]_{\Sigma_2}}{\begin{array}{l}\ell_1 : [\langle\!\langle v_1, \cdots, v_r\rangle\!\rangle \triangleright L.\, 0 \parallel B_1] \;\mid\; \ell_2 : [\Sigma_2 \parallel (E_1, \cdots, E_j; x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r})^X.Q \parallel B_2] \;\rightarrow \\ \ell_1 : [\langle\!\langle v_1, \cdots, v_r\rangle\!\rangle \triangleright L \setminus \{\ell_2\}.\, 0 \parallel B_1] \;\mid\; \ell_2 : [\Sigma_2\{v_{j+1}/x_{j+1}, \cdots, v_r/x_r\} \parallel Q \parallel B_2]\end{array}}$$

$$\text{(Node)}\quad \frac{B \;\rightarrow\; B'}{\ell : [B] \;\rightarrow\; \ell : [B']} \qquad \text{(ParN)}\quad \frac{N_1 \rightarrow N'_1}{N_1 | N_2 \rightarrow N'_1 | N_2} \qquad \text{(ParB)}\quad \frac{B_1 \rightarrow B'_1}{B_1 \| B_2 \rightarrow B'_1 \| B_2} \qquad \text{(CongrY)}\quad \frac{Y'_1 \equiv Y_1 \rightarrow Y_2 \equiv Y'_2}{Y'_1 \rightarrow Y'_2}$$

The first two semantic rules implement the (atomic) asynchronous update of shared variables inside nodes, by using the standard notation $\Sigma\{-/-\}$. According to (S-store), the i^{th} sensor uploads the value v, gathered from the environment, into its store location i. According to (Asgm), a control process updates the variable x with the value of E. The rules for conditional (Cond1) and (Cond2) are as expected. In the rule (A-com) a process with prefix $\langle j, \gamma\rangle$ commands the j^{th} actuator to perform the action γ, if it is one of its actions. The rule (Act) says that the actuator performs the action γ. Similarly, for the rules (Int) for internal actions for representing activities we are not interested in. The rules (Ev-out) and (Multi-com) drive asynchronous IoT-LySA multi-communications and are explained as follows. In the first rule, to send a message $\langle\!\langle v_1, ..., v_r\rangle\!\rangle$ obtained by the evaluation of $\langle\!\langle E_1, ..., E_r\rangle\!\rangle$, a node with label ℓ spawns a new process, running in parallel with the continuation P; this new process offers the evaluated tuple to all the receivers with labels in L. In the second rule, the message coming from ℓ_1 is received by a node labelled ℓ_2, provided that: (i) ℓ_2 belongs to the set L of possible receivers, (ii) the two nodes satisfy a compatibility predicate $Comp$ (e.g. when they are in the same transmission range), and (iii) that the first j values match with the evaluations of the first j terms in the input. Moreover, the

label ℓ_2 is removed by the set of receivers L of the tuple. The spawned process terminates when all the receivers have received the message (L is empty).

The rule (Decr) tries to decrypt the result $\{v_1, \cdots, v_r\}_k$ of the evaluation of E with the key k_0, and matches it against the pattern $\{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_r\}_{k_0}$. Concerning communication, when this match succeeds the variables after the semicolon ";" are assigned to values resulting from the decryption. The last rules propagate reductions across parallel composition ((ParN) and (ParB)) and nodes (Node), while (CongrY) is the standard reduction rule for congruence for nodes and node components.

4 Control Flow Analysis

Here we present a CFA for approximating the abstract behaviour of a system of nodes and for tracking the trajectories of data. This CFA follows the same schema of the one in [5] and in particular of the one in [8] for IoT-LySa. However, here we use different abstract values. Intuitively, abstract values "symbolically" represent runtime data so as to encode where these data have been introduced. Finally, we show how to use the CFA results to check which are the possible trajectories of these data.

Abstract Values. Abstract values correspond to concrete values for sensors, data, functions, and encryptions, and also record the annotations. Since the dynamic semantics may introduce encrypted terms with an arbitrarily nesting level, we have the special abstract values $\top^a$ that denote all the terms with a depth greater than a given threshold d. During the analysis, to cut these values, we will use the function $\lfloor - \rfloor_d$. Its definition is quite intuitive because we recursively visit the abstract value and cut it when we reach the relevant depth. Formally, abstract values are defined as follows, where $a \in \mathcal{A}$.

$\hat{\mathcal{V}} \ni \hat{v} ::=$	*abstract terms*
$(\top, a)$	value denoting cut
(v, a)	value for clear data
$(f(\hat{v}_1, \cdots, \hat{v}_n), a)$	value for aggregated data
$(\{\hat{v}_1, \cdots, \hat{v}_n\}_{k_0}, a)$	value for encrypted data

For simplicity, hereafter we write them as $\top^a, \nu^a, \{\hat{v}_1, \cdots, \hat{v}_n\}^a_{k_0}$, and indicate with $\downarrow_i$ the projection function on the i^{th} component of the pair. We naturally extend the projection to sets, i.e. $\hat{V}_{\downarrow_i} = \{\hat{v}_{\downarrow_i} | \hat{v} \in \hat{V}\}$, where $\hat{V} \subseteq \hat{\mathcal{V}}$. In the abstract value v^a, v abstracts the concrete value from sensors or computed by a function in the concrete semantics, while the first value of the pair $\{\hat{v}_1, \cdots, \hat{v}_n\}^a_{k_0}$ abstracts encrypted data. The second component records the annotation associated to the corresponding term. Note that once given the set of encryption functions occurring in a node N, the abstract values are finitely many.

To extract all the annotations of an abstract value, included the ones possibly nested in it, we use the following auxiliary function.

Definition 1. *Give an abstract value $\hat{v} \in \hat{\mathcal{V}}$, we define the set of labels* $\mathbf{A}(\hat{v})$ *inductively as follows.*

- $\mathbf{A}(\top, a) = \mathbf{A}(v, a) = \{a\}$
- $\mathbf{A}(f(\hat{v}_1, \cdots, \hat{v}_n), a) = \{a\} \cup \bigcup_{i=1}^{n} \mathbf{A}(\hat{v_i})$
- $\mathbf{A}(\{\hat{v}_1, \cdots, \hat{v}_n\}_{k_0}, a) = \{a\} \cup \bigcup_{i=1}^{n} \mathbf{A}(\hat{v_i})$

Trajectories. We now introduce the notion of trajectories of data, in turn composed by micro-trajectories representing a single hop in the communication.

Definition 2. *Given a set of labels $\mathcal{L}$, a set of input tags* $\mathbf{X}$*, we define a* micro-trajectory *μ as a pair $((\ell, \ell'), X) \in (\mathcal{L} \times \mathcal{L}) \times \mathbf{X}$. A* trajectory *$\tau$ is a list of micro-trajectories $[\mu_1, ..., \mu_n]$, such that $\forall \mu_i, \mu_{i+1}$ with $\mu_i = ((\ell_i, \ell'_i), X_i)$ and $\mu_{i+1} = ((\ell_{i+1}, \ell'_{i+1}), X_{i+1})$, $\ell'_i = \ell_{i+1}$.*

In our analysis, trajectories can be obtained, starting from a set of micro-trajectories and by suitably composing them in order. Trajectories can be composed if the head of the second trajectory is equal to tail of the first. In this case the two trajectories can be merged. Technically, we use a closure of a set of micro-trajectories, the inductive definition of which follows.

Definition 3. *Given a set of micro-trajectories $S \in ((\mathcal{L} \times \mathcal{L}) \times \mathbf{X})$*

- $\forall((\ell, \ell'), X) \in S.\ [((\ell, \ell'), X)] \in Clos_X(S)$*;*
- $\forall[L, ((\ell, \ell'), X)],\ [((\ell', \ell''), X'), L''] \in S.\ [L, ((\ell, \ell'), X), ((\ell', \ell''), X'), L''] \in Clos_X(S)$.

CFA Validation and Correctness. We now have all the ingredients to define our CFA to approximate communications and data stored and exchanged and, in particular, the micro-trajectories. We specify our analysis in a logical form through a set of inference rules expressing the validity of the analysis results. The analysis result is a tuple $(\widehat{\Sigma}, \kappa, \Theta, T, \rho)$ (a pair $(\widehat{\Sigma}, \Theta)$ when analysing a term), called *estimate* for N (for E), where $\widehat{\Sigma}$, κ, Θ, T, and ρ are the following *abstract domains*:

- the union $\widehat{\Sigma} = \bigcup_{\ell \in \mathcal{L}} \hat{\Sigma}_\ell$ of the sets $\hat{\Sigma}_\ell : \mathcal{X} \cup \mathcal{I}_\ell \rightarrow 2^{\widehat{\mathcal{V}}}$ of abstract values that may possibly be associated to a given location in $\mathcal{I}_\ell$ or a given variable in $\mathcal{X}$,
- a set $\kappa : \mathcal{L} \rightarrow \mathcal{L} \times \bigcup_{i=1}^{k} \widehat{\mathcal{V}}^i$ of the messages that may be received by the node ℓ, and
- a set $\Theta : \mathcal{L} \rightarrow \mathcal{A} \rightarrow 2^{\widehat{\mathcal{V}}}$ of the information of the actual values computed by each labelled term M^a in a given node ℓ, at run time.
- a set $\rho : \mathbf{X} \rightarrow \mathcal{L} \times \bigcup_{i=1}^{k} \widehat{\mathcal{V}}^i$ is the sets of output tuples that may be accepted by the input variables X.
- a set $T = \mathcal{A} \rightarrow (\mathcal{L} \times \mathcal{L}) \times \mathbf{T}$ of possible micro-trajectories related to the abstract values.

Note that the component T is new, and also the combined use of these five components is new and allows us to potentially integrate the present CFA with the previous analyses of IoT-LySA.

An available estimate has to be validated correct. This requires that it satisfies the judgements defined according to the syntax of nodes, node components and terms. They are defined by the set of clauses presented in Tables 4 and 5.

Table 4. Analysis of labelled terms $(\widehat{\Sigma}, \Theta) \models_\ell M^a$.

$$\frac{(i,a) \in \Theta(\ell)(a)}{(\widehat{\Sigma}, \Theta) \models_\ell i^a} \qquad \frac{(v,a) \in \Theta(\ell)(a)}{(\widehat{\Sigma}, \Theta) \models_\ell v^a} \qquad \frac{\widehat{\Sigma}_\ell(x) \subseteq \Theta(\ell)(a)}{(\widehat{\Sigma}, \Theta) \models_\ell x^a}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\widehat{\Sigma}, \Theta) \models_\ell M_i^{a_i} \wedge \\ \forall\, \hat{v}_1, .., \hat{v}_r : \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \;\Rightarrow\; (\lfloor \{\hat{v}_1, .., \hat{v}_r\}_{k_0} \rfloor_d, a) \in \Theta(\ell)(a)\end{array}}{(\widehat{\Sigma}, \Theta) \models_\ell \{M_1^{a_1}, .., M_r^{a_r}\}_{k_0}^a}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\widehat{\Sigma}, \Theta) \models_\ell M_i \wedge \\ \forall\, \hat{v}_1, .., \hat{v}_r : \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \;\Rightarrow\; (f(\hat{v}_1, .., \hat{v}_r), a) \in \Theta(\ell)(a)\end{array}}{(\widehat{\Sigma}, \Theta) \models_\ell f(M_1^{a_1}, .., M_r^{a_r})^a}$$

The judgement $(\widehat{\Sigma}, \Theta) \models_\ell M^a$, defined by the rules in Table 4, requires that $\Theta(\ell)(a)$ includes all the abstract values $\hat{v}$ associated to M^a. In the case of sensor identifiers, i^a and values v^a must be included in $\Theta(\ell)(a)$. According to the clause for the variable x^a, an estimate is valid if $\Theta(\ell)(a)$ includes the abstract values bound to x collected in $\widehat{\Sigma}_\ell$.

The rule for analysing compound terms requires that the components are in turn analysed. The penultimate rule deals with the application of an r-ary encryption. To do that (i) it analyses each term $M_i^{a_i}$, and (ii) for each r-tuple of values $(\hat{v}_1, \cdots, \hat{v}_r)$ in $\Theta(\ell)(a_1) \times \cdots \times \Theta(\ell)(a_r)$, it requires that the abstract structured value $\{\hat{v}_1, \cdots, \hat{v}_r\}_{k_0}^a$, cut at depth d, belongs to $\Theta(\ell)(a)$. The special abstract value $\top^a$ will end up in $\Theta(\ell)(a)$ if the depth of the term exceeds d. The last rule is for the application of an r-ary function f. Also in this case, (i) it analyses each term $M_i^{a_i}$, and (ii) for all r-tuples of values $(\hat{v}_1, \cdots, \hat{v}_r)$ in $\Theta(\ell)(a_1) \times \cdots \times \Theta(\ell)(a_r)$, it requires that the composed abstract value $f(\hat{v}_1, \cdots, \hat{v}_r)^a$ belongs to $\Theta(\ell)(a)$.

The judgements for nodes with the form $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N$ are defined by the rules in Table 5. The rules for the *inactive node* and for *parallel composition* are standard. The rule for a single node $\ell : [B]$ requires that its internal components B are in turn analysed; in this case we the use rules with judgements $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell B$, where ℓ is the label of the enclosing node. The rule connecting actual stores Σ with abstract ones $\widehat{\Sigma}$ requires the locations of sensors to contain the corresponding abstract values. The rule for sensors is trivial,

because we are only interested in the users of their values. The rule for actuators is equally trivial, because we model actuators as passive entities. The rules for processes require analysing the immediate sub-processes.

Table 5. Analysis of nodes $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N$, and of node components $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell B$.

$$\frac{}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models 0} \qquad \frac{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell B}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models \ell : [B]} \qquad \frac{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N_1 \wedge (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N_2}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N_1 \mid N_2}$$

$$\frac{\forall\, i \in \mathcal{I}_\ell . i^\ell \in \hat{\Sigma}_\ell(i)}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell \Sigma} \qquad \frac{}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell S} \qquad \frac{}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell A}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\widehat{\Sigma}, \Theta) \models_\ell M_i^{a_i} \ \wedge\ (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P \ \wedge \\ \forall \hat{v}_1, \cdots, \hat{v}_r : \ \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \ \Rightarrow \forall \ell' \in L : (\ell, \langle\!\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \kappa(\ell')\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell \langle\!\langle M_1^{a_1}, \cdots, M_r^{a_r} \rangle\!\rangle \triangleright L.\, P}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{j} (\widehat{\Sigma}, \Theta) \models_\ell M_i^{a_i} \ \wedge \\ \forall (\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \kappa(\ell) : \ Comp(\ell', \ell) \Rightarrow \\ (\bigwedge_{i=j+1}^{r} \hat{v}_i \in \hat{\Sigma}_\ell(x_i) \ \wedge \\ (\ell', \langle\!\langle (\hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \rho(X) \ \wedge\ \forall a \in \mathbf{A}(\hat{v}_i).((\ell, \ell'), X) \in T(a) \\ \wedge (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P)\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell (M_1^{a_1}, \cdots, M_j^{a_j}; x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r})^X . P}$$

$$\frac{\begin{array}{c}(\widehat{\Sigma}, \Theta) \models_\ell M^a \ \wedge\ \bigwedge_{i=1}^{j} (\widehat{\Sigma}, \Theta) \models_\ell M_i^{a_i} \ \wedge \\ \forall \{\hat{v}_1, \cdots, \hat{v}_r\}_{k_0}^b \in \Theta(\ell)(a) \Rightarrow \left(\bigwedge_{i=j+1}^{r} \hat{v}_i \in \hat{\Sigma}_\ell(x_i) \ \wedge\ (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P\right)\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell \mathsf{decrypt}\ M^a\ \mathsf{as}\ \{M_1^{a_1}, \cdots, M_j^{a_j}; x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r}\}_{k_0}\ \mathsf{in}\ P}$$

$$\frac{\begin{array}{c}(\widehat{\Sigma}, \Theta) \models_\ell M^a \ \wedge \\ \forall\, \hat{v} \in \Theta(\ell)(a) \ \Rightarrow \hat{v} \in \hat{\Sigma}_\ell(x) \ \wedge (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell x^{a_x} := M^a . P} \qquad \frac{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{n\ell} P}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell \langle j, \gamma \rangle . P}$$

$$\frac{\begin{array}{c}(\widehat{\Sigma}, \Theta) \models_\ell M^a \ \wedge \\ (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P_1 \ \wedge\ (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P_2\end{array}}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell M^a ? P_1 : P_2} \qquad \frac{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell B_1 \wedge (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell B_2}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell B_1 \| B_2}$$

$$\frac{}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell 0} \qquad \frac{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell \mu h.\, P} \qquad \frac{}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell h}$$

An estimate is valid for *multi-output*, if it is valid for the continuation of P and the set of messages communicated by the node ℓ to each node ℓ' in L, includes all the messages obtained by the evaluation of the r-tuple $\langle\!\langle M_1^{a_1}, \cdots, M_r^{a_r} \rangle\!\rangle$. More precisely, the rule (i) finds the sets $\Theta(\ell)(a_i)$ for each term $M_i^{a_i}$, and (ii) for all tuples of values $(\hat{v}_1, \cdots, \hat{v}_r)$ in $\Theta(\ell)(a_1) \times \cdots \times \Theta(\ell)(a_r)$ it checks whether they belong to $\kappa(\ell')$ for each $\ell' \in L$. Symmetrically, the rule for *input* requires that the values inside messages that can be sent to the node ℓ, passing the pattern matching, are included in the estimates of the variables $x_{j+1}, \cdots, x_r$. More in detail, the rule analyses each term $M_i^{a_i}$, and requires that for any message that the node with label ℓ can receive, i.e. $(\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_j, \hat{v}_{j+1}, \ldots, \hat{v}_r \rangle\!\rangle)$ in $\kappa(\ell)$, provided that the two nodes can communicate (i.e. $Comp(\ell', \ell)$), the abstract

values $\hat{v}_{j+1}, \ldots, \hat{v}_r$ are included in the estimates of $x_{j+1}, \cdots, x_r$. Furthermore, the micro-trajectory $((\ell, \ell'), X)$ is recorded in the T component for each annotation related (via $\mathbf{A}$) to the abstract value $\hat{v}_i$, to record that the abstract value $\hat{v}_i$ coming from the node ℓ can reach the node labelled ℓ', in the input with tag X. For instance, if $\hat{v}_i = (f((v_{i1}, a_{i1}), (v_{i2}, a_{i2})), a_i)$, then the micro-trajectory is recorded in $T(a_i)$, $T(a_{i1})$ and $T(a_{i2})$. Finally, the ρ component records the sets of output tuples that can be bound in the input with tag X.

The rule for *decryption* is similar to the one for communication: it also requires that the keys coincide. The rule for *assignment* requires that all the values $\hat{v}$ in the estimate $\Theta(\ell)(a)$ for M^a belong to $\widehat{\Sigma}_\ell(x)$. The rules for the *inactive process*, for *parallel composition*, and for *iteration* are standard (we assume that each iteration variable h is uniquely bound to the body P).

Given a term E annotated by a, the over-approximation of its possible trajectories is obtained by computing the trajectory closure of the set composed by all the possibly enriched micro-trajectories $((\ell, \ell'), X)$ or $((\ell_i, \ell_i'), (\phi(\ell_i), \phi(\ell_i')), X_i)$ in $T(a)$.

$$Trajectories(E^a) = Clos_X(T(a))$$

Therefore, our analysis enables traceability of data. For every exchanged message $\langle\!\langle v_1, \ldots, v_r \rangle\!\rangle$, the CFA keeps track of the possible paths of each of its components v_i and, in turn, for each v_i it keeps recursively track of the paths of the possible data used to compose it.

Example 1. To better understand how our analysis works, we apply it to the following simple system, where P_i' and B_i (with $i = 1, 2, 3$) abstract other components we are not interested in.

$$\ell_1 : [\langle\!\langle v^{a_1} \rangle\!\rangle \triangleright \ell_2.\, P_1' \parallel B_1] \mid \ell_2 : [(; x_2^{b_2})^{X_2}.\langle\!\langle f(x_2^{b_x})^m \rangle\!\rangle \triangleright \ell_3.P_2' \parallel B_2] \mid \ell_3 : [(; y_3^{c_3})^{Y_3}.P_3' \parallel B_3]$$

Every valid estimate $(\widehat{\Sigma}, \kappa, \Theta, T, \rho)$ must include at least the following entries, with $d = 4$.

$$\begin{array}{l}
\Theta(\ell_1)(a_1) \supseteq \{v^{a_1}\} \\
\kappa(\ell_2) \supseteq \{(\ell_1, \langle\!\langle v^{a_1} \rangle\!\rangle\} \\
\rho(X_2) \supseteq \{(\ell_1, \langle\!\langle v^{a_1} \rangle\!\rangle\} \\
\hat{\Sigma}_{\ell_2}(x^{b_2}) \supseteq \{v^{a_1}\} \\
T(a_1) \supseteq \{((\ell_1, \ell_2), X_2)\} \\
\Theta(\ell_2)(b_2) \supseteq \{f(v^{a_1})^m\} \\
\kappa(\ell_3) \supseteq \{(\ell_3, \langle\!\langle f(v^{a_1} \rangle\!\rangle)^m\} \\
\rho(Y_3) \supseteq \{(\ell_2, \langle\!\langle v^{a_1} \rangle\!\rangle\} \\
\hat{\Sigma}_{\ell_3}(y^{c_3}) \supseteq \{f(v^{a_1})^m\} \\
T(a_1) \supseteq \{((\ell_2, \ell_3), Y_3)\} \\
T(m) \supseteq \{((\ell_2, \ell_3), Y_3)\}
\end{array}$$

Indeed, an estimate must satisfy the checks of the CFA rules. The validation of the system requires the validation of each node, i.e. $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N_i$ and of the processes there included, i.e. $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_i} P_i$, with $i = 1, 2, 3$. In particular, the validation of the process included in N_1, i.e. $\langle\!\langle v^{a_1} \rangle\!\rangle \triangleright \{\ell_2\}$ holds because the checks required by CFA clause for output succeed. We can indeed verify that $(\widehat{\Sigma}, \Theta) \models_\ell v^{a_1}$ holds because $v^{a_1} \in \Theta(\ell_1)(a_1)$, according to the CFA

clause for names. Furthermore $(\ell_1, \langle\!\langle v^{a_1} \rangle\!\rangle) \in \kappa(\ell_2)$. This suffices to validate the output, by assuming that the continuation P_1' is validated as well. We have the following instantiation of the clause for output.

$$\frac{v^{a_1} \in \Theta(\ell_1)(a_1)}{(\widehat{\Sigma}, \Theta) \models_\ell v^{a_1}} \ \wedge \ (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell P_1' \wedge$$

$$\frac{v^{a_1} \in \Theta(\ell_1)(a_1) \ \Rightarrow (\ell_1, \langle\!\langle v^{a_1} \rangle\!\rangle) \in \kappa(\ell_2)}{(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_1} \langle\!\langle v^{a_1} \rangle\!\rangle \triangleright \{\ell_2\}.\, P_1'}$$

Instead $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_1} (; x_2^{b_2})^{X_2}.\langle\!\langle f(x_2^{b_x})^m \rangle\!\rangle \triangleright \ell_3.P_2'$ holds because the checks for the CFA clause for input succeed. From $(\ell_1, \langle\!\langle v^{a_1} \rangle\!\rangle) \in \kappa(\ell_2)$, we can indeed obtain that $\rho(X_2) \supseteq \{(\ell_1, \langle\!\langle v^{a_1} \rangle\!\rangle\}$, $\hat{\Sigma}_{\ell_2}(x^{b_2}) \supseteq \{v^{a_1}\}$, and that $T(a_1) \supseteq \{((\ell_1, \ell_2), X_2)\}$. The other entries can be similarly validated as well. Finally note that from $T(a_1) \supseteq \{((\ell_1, \ell_2), X_2)$ and $T(a_1) \supseteq \{((\ell_2, \ell_3), Y_3)\}$, we can obtain the trajectory $[((\ell_1, \ell_2), X_2), ((\ell_2, \ell_3), Y_3)]$, by applying $\widehat{Clos}_X$ to $(T(a_1))$. Note that the second component of each micro-trajectory records the input in which the communication of the value may take place and can help in statically backtracking the data path. Given the score of each node, the cost of the corresponding scored trajectory amounts to $\phi(\ell_1) + \phi(\ell_2) + \phi(\ell_3)$.

Example 2. Consider now our running example on the visual sensor network in Sect. 2. Every valid estimate $(\widehat{\Sigma}, \kappa, \Theta, T, \rho)$ must include at least the following entries, assuming $d = 4$, and $m \neq i$, with m and i indexes of nodes that are neighbours.

$\Theta(\ell_i^1)(a_{i1}) \supseteq \{1^{a_{i1}}\}$, $\Theta(\ell_{1i})(a_{i2}) \supseteq \{2^{a_{i2}}\}$
$\hat{\Sigma}_{\ell_i^1}(z^{v_{i1}}) \supseteq \{1^{a_{i1}}\}$, $\hat{\Sigma}_{\ell_i^1}(z^{v_{i2}}) \supseteq \{2^{a_{i2}}\}$
$\kappa(\ell_m^1) \supseteq \{(\ell_i^1, \langle\!\langle p(1^{a_{i1}}, 2^{a_{i2}})^{p_i} \rangle\!\rangle\}$
$\rho(X_m^i) \supseteq \{(\ell_i^1, \langle\!\langle p(1^{a_{i1}}, 2^{a_{i2}})^{p_i} \rangle\!\rangle\}$
$\hat{\Sigma}_{\ell_m^1}(x_m^i) \supseteq \{p(1^{a_{i1}}, 2^{a_{i2}})^{p_i}\}$
$\hat{\Sigma}_{\ell_j^2}(confirm_{2j}^{w2j}) =$
$\quad check(d(1^{a_{i1}}, 1^{a_{i2}}, p(1^{a_{r11}}, 2^{a_{r12}})^{p_{r1}}, ..., p(1^{a_{rt1}}, 2^{a_{rt2}})^{p_{rt}})^{d_{1i}}, 1^{d_{2j1}}, 1^{d_{2j2}})^{c2j}$
$T(p_i) \ni ((\ell_i^1, \ell_m^1), X_i^m), ((\ell_m^1, \ell_j^2), W_j^2), ((\ell_j^2, \ell_l), A_j^l)$
$T(d_{1i}) \ni ((\ell_i^1, \ell_j^2), W_j^2), ((\ell_j^2, \ell_l), A_j^l)$

Our analysis respects the operational semantics of IoT-LySA, as witnessed by the following subject reduction result. It is also possible to prove the existence of a (minimal) estimate, as in [5]. The proofs follow the usual schema and benefit from an instrumented denotational semantics for expressions, the values of which are pairs $\langle v, \hat{v} \rangle$, where v is a concrete value and $\hat{v}$ is the corresponding abstract value. The store (Σ_ℓ^i with an undefined $\perp$ value) is accordingly extended. The semantics used in Table 3 just uses the projection on the first component.

The following subject reduction theorem establishes the correctness of our CFA, by relying on the agreement relation $\bowtie$ between the concrete and the abstract stores. Its definition is immediate, since the analysis only considers the second component of the extended store, i.e. the abstract one: $\Sigma_\ell^i \bowtie \widehat{\Sigma}_\ell$ iff $w \in \mathcal{X} \cup \mathcal{I}_\ell$ such that $\Sigma_\ell^i(w) \neq \perp$ implies $(\Sigma_\ell^i(w))_{\downarrow_2} \in \widehat{\Sigma}_\ell(w)$.

Theorem 1 (Subject reduction). *If* $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N$ *and* $N \to N'$ *and* $\forall \Sigma_\ell^i$ *in* N *it is* $\Sigma_\ell^i \bowtie \widehat{\Sigma}_\ell$, *then* $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N'$ *and* $\forall \Sigma_\ell^{i\prime}$ *in* N' *it is* $\Sigma_\ell^{i\prime} \bowtie \widehat{\Sigma}_\ell$.

Checking Trajectories. We now show that by inspecting the results of our CFA, we detect all the possible micro-trajectories of the data produced in the system of nodes that, put together, provide the overall trajectories.

The following corollary of subject reduction shows that we do track the trajectories of IoT data. The first item guarantees that κ and ρ predict all the possible inter-node communications, while the second item shows that our analysis records the micro-trajectory in the T component of each abstract value possibly involved in the communication.

Corollary 1. *Let* $N \xrightarrow{\langle\!\langle v_1,\ldots,v_r \rangle\!\rangle}_{\ell_1,\ell_2,X} N'$ *denote a reduction in which the message sent by node* ℓ_1 *is received by node* ℓ_2 *with an input tagged* X. *If* $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N$ *and* $N \xrightarrow{\langle\!\langle v_1,\ldots,v_r \rangle\!\rangle}_{\ell_1,\ell_2} N'$ *then it holds:*

- $(\ell_1, \langle\!\langle \hat{v}_1, \ldots, \hat{v}_r \rangle\!\rangle) \in \kappa(\ell_2) \wedge (\ell_1, \langle\!\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \rho(X)$, *where* $\hat{v}_i = v_{i\downarrow_2}$.
- $((\ell_1, \ell_2), X) \in T(a)$, *for all* $a \in \mathbf{A}(\hat{v}_i)$, *for all* $i \in [j+1, r]$.

4.1 Proofs

In this subsection, we provide the formal proofs of the results presented above. The reader not interested in the technical details of this formalisation, can safely skip this subsection without compromising the comprehension of the rest of the paper.

We recall that for the proofs, we resort to an instrumented denotational semantics for expressions, the values of which are pairs $\langle v, \hat{v} \rangle$ where v is a concrete value and $\hat{v}$ is the corresponding abstract value, and that the store and its updates are accordingly extended.

Lemma 1 (Congruence). *If* $N \equiv N'$ *then* $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N$ *iff* $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N'$.

Proof. It suffices to inspect the rules for $\equiv$, since associativity and commutativity of $\wedge$ reflects the same properties of both $|$ and $\|$, and to recall that any triple is a valid estimate for $\mathsf{0}$. Note that for the case of iteration, the following definition of limited unfolding suffices $\lfloor \mu h.\, P \rfloor_0 = P\{\mathsf{0}/h\}$ and $\lfloor \mu h.\, P \rfloor_d = P\{\lfloor \mu h.\, P \rfloor_{d-1}/h\}$.

Theorem 1 (Subject reduction). *If* $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N$ *and* $N \to N'$ *and* $\forall \Sigma_\ell^i$ *in* N *it is* $\Sigma_\ell^i \bowtie \widehat{\Sigma}_\ell$, *then* $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models N'$ *and* $\forall \Sigma_\ell^{i\prime}$ *in* N' *it is* $\Sigma_\ell^{i\prime} \bowtie \widehat{\Sigma}_\ell$.

Proof. Our proof is by induction on the shape of the derivation of $N \to N'$ and by cases on the last rule used. In all the cases below we will have that (*) $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell \Sigma^i$, as well as that (**) $(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_\ell B_1$ and B_2, so we will omit mentioning these judgements.

– Case (Multi-com). We assume

$$(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models \ell_1 : [\langle\!\langle v_1, \cdots, v_k \rangle\!\rangle \triangleright L.\,\mathsf{0} \parallel B_1] \mid \ell_2 : [\Sigma_2^i \parallel (E_1, \cdots, E_j; x_{j+1}, \cdots, , x_k)^X.Q \parallel B_2]$$

that is implied by

$$(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_1} [\langle\!\langle v_1, \cdots, v_k \rangle\!\rangle \triangleright L.\,\mathsf{0} \parallel B_1] \text{ and by}$$
$$(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_2} [\Sigma_2^i \parallel (E_1', \cdots; x_{j+1}, \cdots)^X.Q \parallel B_2]$$

that have been proved because the following conditions hold:

$$Comp(\ell_1, \ell_2) \tag{1}$$
$$\bigwedge_{i=1}^{k} (\widehat{\Sigma}, \Theta) \models_{\ell_1} v_i \tag{2}$$
$$\forall \hat{v}_1, \cdots, \hat{v}_k : \bigwedge_{i=1}^{k} \hat{v}_i \in \vartheta_i \Rightarrow$$
$$\forall \ell' \in L : (\ell_1, \langle\!\langle \hat{v}_1, \cdots, \hat{v}_k \rangle\!\rangle) \in \kappa(\ell') \tag{3}$$
$$(\widehat{\Sigma}, \kappa, \Theta) \models_{\ell_1} \mathsf{0} \tag{4}$$
$$\bigwedge_{i=1}^{j} (\widehat{\Sigma}, \Theta) \models_{\ell_2} E_i \tag{5}$$
$$\forall (\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_k \rangle\!\rangle) \in \kappa(\ell_2) : \bigwedge_{i=1}^{j} \hat{v}_i \in \vartheta_i' \Rightarrow \tag{6}$$
$$\bigwedge_{i=j+1}^{k} \hat{v}_i \in \hat{\Sigma}_{\ell_2}(x_i) \tag{7}$$
$$\forall (\ell', \langle\!\langle \hat{v}_1, \cdots, \hat{v}_k \rangle\!\rangle) \in \rho(X) \tag{8}$$
$$\forall a \in \mathbf{A}(\hat{v}_i).((\ell, \ell'), X, w) \in T(a) \tag{9}$$
$$(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_2} Q \tag{10}$$

Note that $\forall i\ (\widehat{\Sigma}, \Theta) \models_{\ell_1} v_i$ implies $\hat{v}_i \in \vartheta_i$, where $\hat{v}_i = (\llbracket v_i \rrbracket^i_{\Sigma^i_{\ell_2}})_{\downarrow_2}$, and that $\ell_2 \in L$ because $N \to N'$. We have to prove that

$$(\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models , \ell_1 : [\langle\!\langle v_1, \cdots, v_k \rangle\!\rangle \triangleright L'.\mathsf{0} \| B_1] \mid \ell_2 : [\Sigma_2^i\{v_{j+1}/x_{j+1}, \cdots, v_k/x_k\} \| Q \| B_2]$$

where $L' = L \setminus \{\ell_2\}$ that, in turn, amounts to prove that

$$(a)\ (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_1} \langle\!\langle v_1, \cdots, v_k \rangle\!\rangle \triangleright L \setminus \{\ell_2\}.\,\mathsf{0} \parallel B_1$$
$$(b)\ (\widehat{\Sigma}, \kappa, \Theta, T, \rho) \models_{\ell_2} \Sigma_2^i\{(v_{j+1}, \hat{v}_{j+1})/x_{j+1}, \cdots\} \parallel Q \parallel B_2$$

We have that (a) holds trivially because of (2–4) (of course $L \setminus \{\ell_2\} \subseteq L$), while (b) holds because of (8). We are left to prove that ${\Sigma^i_{\ell_2}}' \bowtie \hat{\Sigma}_{\ell_2}$. Now, we know that ${\Sigma^i_{\ell_2}}'(y) = \Sigma^i_{\ell_2}(y)$ for all $y \in \mathcal{X}_{\ell_2} \cup \mathcal{I}_{\ell_2}$ such that $y \neq x_i$. The condition $(\Sigma^i_{\ell_2}(x_i))_{\downarrow_2} \in \hat{\Sigma}_{\ell_2}(x_i)$ for all x_i holds because of (7).

– The cases (ParN), (StructN), and (Node) directly follow from the induction hypothesis, and the case (CongrN) from Lemma 1.

Corollary 1. *Let* $N \xrightarrow{\langle\!\langle v_1,\ldots,v_r\rangle\!\rangle}_{\ell_1,\ell_2,X} N'$ *denote a reduction in which the message sent by node* ℓ_1 *is received by node* ℓ_2 *with an input tagged* X*. If* $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models N$ *and* $N \xrightarrow{\langle\!\langle v_1,\ldots,v_r\rangle\!\rangle}_{\ell_1,\ell_2} N'$ *then it holds:*

- $(\ell_1,\langle\!\langle \hat{v}_1,\ldots,\hat{v}_r\rangle\!\rangle) \in \kappa(\ell_2) \wedge (\ell_1,\langle\!\langle \hat{v}_1,\cdots,\hat{v}_r\rangle\!\rangle) \in \rho(X)$*, where* $\hat{v}_i = v_{i\downarrow_2}$*.*
- $((\ell_1,\ell_2),X) \in T(a)$*, for all* $a \in \mathbf{A}(\hat{v}_i)$*, for all* $i \in [j+1,r]$*.*

Proof. By Theorem 1, we have that $(\widehat{\Sigma},\kappa,\Theta) \models N'$, so we proceed by induction on the shape of the derivation of $N \to N'$ and by cases on the last rule used.

- Case (Multi-com). If this rule is applied, than N is in the form

$$(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models \ell_1{:}[\langle\!\langle v_1,\cdots,v_k\rangle\!\rangle \triangleright L.\,\mathbf{0} \parallel B_1] \mid \ell_2{:}[\Sigma_2^i \parallel (E_1,\cdots,E_j;x_{j+1},\cdots,,x_k)^X.Q \parallel B_2]$$

  with $\ell_2 \in L$. Since $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models N$ we have $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models_{\ell_1} \langle\!\langle v_1,\cdots,v_k\rangle\!\rangle \triangleright L.\,\mathbf{0}$ and the required $(\ell_1,\langle\!\langle \hat{v}_1,\cdots,\hat{v}_k\rangle\!\rangle) \in \kappa(\ell_2)$.
  From $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models_{\ell_2} (E_1,\cdots,E_j;x_{j+1},\cdots,,x_k)^X.Q$, we can obtain instead the required $(\ell_1,\langle\!\langle \hat{v}_1,\cdots,\hat{v}_k\rangle\!\rangle) \in \rho(X)$, and $\forall a \in \mathbf{A}(\hat{v}_i).((\ell_1,\ell_2),X,w) \in T(a)$.
- Cases (ParN), (StructN), and (Node) directly follow from the induction hypothesis, and for the other rules the premise is false.

5 Scored Trajectories

We now extend the notion of trajectories by associating a *score* $\phi(\ell)$ to each node with label ℓ, representing some quantitative and logical information: in our case with a measure of the risk of node, in a style reminiscent of [1].

Trajectories can be compared on the basis of their overall score.

We assume that a table of scores is known that associates a score $\phi(\ell_i)$ to each node label ℓ_i. As a consequence we can decorate micro-trajectories with the scores of the nodes involved:

$$((\ell_i,\ell_i'),(\phi(\ell_i),\phi(\ell_j)),X_i),$$

resulting in *scored micro-trajectories*. The corresponding *scored trajectories* can be obtained as follows, by using the suitable extended closure function $\widehat{Clos}_X$.

Definition 4.

- $\forall((\ell,\ell'),(\phi(\ell),\phi(\ell')),X) \in M.\ [((\ell,\ell'),(\phi(\ell),\phi(\ell'),X)] \in \widehat{Clos}_X(M)$*;*
- $\forall[L,((\ell,\ell'),(\phi(\ell),\phi(\ell'),X)],\ [((\ell',\ell''),(\phi(\ell'),\phi(\ell''),X'),L''] \in M.$
  $[L,((\ell,\ell'),(\phi(\ell),\phi(\ell'),X),((\ell',\ell''),(\phi(\ell'),\phi(\ell''),X'),L''] \in \widehat{Clos}_X(M)$*.*

We now need a function to extract the overall cost of each trajectory, given the sequence of crossed nodes.

Definition 5.

- $Cost([((\ell,\ell'),(\phi(\ell),\phi(\ell')),X)]) = \phi(\ell)+\phi(\ell')$;
- $Cost([L,((\ell,\ell'),(\phi(\ell),\phi(\ell')),X),((\ell',\ell''),(\phi(\ell'),\phi(\ell'')),X'),L''])$ $=$ $Cost(L)+\phi(\ell)+\phi(\ell')+Cost(L'')$.

We can now using the enriched trajectories to reason on which of them are more risky.

Example 3. Back to our example, by using the analysis, we can determine some of the possible trajectories of data, e.g. the ones of the term annotated with d_{1i}, i.e. $Trajectories(d_{1i})$ that includes $[(\ell_i^1,\ell_j^2),W_j^2),((\ell_j^2,\ell_l),A_j^l)]$.

This allows us to check which are the nodes the data may pass from, in this case and which are the corresponding inputs. The communication pattern here is admittedly simple in order to illustrate our approach. It is easy to verify that the above CFA results reflect the dynamic behaviour.

Now, given a security score for each node, we can analyse the trajectories of each piece of data of the analysed system, in order to determine the more vulnerable ones. We can also inspect the paths possibly followed by sensible data and also be suspicious about data produced or passed by unreliable nodes. For the sake of simplicity, we can use only two values for ϕ, by partitioning nodes in less (0) or more (1) secure. The less secure nodes are the ones put in an open and public area of the building, whereas the more secure nodes are the ones placed in areas with restricted access. In our scenario, suppose that all the nodes are secure apart from the node $N_{1\ 13}$ that is in an open area. Under these hypotheses, our analysis points out that the data that arrive to alert the node of type 2 in Room1 use a possibly vulnerable trajectory, i.e. $[(\ell_m^1,\ell_i^1),X_i^m),((\ell_i^1,\ell_j^2),W_j^2),((\ell_j^2,\ell_l),A_j^l)]$ with $i=13$ has a cost 1 whereas with $i\neq 13$ the cost is 0. As a consequence, it could be the case to use a videocamera more difficult to tamper, or, alternatively, to add a new video camera in the restricted area.

We could instead classify links and making a similar reasoning, by associating weights to each of them in micro-trajectories, as in $((\ell_1,\ell_2),X,w)$. Given a classification of the "dangerous" links, we can analyse the trajectories of each piece of data and the way they are transmitted. This is particularly crucial in a setting where encryption and other security mechanisms can be costly and power consuming.

Another possibility to exploit our analysis is to detect possible illegal or bad flows from one point to another based on security levels, by investigating our trajectories, along the lines of [4]. Suppose for instance that nodes are classified according to a hierarchy of clearance levels for nodes (encoded in a value), and that a no read-up/no write-down policy is required. A node classified at a high level cannot send (write) any value to a node at a lower level, while the converse is allowed. The constraint can be restricted to least sensible data. In any case, by inspecting the possible trajectories, we can check for the presence or not of micro-trajectories (ℓ_1,ℓ_2), where the corresponding nodes do not respect the policy.

Note that each kind of enrichment of trajectories with quantitative information can be added after the analysis, making its results useful for different purposes.

6 Conclusions

We proposed a data path analysis, based on the CFA of IoT-LySA specification language, for tracking the propagation of data and for identifying their possible trajectories.

The results of CFA can be exploited in an early phase of system design, as a supporting technique. The analysis is quite general because the underlying idea is that its results can be used as a starting point for many different investigations on a given system behaviour. On the one hand, a designer can answer whether the provenance of the data that are processed or stored in a particular node offers sufficient security guarantees, e.g. these data traveled only along nodes that are considered robust. Furthermore, we can also check whether a system respects policies that rule information flows among nodes, by allowing some flows and forbidding others, e.g. data traveled only across nodes with a certain level of clearance. Answering to these questions can give some confidence to designers about the quality of the data managed by the considered system and how much secure are the data which are essential to take critical decisions. By using this information designers can detect the potential vulnerabilities related to the presence of dangerous nodes, and can determine possible solutions and mitigation.

On the other hand, analysing the trajectories may allow discovering patterns in data, e.g. there are pieces of data that always move together or in a similar way, thus, allowing designers to determine possible emerging features of the system behaviour. Furthermore, we can find which are the paths or segments of paths that are more used, and therefore may need special attention and suitable security mechanisms.

An approach close to the present one is that of [10], where Control Flow Analysis is used to over-approximate the behaviour of KLAIM processes and to track how tuple data can move in the network.

Our approach is quite flexible and can be adapted to different purposes, just by enriching the trajectories obtained from the analysis' results with different kinks of quantitative information. We would like to resort to other possible metrics. An interesting option is the one introduced in [1,23], where each node is associated to a value that quantitatively represents the effort (in terms of cost) required by an attacker to compromise the node. This allows the authors to reason on the dependencies among nodes and to identify the minimal set of nodes that must be compromised in order to impair the functionalities of a given target node. In the same paper, further metrics are proposed. In the first case, they suppose that the edge (or perimeter) nodes are easier to be compromised, and the effort becomes higher while moving to inner nodes of the graph. As a consequence, they assign costs to the nodes based on their depth in a given graph.

The second proposed security metric associates as a priority to the nodes that require utmost attention. In the third one, the metrics includes budget considerations, in order to provide a balance between the efforts required by an attacker to compromise critical nodes and the cost required to fix them.

Another future direction of investigation consists in integrating our present analysis with the taint analysis of [8]. In that analysis, data are marked as tainted when sensitive, and are marked as tamperable when coming from places where they can be tampered. The analysis statically predicts how marked data spread across an IoT system.

We further plan to study how to ensure a certain level of quality service of a system even when in the presence of not completely reliable data, by linking our approach to that used in [24,25]. In those paper authors introduce the Quality Calculus that allows defining and reasoning on software components that have a sort of backup plan in case the ideal behaviour fails due to unreliable communication or data.

Finally, since in many IoT system the behaviour of node adapts to their computational context, we aim at extending IoT-LySA with constructs for representing contexts along the lines of [14,15], and to study their security following [6,7].

References

1. Barrère, M., Hankin, C., Nicolaou, N., Eliades, D.G., Parisini, T.: Identifying security-critical cyber-physical components in industrial control systems CoRR abs/1905.04796 (2019). http://arxiv.org/abs/1905.04796
2. Bodei, C., Brodo, L., Focardi, R.: Static evidences for attack reconstruction. In: Bodei, C., Ferrari, G.-L., Priami, C. (eds.) Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 162–182. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25527-9_12
3. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. J. Comput. Secur. **13**(3), 347–390 (2005)
4. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: A step towards checking security in IoT. In: Proceedings of ICE 2016. EPTCS, vol. 223, pp. 128–142 (2016)
5. Bodei, C., Degano, P., Ferrari, G.-L., Galletta, L.: Where do your IoT ingredients come from? In: Lluch Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 35–50. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39519-7_3
6. Bodei, C., Degano, P., Galletta, L., Salvatori, F.: Linguistic mechanisms for context-aware security. In: Ciobanu, G., Méry, D. (eds.) ICTAC 2014. LNCS, vol. 8687, pp. 61–79. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10882-7_5
7. Bodei, C., Degano, P., Galletta, L., Salvatori, F.: Context-aware security: linguistic mechanisms and static analysis. J. Comput. Secur. **24**(4), 427–477 (2016)
8. Bodei, C., Galletta, L.: Tracking sensitive and untrustworthy data in IoT. In: Proceedings of the First Italian Conference on Cybersecurity (ITASEC 2017), pp. 38–52. CEUR Vol-1816 (2017)
9. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: Tracing where IoT data are collected and aggregated. Log. Methods Comput. Sci. **13**(3) (2017)

10. Bodei, C., Degano, P., Ferrari, G.-L., Galletta, L.: Revealing the trajectories of KLAIM tuples, statically. In: Boreale, M., Corradini, F., Loreti, M., Pugliese, R. (eds.) Models, Languages, and Tools for Concurrent and Distributed Programming. LNCS, vol. 11665, pp. 437–454. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21485-2_24
11. Bodei, C., Galletta, L.: Tracking data trajectories in IoT. In: International Conference on Information Systems Security and Privacy (ICISSP2019). Lecture Notes in Computer Science, vol. 1. ScitePress (2019)
12. Chessa, S., Pelagatti, S., Triolo, N.: Engineering energy efficient visual sensor network applications using skeletons. Int. J. Parallel Program. **42**(4), 663–680 (2014). https://doi.org/10.1007/s10766-013-0260-y
13. Concha, Ó.P., Patricio, M.A., Herrero, J.G., Rubiera, J.C., Molina, J.M.: Fusion of surveillance information for visual sensor networks. In: 9th International Conference on Information Fusion, FUSION 2006, pp. 1–8. IEEE (2006)
14. Degano, P., Ferrari, G.L., Galletta, L.: A two-component language for COP. In: Proceedings of 6th International Workshop on Context-Oriented Programming, COP@ECOOP 2014, pp. 6:1–6:7. ACM (2014)
15. Degano, P., Ferrari, G.L., Galletta, L.: A two-component language for adaptation: design, semantics, and program analysis. IEEE Trans. Softw. Eng. **42**(6), 505–529 (2016)
16. Gao, H., Bodei, C., Degano, P.: A formal analysis of complex type flaw attacks on security protocols. In: Meseguer, J., Roşu, G. (eds.) AMAST 2008. LNCS, vol. 5140, pp. 167–183. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79980-1_14
17. Gao, H., Bodei, C., Degano, P., Riis Nielson, H.: A formal analysis for capturing replay attacks in cryptographic protocols. In: Cervesato, I. (ed.) ASIAN 2007. LNCS, vol. 4846, pp. 150–165. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76929-3_15
18. Herlihy, M.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. **13**(1), 124–149 (1991)
19. Lanese, I., Bedogni, L., Felice, M.D.: Internet of Things: a process calculus approach. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC 2013, pp. 1339–1346. ACM (2013)
20. Lanotte, R., Merro, M.: A semantic theory of the Internet of Things. In: Lluch Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 157–174. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39519-7_10
21. Lanotte, R., Merro, M.: A semantic theory of the Internet of Things. Inf. Comput. **259**(1), 72–101 (2018)
22. Lanotte, R., Merro, M., Muradore, R., Viganò, L.: A formal approach to cyber-physical attacks. In: 30th IEEE Computer Security Foundations Symposium, CSF 2017, pp. 436–450 (2017)
23. Nicolaou, N., Eliades, D.G., Panayiotou, C.G., Polycarpou, M.M.: Reducing vulnerability to cyber-physical attacks in water distribution networks. In: 2018 International Workshop on Cyber-physical Systems for Smart Water Networks, CySWater@CPSWeek, pp. 16–19. IEEE Computer Society (2018)
24. Nielson, H.R., Nielson, F., Vigo, R.: A calculus for quality. In: Păsăreanu, C.S., Salaün, G. (eds.) FACS 2012. LNCS, vol. 7684, pp. 188–204. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35861-6_12

25. Nielson, H.R., Nielson, F., Vigo, R.: A calculus of quality for robustness against unreliable communication. J. Log. Algebr. Methods Program. **84**(5), 611–639 (2015)
26. Zillner, T.: ZigBee exploited (2015). https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly-wp.pdf