Tracking sensitive and untrustworthy data in IoT *

Chiara Bodei Letterio Galletta
{chiara,galletta}@di.unipi.it

Dipartimento di Informatica, Università di Pisa

Abstract

The Internet of Things (IoT) produces and processes large amounts of data. Among these data, some must be protected and others must be carefully handled because they come from untrusted sources. Taint analysis techniques can be used to for marking data and for monitoring their propagation at run time, so to determine how they influence the rest of the computation. Starting from the specification language IoT-LySA, we propose a Control Flow Analysis for statically predicting how tainted data spread across an IoT system and for checking whether those computations considered security critical are not affected by tainted data.

1 Introduction

We are living the Internet of Things (IoT) revolution, where the *things* we use every day have computational capabilities and are always connected to the Internet. These devices are equipped with sensors, produce huge amounts of data, store them on the cloud or use them to affect the surrounding environment through actuators.

The IoT paradigm introduces new pressing security challenges. On the one hand, a large part of the collected data is sensitive and must be protected against attackers. However, these data need to be shared and processed to be useful. For instance, a fitness app should store and manipulate data about the heart rate or the expended calories to be helpful for users. On the other hand, an attacker can easily intercept communications or manipulate sensors to alter data (*tampering*). Thus, an IoT system must be resistant against "bad" data that may come from malicious sources or that may be the result of tampering with "good" sensors. For instance, deciding to stop an industrial plant in dependence of data coming from tampered sensors can have severe consequences.

Usually, formal methods provide designers with tools to support the development of systems and to reason about their properties. We follow this line of research by presenting preliminary results about using static analysis to study simple security properties of IoT systems, i.e. networks of nodes, where each node interacts with the environment through sensors and actuators.

Technically, our starting point is the formal specification language IoT-LySA, a process calculus recently proposed for IoT systems [4, 3]. IoT-LySA may help designers to adopt a *Security by Design* development model. Indeed, designers can model the structure of the system and how its components (smart objects) interact with each other through the IoT-LySA primitives. Furthermore, they can reason about the system correctness and robustness by using the Control Flow Analysis (CFA) of IoT-LySA. This analysis safely approximates the system behaviour, by statically predicting how data from sensors may spread across the system and how objects may interact with each other. Technically, it "mimics" the evolution of the system, by using abstract values in place of concrete ones and by modelling the consequences of each

*Partially supported by Università di Pisa PRA_2016_64 Project *Through the fog*.

possible action. Designers can detect possible security flaws through this "abstract simulation" and intervene as early as possible during the design phase.

Here, we propose a variant of this CFA for *static taint analysis*. Taint analysis predicts how information flows from specific data sources to the computations that use these data [11, 10]. The basic idea is to classify data sources in *tainted* and *untainted* and to mark as tainted or untainted the derived data, through suitable propagation rules.

In order to detect possible confidentiality or privacy leaks, we mark as tainted data coming from *sensitive* sensors and check whether they are protected when in transit. Otherwise, we mark as *tamperable* data coming from sensors or memory locations that can be tampered and check which computations they affect, thus addressing integrity issues. Furthermore, we require designers (1) to identify some parts of the code as critical (*critical points*), e.g. a conditional statement whose result may trigger a risky actuation; and (2) to provide a set of functions that untaint data. In the case of sensitive data these functions remove privacy-critical information, e.g. by blurring people faces in surveillance videos. In the case of tamperable data these functions perform sanitisation. By exploiting the result of our analysis, we check whether (i) a variable is untrustworthy, because affected by possibly tampered data; (ii) sensitive data may be leaked; and (iii) computations in critical points depend on tamperable data. As a consequence, analysis results may help designers in making educated decisions, on the exposure of their data, as well as on their trustfulness and robustness. As far as the last point is concerned, they can indeed statically evaluate the impact of a certain amount of tamperable data, e.g. by answering to question like: how many sensors can be compromised without dramatically impacting on the overall behaviour of the system? Note that tamperable data may also produce, besides security issues, quality ones, such as unexpected behaviours.

Since the CFA computes an over-approximation of the behaviour of a system, if the analysis does not predict any dependence or any leak, we can be sure that at run time it will never occur. If instead the CFA does, there is only the possibility of the violation. Nevertheless, it can be worthwhile to track where the tainted values may come from and decide the points that deserve to be monitored at run time in order to prevent violations.

We assume that the processes running inside a node are protected against tampering. This can be obtained by resorting to standard mechanisms for integrity. However, we assume that an attacker can tamper with every sensor or memory location considered tamperable. When we focus on confidentiality, we assume instead that communication is not protected against eavesdropping, but that the attacker does not intervene in communications. Dealing with an attacker able to inject forged messages can be managed by following [2]. Finally, we assume to use perfect cryptography and to have keys, fixed once and for all, exchanged in a secure manner at deployment time, as it is often the case, e.g. ZigBee [12].

The paper is organised as follows. In Section 2 we introduce a motivating example, which is also instrumental in giving an overall picture of our methodology. In Section 3 we briefly recall the process calculus IoT-LySa as introduced in [4, 3]. Then, we define the CFA for static taint analysis in Section 4, and we also show how to check the security properties described above. The Appendix includes part of the technical details of our formalism.

2 A storehouse with perishable food

In this section we illustrate our methodology through a scenario similar to the one of [5]. Consider an IoT system for keeping the temperature under control inside a storehouse storing perishable food (see Figure 1). The temperature must be regulated depending on the quantity and the kind of the food. The system also monitors how long the food is kept inside the

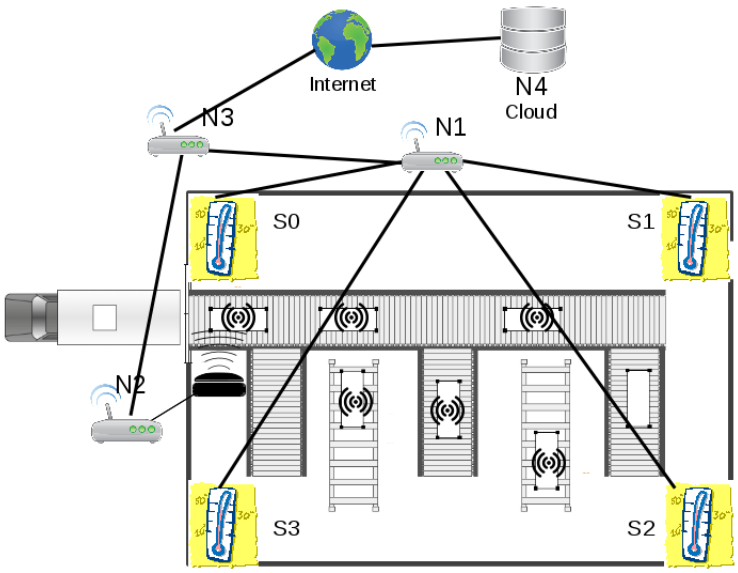

Figure 1: The organisation of nodes in the IoT system of a storehouse with perishable food.

storehouse to avoid passing the expiry-date. In the storehouse there are four nodes: N_1, N_2, N_3 and N_4. The specification of the node N_1 in IoT-LySA is as follows:

$$N_1 = \ell_1 : [\Sigma_1 \parallel P_c \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \qquad \text{where } S_i = \mu h.i := v.\tau.h \qquad i \in [0,3]$$

$$P_c = \mu h.z_0 := 0.z_1 := 1.z_2 := 2.z_3 := 3.\langle\!\langle avg(z_0,z_1,z_2,z_3)\rangle\!\rangle \triangleright \{\ell_3\}.$$
$$th_1 - th_3 \leq avg(z_0,z_1,z_2,z_3) \leq th_2 + th_3 \ ?$$
$$th_1 \leq avg(z_0,z_1,z_2,z_3) \leq th_2 \ ? \ h$$
$$: \langle j, start\rangle.\, h$$
$$: \langle\!\langle \texttt{alarm} \rangle\!\rangle \triangleright \{\ell_3\}.h$$

The node N_1, uniquely identified in the system by the label ℓ_1, is made of P_c, a software control process that specifies the logic of the node, and four temperature sensors S_i, one for each corner of the storehouse (a rectangular room). Each sensor S_i periodically senses the environment and sends the temperature through a wireless communication to P_c. In IoT-LySA each sensor inside a node is identified by an index i and communicates the value v read from the environment by storing it in the corresponding reserved location i of the shared store Σ_1. In IoT-LySA intra-node communications are abstractly modelled through operations over the shared store. The action τ denotes internal actions, which we are not interested in modelling, e.g. noise reduction of sensed data; the construct $\mu h.$ implements the iterative behaviour of the sensor. The control process P_c reads temperatures from sensors S_i and stores them to local variables z_i through the assignments $z_i := i$ (with i identifier of the sensor S_i); then, it computes the average through the function avg and sends it to the node N_3 with label ℓ_3 through the primitive $\langle\!\langle \_ \rangle\!\rangle \triangleright \_$. Then, P_c checks if this average is within the range $[th_1, th_2]$ (th_1 and th_2 are the threshold variables and the primitive $\_?\_ : \_$ is a conditional). If this is not the case and the temperature is out of range of a value lower than th_3 (tolerance threshold), the actuator j is started through the

message $\langle j, start \rangle$ to accordingly turn on/off the cooling system. If the difference is too high (greater than th_3), the control unit sends instead an `alarm` to the node N_3.

The node N_2 with label ℓ_2 does the stocktaking and sends it to the node N_3. We assume that each wood box containing food is equipped with a RFID read by the reader R_0 (0 is the index of the sensor) of N_2 when the box enters the storehouse. The specification of N_2 is:

$$N_2 = \ell_2 : [\Sigma_2 \parallel \mu h.x := 0.db := update(db, x).\tau.h \parallel \mu h.\langle\!\langle db \rangle\!\rangle \triangleright \{\ell_3\}.h \parallel R_0]$$

The node is made of two processes. The first reads a value from R_0 (defined as the sensors S_i above) and updates the stocktaking db that the second one periodically sends to N_3.

The node N_3 with label ℓ_3 works as the system controller and as gateway towards the Cloud (i.e. the node N_4 with label ℓ_4). It receives the stocktaking and the temperature, and checks whether the temperature is acceptable for the quantity and the kind of the stored food. If this is not the case it drives N_1 to take the proper actions and raises an alarm through the Cloud. The specification of the process P_k of N_3 performing these checks is as follows:

$$P_k = \mu h.temp \notin validRange(db) \ ? \ \langle\!\langle \texttt{alarm} \rangle\!\rangle \triangleright \{\ell_4\}.\langle\!\langle validRange(db) \rangle\!\rangle \triangleright \{\ell_1\}.h : h$$

where *validRange*, given the current content of the storehouse, returns a range of admissible temperatures to be maintained inside the room.

Since an attacker may manipulate data from sensors and also the sensors themselves, we consider sensors as tamperable data sources, and we tag with the taint label for tamperable data ⟠ the locations of Σ_1 reserved for the sensors S_i of N_1. Similarly, we tag the location of Σ_2 reserved for R_0 in N_2. As discussed above, N_3 controls the system based on data received by others nodes. Thus, we mark the code checking if the temperature is right (the guard of the conditional of P_k) as a critical point.

However, our CFA can detect that our system is not robust against data manipulations. Indeed, an attacker by manipulating the sensors S_i and R_0 can interfere or drive the decision of N_3 and may damage the system, because the computations in the critical point directly depend on these data. This is because the nodes N_1 and N_2 simply forward their data to N_3 without performing any sanitisation. In particular, the analysis propagates the tag ⟠ and detects that local variables z_i inside N_1 are also tamperable, and so it is the computed average that is sent to N_3. The same holds for the stocktaking computed by N_1.

To solve this problem we need to modify our design to perform data sanitisation. The CFA can help us during this process because we can try different approaches and test them through the analysis (*what if* reasoning). For instance, assume that the attacker can tamper with only one sensor, e.g. S_0. A possible approach to deal with this fact is to modify the behaviour of N_1 by observing that sensors on the same side of the room should perceive the same temperature with a difference that can be at most a given value ϵ. The same happens when we consider the difference between two consecutive not manipulated samples: this difference is at most a given value δ. In this way we may compare data and discover possible manipulations and discard possibly tampered data. These checks can be implemented in N_1 specification through a function *adjust* that returns the temperature read from a sensor, adjusted taking into account the previous samples and temperature of the adjacent sensor. In the specification of P_c the assignment to the variable z_0 would become $z_0 = adjust(0, 3, s_0)$ where s_0 contains the previous samples of the sensor S_0. The same happens for the other variables. Thus, the function *adjust* sanitises the input and does not propagate taint information, i.e. it never returns a value with tag ⟠. By using the CFA we can be sure that N_1 sends no tainted data to the node N_3.

3 The calculus IoT-LySA

Here, we briefly review the process calculus IoT-LySA [4, 3]. Differently from other process calculus approches to IoT, e.g. [8, 9], the focus of IoT-LySA is on the development of a design framework that includes a static semantics to support verification techniques and tools for certifying properties of IoT applications.

Systems in IoT-LySA have a two-level structure and consist of a finite number of nodes (things), each of which hosts a store for internal communication and a finite number of control processes (representing the software), sensors and actuators. 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). Data are represented by terms. Labels $a, a', a_i, ...$, ranged over by $\mathcal{A}$, identify the occurrences of terms. They are used in the analysis and do not affect the dynamic semantics. Formally, the syntax of labelled expression and the one of *unlabelled* terms are as follows.

$\mathcal{E} \ni E$	::= *labelled terms*		$\mathcal{M} \ni M, N$	::= *terms*	
	M^a	labelled 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

We assume as given a finite set $\mathcal{K}$ of secret keys owned by nodes, previously exchanged in a secure way, as it is often the case [12]. 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. 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.

The syntax of systems of nodes and of its components is as follows.

$\mathcal{N} \ni N$ ::=	*systems of nodes*		$\mathcal{B} \ni B$ ::=	*node components*	
	$\mathsf{0}$	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

A node $\ell : [B]$ is uniquely identified by a label $\ell \in \mathcal{L}$ that may represent further characterising information (e.g. node location). Sets of nodes are described through the (associative and commutative) operator $\mid$ for parallel composition. The $\mathsf{0}$ denotes a system with no nodes. Inside a node $\ell : [B]$ there is a finite set of components described by the parallel operator $\parallel$. We impose that there is a *single* store $\Sigma_\ell : \mathcal{X} \cup \mathcal{I}_\ell \to \mathcal{V}$, 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 [7]. 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 syntax of processes is as follows.

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

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 $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_r)$ 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. A process repeats its behaviour, when defined through the tail iteration construct $\mu h.P$ (h is the iteration variable). 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 we use the pattern matching. If the pattern matching succeeds, the process continues as P and the variables $x_{j+1}, \ldots, x_r$ are suitably assigned. Sensors and actuators have the form:

S	$::=$ *sensors*		A	$::=$ *actuators*	
	$\mathsf{0}$	inactive sensor		$\mathsf{0}$	inactive actuator
	$\tau.S$	internal action		$\tau.A$	internal action
	$i := v.\, S$	store of $v \in \mathcal{V}$		$(\lvert j, \Gamma \rvert).\, A$	command for actuator j ($\Gamma \subseteq Act$)
		by the i^{th} sensor		$\gamma.A$	triggered action ($\gamma \in Act$)
	h	iteration var.		h	iteration var.
	$\mu\, h\, .\, S$	tail iteration		$\mu\, h\, .\, A$	tail iteration

A sensor can perform an internal action τ or store 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 action γ, possibly received from its controlling process. Both 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.

The semantics is based on a standard structural congruence and a two-level *reduction relation* $\rightarrow$ defined as the least relation on nodes and its components, where we assume the standard denotational interpretation $[\![E]\!]_\Sigma$ for evaluating terms. The complete semantic is in Appendix. The reader not interested in the technical details can safely skip it without compromising the comprehension of the rest of the paper. As examples of semantic rules, we show the rules (Ev-out) and (Multi-com) that drive asynchronous multi-communications.

(Ev-out)

$$\frac{\bigwedge_{i=1}^{r} v_i = [\![E_i]\!]_\Sigma}{\Sigma \parallel \langle\langle E_1, \cdots, E_r \rangle\rangle \triangleright L.\, P \parallel B \ \rightarrow\ \Sigma \parallel \langle\langle v_1, \cdots, v_r \rangle\rangle \triangleright L.\mathsf{0} \parallel P \parallel B}$$

(Multi-com)

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

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 ($L = \varnothing$).

4 Control flow analysis

Here we present a CFA for static taint analysis to track the propagation of sensitive data and data coming from possibly tampered sensors or variables. This CFA follows the same schema of the one in [4, 3] for IoT-LySA. However, here we use different abstract values and propagation rules. Intuitively, abstract values "symbolically" represent runtime data so as to encode whether these data are tainted or not. Furthermore, we define functions over abstract values encoding how information about the taint is propagated to derived data. Finally, we show how to use the CFA results to check whether data from tamperable sources affect variables and computations considered security critical. We also check whether sensitive data are leaked.

Taint information To formally introduce our abstract values and operators through which we can combine and manipulate them, we first define the set of *taint labels* $\mathcal{B}$ (ranged over by $b, b_1, ...$) whose elements (the colours in the pdf should help the intuition) are:

$\diamondsuit$ untainted $\quad$ $\diamondsuit$ sensitive $\quad$ $\diamondsuit$ tamperable $\quad$ $\diamondsuit$ sensitive & tamperable

The idea is that these labels mark our abstract values with information about their sources. Formally, abstract values are pairs in $\hat{\mathcal{V}}$ defined as follows, where $b \in \mathcal{B}$.

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

For simplicity, hereafter we write them as $\top^b, \nu^b, \{\hat{v}_1, \cdots, \hat{v}_n\}^b_{k_0}$, 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 ν^b, ν abstracts the concrete value from sensors or computed by a function in the concrete semantics, while the first value of the pair $\{\hat{v}_1, \cdots, \hat{v}_n\}^b_{k_0}$ abstracts encrypted data. Since the dynamic semantics may introduce encrypted terms with an arbitrarily nesting level, we have the special abstract values $\top^b$ 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 $\lfloor - \rfloor_d$, defined as expected (see Appendix). Note that once given the set of encryption functions occurring in a node N, the abstract values are finitely many.

We assume that designers provide the analysis with a classification of the data sources, i.e. the set of *sensitive sensors* $\mathcal{S}_\ell$, and the set of the *tamperable sources (sensors and variables)* $\mathcal{T}_\ell$, for each node with label ℓ.

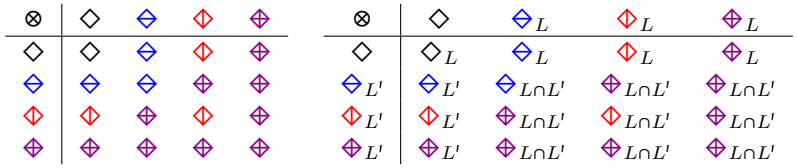

Table 1: The operator $\otimes$, version 1 (on the left) and version 2 (on the right)

Definition 4.1 (Data classification)**.** *Given the set of* sensitive sensors $\mathcal{S}_\ell$*, and the set of the* tamperable sensors and variables $\mathcal{T}_\ell$*, the* taint assignment function τ *is defined as follows:*

$$\tau(y,\ell) = \begin{cases} \diamondsuit & \text{if } y \in \mathcal{S}_\ell \\ \diamondsuit & \text{if } y \in \mathcal{T}_\ell \\ \diamondsuit & \text{if } y \in \mathcal{S}_\ell \cap \mathcal{T}_\ell \\ \diamondsuit & o.w. \end{cases} \qquad \text{where } y \in \mathcal{I}_\ell \cup \mathcal{X} \qquad \tau(v,\ell) = \diamondsuit$$

The above function classifies only the data source; to propagate the taint information we resort to the *taint combination operator* $\otimes : \mathcal{B} \times \mathcal{B} \to \mathcal{B}$ defined in Table 1, on the left part. Note that $\otimes$ works as a join operator making our abstract values a lattice similar to the 4-point lattice LL,LH,HL,HH, used in the information flow literature, which combines confidentiality and integrity. This operator naturally extends to abstract values: $b \otimes \hat{v} = \hat{v}_{\downarrow_1}^{b'}$ where $\hat{v} \in \hat{\mathcal{V}}$ and $b' = b \otimes \hat{v}_{\downarrow_2}$; and to sets of abstract values: $b \otimes \hat{V} = \{b \otimes \hat{v} | \hat{v} \in \hat{V} \subseteq \hat{\mathcal{V}}\}$.

Now we need to specify how taint information propagate with aggregation functions. Below we define two simple propagation policies for function applications and encryptions. In the case of function application, the idea is that a single tainted argument turns to tainted the result of the application. Encryption protects its sensitive data, but preserves tamperable taint information. This means that encryption as a structure is marked with label $\diamondsuit$, unless one of its component is marked with $\diamondsuit$ or $\diamondsuit$; in these cases, encryption is labelled with $\diamondsuit$.

Definition 4.2 (Taint propagation policies)**.** *Given the combination operator* $\otimes : \mathcal{B} \times \mathcal{B} \to \mathcal{B}$*, the taint resulting by the application of*

- *a function* f *is* $F_\tau(f, \hat{v}_1, \ldots, \hat{v}_r) = \otimes(\hat{v}_{1\downarrow_2}, \ldots, \hat{v}_{r\downarrow_2})$
- *an encryption function is* $Enc_\tau(\hat{v}_1, \ldots, \hat{v}_r) = \begin{cases} \diamondsuit & \text{if } \forall i.\hat{v}_{i\downarrow_2} \in \{\diamondsuit, \diamondsuit\} \\ \diamondsuit & o.w. \end{cases}$

It is obviously possible to consider different policies for propagation, e.g. we could consider a set of functions that produce sensitive data independently from the taint information of its arguments. Moreover, we could deal with anonymisation functions that process their arguments to remove sensitive information. Consider e.g. blurring the faces of people in surveillance videos to protect their privacy. A policy for these functions is similar to the one for the encryption: the resulting taint label is $\diamondsuit$, if no argument is also tamperable, otherwise is $\diamondsuit$.

Finally, we could extend the whole presented schema to keep also the source nodes of tainted information. To do that, we could introduce in our abstract values also the labels of nodes from which the taint information derives. To deal with this new information, the taint assignment function τ returns pairs (b, L), written b_L (b_ℓ when $L = \{\ell\}$), our combinator operator becomes the one shown on the right part in Table 1, and the encryption operator takes ℓ as argument and uses it to annotate the resulting taint label.

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 taint contents of messages and values. 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 triple $(\hat{\Sigma}, \kappa, \Theta)$ (a pair $(\hat{\Sigma}, \Theta)$ when analysing a term), called *estimate* for N (for E), where $\hat{\Sigma}$, κ, and Θ are the following *abstract domains*:

- the union $\hat{\Sigma} = \bigcup_{\ell \in \mathcal{L}} \hat{\Sigma}_\ell$ of the super-sets $\hat{\Sigma}_\ell : \mathcal{X} \cup \mathcal{I}_\ell \to 2^{\widehat{\mathcal{V}}}$ of abstract values that may possibly be associated to a given location in $\mathcal{I}_\ell$ or a given variable in $\mathcal{X}$,
- a super-set $\kappa : \mathcal{L} \to \mathcal{L} \times \bigcup_{i=1}^{k} \widehat{\mathcal{V}}^i$ of the messages that may be received by the node ℓ, and
- a super-set $\Theta : \mathcal{L} \to \mathcal{A} \to 2^{\widehat{\mathcal{V}}}$ of the taint information of the actual values computed by each labelled term M^a in a given node ℓ, at run time.

Once a proposed estimate is available, its correctness has to be validated. This requires that it satisfies the judgements defined on the syntax of nodes, node components and terms. They are defined by a set of clauses, fully presented in Appendix (see Tables 3 and 4). Here, we show some examples. The judgements for labelled terms have the form $(\hat{\Sigma}, \Theta) \vDash_\ell M^a$. For each term M^a occurring in the node ℓ, the corresponding judgement requires that $\Theta(\ell)(a)$ includes all the abstract values $\hat{v}$ associated to M^a. Consider, e.g. the following clause for the variable x^a.

$$\frac{\tau(x, \ell) \otimes \hat{\Sigma}_\ell(x) \subseteq \Theta(\ell)(a)}{(\hat{\Sigma}, \Theta) \vDash_\ell x^a}$$

In this case, an estimate is valid if $\Theta(\ell)(a)$ includes the abstract values resulting from the combination (through the operator $\otimes$) of the taint information associated to the variable x (via $\tau(x, \ell)$), and the abstract values bound to x in $\hat{\Sigma}_\ell$. This combination allows us to propagate the tamperable taint if x belongs to the set of tamperable variables.

The judgements for nodes have the form $(\hat{\Sigma}, \kappa, \Theta) \vDash N$. The rule for a single node $\ell : [B]$ requires that its internal components B are analysed with judgements $(\hat{\Sigma}, \kappa, \Theta) \vDash_\ell B$. As examples of clauses, we consider the clauses for communication.

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\hat{\Sigma}, \Theta) \vDash_\ell M_i^{a_i} \ \wedge \ (\hat{\Sigma}, \kappa, \Theta) \vDash_\ell P \ \wedge \\ \forall \hat{v}_1, \cdots, \hat{v}_r : \ \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \ \Rightarrow \forall \ell' \in L : (\ell, \langle\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\rangle) \in \kappa(\ell')\end{array}}{(\hat{\Sigma}, \kappa, \Theta) \vDash_\ell \langle\langle M_1^{a_1}, \cdots, M_r^{a_r} \rangle\rangle \rhd L.\, P}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{j} (\hat{\Sigma}, \Theta) \vDash_\ell M_i^{a_i} \ \wedge \\ \forall (\ell', \langle\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\rangle) \in \kappa(\ell) : \ Comp(\ell', \ell) \Rightarrow \left(\bigwedge_{i=j+1}^{r} \hat{v}_i \otimes \tau(x_i, \ell) \in \hat{\Sigma}_\ell(x_i) \ \wedge \ (\hat{\Sigma}, \kappa, \Theta) \vDash_\ell P \right)\end{array}}{(\hat{\Sigma}, \kappa, \Theta) \vDash_\ell (M_1^{a_1}, \cdots, M_j^{a_j};\, x_{j+1}^{a_{j+1}}, \cdots, x_r^{a_r}).\, P}$$

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 if 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 it 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 $\kappa(\ell)$ and $Comp(\ell', \ell)$, $\hat{v}_{j+1}, \ldots, \hat{v}_r$ are included and combined with the estimates of $x_{j+1}, \cdots, x_r$.

Example 4.3. *Consider again our example of the storehouse in Sect. 2. A valid estimate* $(\hat{\Sigma}, \kappa, \Theta)$ *must include the following entries, where* $\tau(z_0, \ell) = \diamondsuit$, $\nu^{F_\tau(avg,\nu^\diamondsuit,\nu^\diamondsuit,\nu^\diamondsuit,\nu^\diamondsuit)} = \nu^\diamondsuit$, *and* Avg, *and* Db *are the labels of the terms* $avg(z_0, z_1, z_2, z_3)$ *and* db.

$$\hat{\Sigma}_{\ell_1}(z_0) \supseteq \{\nu^\diamondsuit\} \quad \hat{\Sigma}_{\ell_2}(x) \supseteq \{\nu^\diamondsuit\} \quad \hat{\Sigma}_{\ell_2}(db) \supseteq \{\nu^\diamondsuit\} \quad \hat{\Sigma}_{\ell_3}(temp) \supseteq \{\nu^\diamondsuit\}$$
$$\Theta(\ell_1)(\text{Avg}) \supseteq \{\nu^\diamondsuit\} \quad \Theta(\ell_2)(\text{Db}) \supseteq \{\nu^\diamondsuit\} \quad \kappa(\ell_3) \supseteq \{(\ell_1, \nu^\diamondsuit)\} \quad \kappa(\ell_3) \supseteq \{(\ell_2, \nu^\diamondsuit)\}$$

Indeed, an estimate must satisfy the checks of the CFA rules. For instance, the validation of P_c *requires, in particular, that* $\nu^\diamondsuit$ *is in* $\hat{\Sigma}_{\ell_1}(z_0)$ *for the rule for variables,* $(\ell_1, \nu^\diamondsuit) \in \kappa(\ell_3)$ *for the rule for output, and* $\nu^\diamondsuit \in \Theta(\ell_1)(\text{Avg})$ *for the rule of functions (see Appendix).*

Our analysis respects the operational semantics of IoT-LySa. As usual, we prove a subject reduction result for our analysis. 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 formal definition can be found in Appendix. The store (Σ^i_ℓ with an undefined $\perp$ value) is accordingly extended. Of course, we compute the second component of the pair by using the operators introduced in Definition 4.2, while, the semantics used in Table 2 uses the projection on the first component.

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

Theorem 4.4 (Subject reduction)**.** *If* $(\hat{\Sigma}, \kappa, \Theta) \vDash N$ *and* $N \rightarrow N'$ *and* $\forall \Sigma^i_\ell$ *in* N *it is* $\Sigma^i_\ell \bowtie \hat{\Sigma}_\ell$, *then* $(\hat{\Sigma}, \kappa, \Theta) \vDash N'$ *and* $\forall \Sigma^{i'}_\ell$ *in* N' *it is* $\Sigma^{i'}_\ell \bowtie \hat{\Sigma}_\ell$.

Checking taint We now show that by inspecting the results of our CFA, we detect whether a variable receives a value coming from a tamperable source, and hence is not trustworthy. Then, we rephrase the confidentiality property of [3] in terms of propagation of sensitive contents, in order to prevent their leaks. Finally, we check when a computation, considered security critical by designers, depends on a tamperable source (a generalisation of the first property).

In the following, we denote with $N \xrightarrow{M_1^{a_1},\ldots,M_r^{a_r}}_\ell N'$ when all the terms $M_i^{a_i}$ are evaluated inside node ℓ, and with $N \xrightarrow{\langle\!\langle v_1,\ldots,v_r \rangle\!\rangle}_{\ell_1,\ell_2} N'$ when the message $\langle\!\langle v_1, \ldots, v_r \rangle\!\rangle$ is sent from the node ℓ_1 to the node ℓ_2.

First we characterise when a variable x of a node ℓ is *untrustworthy*, i.e. when it stores a value propagated from a tamperable source, and therefore with taint label in $\{\diamondsuit, \oplus\}$. To statically check this property we can simply inspect the abstract store $\hat{\Sigma}$ and the taint labels of the corresponding abstract values.

Definition 4.5. *Let* N *be a system of nodes with labels in* $\mathcal{L}$, *and* $\mathcal{T} = \{\mathcal{T}_\ell \mid \ell \in \mathcal{L}\}$ *be the set of tamperable sources. Then, the variable* x *of a node* $\ell \in \mathcal{L}$ is untrustworthy *w.r.t.* $\mathcal{T}$, *if for all derivatives* N' *s.t.* $N \rightarrow^* N'$ *it holds that* $\Sigma^{\prime i}_\ell(x)\downarrow_2 \in \{\diamondsuit, \oplus\}$ *where* $\Sigma^{\prime i}_\ell$ *is the store of* ℓ *in* N'.

Theorem 4.6. *Let* N *be a system of nodes with labels in* $\mathcal{L}$, *and let* $\mathcal{T} = \{\mathcal{T}_\ell \mid \ell \in \mathcal{L}\}$ *the set of tamperable sources. Then a variable* x *of the node* ℓ *is untrustworthy w.r.t.* $\mathcal{T}$ *if* $(\hat{\Sigma}, \kappa, \Theta) \vDash N$, *and* $\hat{\Sigma}_\ell(x)_{\downarrow_2} \subseteq \{\diamondsuit, \oplus\}$.

We define the confidentiality property in terms of sensitive taint information. There are *no leaks* when messages do *not* expose values with taint label in $\{\ominus, \oplus\}$. We statically verify it, by inspecting the labels of the corresponding abstract values in the component κ.

Definition 4.7. *Let N be a system of nodes with labels in $\mathcal{L}$, and $\mathcal{S} = \{\mathcal{S}_\ell \mid \ell \in \mathcal{L}\}$ the set of its sensitive sensors. Then N has* no leaks *w.r.t. $\mathcal{S}$ if $N \rightarrow^* N'$ and, for all $\ell_1, \ell_2 \in \mathcal{L}$, there is no transition $N' \xrightarrow{\langle\!\langle v_1,\ldots,v_n \rangle\!\rangle}_{\ell_1,\ell_2} N''$ such that $v_{i\downarrow_2} \in \{\diamondsuit, \diamondsuit\}$ for some i.*

Theorem 4.8. *Let N be a system of nodes with labels in $\mathcal{L}$, and $\mathcal{S} = \{\mathcal{S}_\ell \mid \ell \in \mathcal{L}\}$ the set of its sensitive sensors. Then N has no leaks w.r.t. $\mathcal{S}$ if $(\hat{\Sigma}, \kappa, \Theta) \vDash N$, and $\forall \ell_1, \ell_2 \in \mathcal{L}$ such that $(\ell_2, \langle\!\langle \hat{v}_1, \cdots, \hat{v}_r \rangle\!\rangle) \in \kappa(\ell_2)$ we have that $\forall i.\ \hat{v}_{i\downarrow_2} \in \{\diamondsuit, \diamondsuit\}$.*

The last property characterises when computations considered security critical are not directly or indirectly affected by tainted data i.e. they are reached by untrustworthy data. We assume that the designer identify a set $\mathcal{P} \subseteq \mathcal{A}$ of *critical points* in the application, i.e. points where possibly tainted data should flow, unless they are checked for validity. We statically verify this property, by inspecting the taint labels of the values in Θ for each critical point.

Definition 4.9. *Let N be a system of nodes with labels in $\mathcal{L}$, $\mathcal{S} = \{\mathcal{S}_\ell \mid \ell \in \mathcal{L}\}$ the set of its sensitive sensors, $\mathcal{T} = \{\mathcal{T}_\ell \mid \ell \in \mathcal{L}\}$ the set of its tamperable sources, and $\mathcal{P}$ a set of program critical points. Then N* does not use tainted values in a critical point *if $N \rightarrow^* N'$ and there is no transition $N' \xrightarrow{M_1^{a_1},\ldots,M_r^{a_r}}_\ell N''$ s.t. $a_i \in \mathcal{P}$ and $([\![M_i^{a_i}]\!]_{\Sigma_\ell^i})_{\downarrow_2} \subseteq \{\diamondsuit, \diamondsuit, \diamondsuit\}$ for some $i \in \{1,\ldots,r\}$ and $\ell \in \mathcal{L}$.*

Theorem 4.10. *Let N be a system of nodes with labels in $\mathcal{L}$, $\mathcal{S} = \{\mathcal{S}_\ell \mid \ell \in \mathcal{L}\}$ the set of its sensitive sensors, $\mathcal{T} = \{\mathcal{T}_\ell \mid \ell \in \mathcal{L}\}$ the set of its tamperable sources, and $\mathcal{P}$ a set of program critical points. Then N does not use tainted values in a program critical point if $(\hat{\Sigma}, \kappa, \Theta) \vDash N$, and $\Theta(\ell)(a)_{\downarrow_2} = \{\diamondsuit\}$ for all labels $a \in \mathcal{P}$ and $\ell \in \mathcal{L}$.*

Example 4.11. *Back to our example, consider the critical point $(temp \notin validRange(db))^a$ in the process P_k. Our CFA detects that this decision is based on possibly tainted values, since $\Theta(\ell_3)(a)_{\downarrow_2} = \{\diamondsuit\}$. More in detail, the analysis of the condition depends on the analysis of temp and on the one of $validRange(db)$, which in turn depends on the one of db (temp and db are both untrustworthy, see Ex. 4.3). By using the labelled version of abstract values and the operator $\otimes$, as defined on the right part in Table 1, we can determine where the possibly tampered data originally come from, i.e. which are the "bad sinks". In our example, we would obtain ℓ_1 and ℓ_2, because $\hat{\Sigma}_{\ell_3}(temp) \supseteq \{\nu^{\diamondsuit_{\ell_1}}\}$ and $\hat{\Sigma}_{\ell_2}(db) \supseteq \{\nu^{\diamondsuit_{\ell_2}}\}$. Similarly, we can trace back possible leakages.*

5 Conclusions

Based on IoT-LySA, we developed a CFA for static taint analysis to track the propagation of sensitive data and data coming from possibly tampered sensors or variables, as illustrated by a motivating example that provides a simple but non-trivial application of our approach. Taint tracking is a relevant issue in security and can be used for checking both confidentiality and integrity of data (see [6, 11, 10] to cite only a few). The idea underlying our approach requires that the designer classify data sources as untainted, sensitive, tamperable. Static analysis has the advantage of giving hints on propagation of tainted data in a early phase of system design, thus guiding designers to understand the potential vulnerabilities and to direct their efforts towards the suitable modifications and validity checks. It is not meant to being a substitute of dynamic checks, but only as a supporting technique.

Here we relied on CFA approximations for checking various security properties about data propagation. In particular, we could detect if a variable is untrustworthy, because affected

by possibly tampered data; if sensitive data are leaked; and if computations in critical point depend on tamperable data. Our CFA could also provide a prescriptive usage (along the lines of [1]), by embedding taints in data and forcing some data to be accepted only if they come with the expected taints.

In future we would like to extend our analysis to also deal with *implicit flows* of tainted data and to understand the relationships with the properties based on implicit flow (see [10]).

References

[1] C. Bodei, L. Brodo, P. Degano, and H. Gao. Detecting and preventing type flaws at static time. *Journal of Computer Security*, 18(2):229–264, 2010.

[2] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[3] C. Bodei, P. Degano, G-L. Ferrari, and L. Galletta. A step towards checking security in IoT. In *Procs. of ICE 2016*, volume 223 of *EPTCS*, pages 128–142, 2016.

[4] C. Bodei, P. Degano, G-L. Ferrari, and L. Galletta. Where do your IoT ingredients come from? In *Procs. of Coordination 2016*, volume 9686 of *LNCS*, pages 35–50. Springer, 2016.

[5] C. Bodei and L. Galletta. The cost of securing IoT communications. In *Procs.of Italian Conference on Theoretical Computer Science*, CEUR Proceedings Vol-1720, 2016.

[6] W. Enck, P. Gilbert, S. Han, Vasant Tendulkar, Byung-Gon Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Trans. Comput. Syst.*, 32(2):5:1–5:29, 2014.

[7] M. Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1), 1991.

[8] I. Lanese, L. Bedogni, and M. Di Felice. Internet of things: a process calculus approach. In *Procs of the 28th Annual ACM Symposium on Applied Computing, SAC '13*, pages 1339–1346. ACM, 2013.

[9] R. Lanotte and M. Merro. A semantic theory of the Internet of Things. In *Procs. of Coordination 2016*, volume 9686 of *LNCS*, pages 157–174. Springer, 2016.

[10] D. Schoepe, M. Balliu, B. C. Pierce, and A. Sabelfeld. Explicit secrecy: A policy for taint tracking. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 15–30, 2016.

[11] E. J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *31st IEEE Symposium on Security and Privacy, S&P 2010*, pages 317–331. IEEE Computer Society, 2010.

[12] T. Zillner. ZigBee Exploited, 2015.

A Appendix

A.1 Operational Semantics of IoT-LySa

Our reduction semantics is based on the following *Structural congruence* $\equiv$ on nodes and node components. It is standard except for rule (3) 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.

$$
\begin{array}{ll}
(1) & (\mathcal{N}/_{\equiv}, |, \mathsf{0}) \text{ and } (\mathcal{B}/_{\equiv}, \|, \mathsf{0}) \text{ are commutative monoids} \\
(2) & \mu\, h\,.\, X \equiv X\{\mu\, h\,.\, X/h\} \quad \text{for } X \in \{P, A, S\} \\
(3) & \langle\langle E_1, \cdots, E_r \rangle\rangle : \varnothing.\ \mathsf{0} \equiv \mathsf{0}
\end{array}
$$

We have a two-level *reduction relation* $\rightarrow$ defined as the least relation on nodes and its components satisfying the set of inference rules in Table 2. For the sake of simplicity, we use

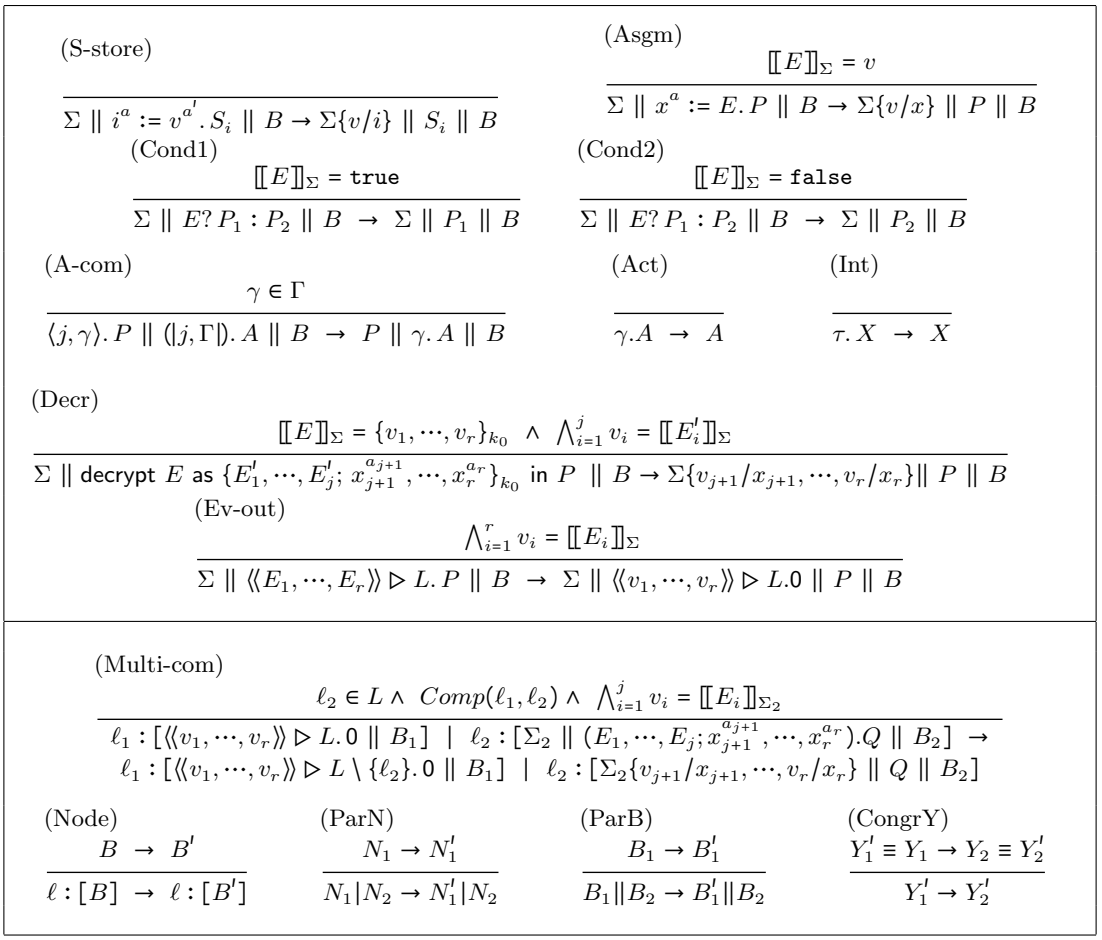

Table 2: Reduction semantics (the upper part on nodes, the lower one on node components), where $X \in \{S, A\}$ and $Y \in \{N, B\}$.

one relation. We assume the standard denotational interpretation $[\![E]\!]_\Sigma$ for evaluating terms, while for the proof of the subject reduction results 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:

$$[\![v^a]\!]^i_{\Sigma^i_\ell} = (v^a, \nu^\diamond) \qquad [\![i^a]\!]^i_{\Sigma^i_\ell} = \Sigma^i_\ell(i) \qquad [\![x^a]\!]^i_{\Sigma^i_\ell} = \Sigma^i_\ell(x)$$
$$[\![f^a(E_1, \cdots, E_r)]\!]^i_{\Sigma^i_\ell} = (f^a([\![E_1]\!]^i_{\Sigma^i_{\ell\downarrow_1}}, \cdots, [\![E_r]\!]^i_{\Sigma^i_{\ell\downarrow_1}}), F_\tau(f^a, [\![E_1]\!]^i_{\Sigma^i_{\ell\downarrow_2}}, \cdots, [\![E_r]\!]^i_{\Sigma^i_{\ell\downarrow_2}}))$$
$$[\![\{E_1, \cdots, E_r\}^a_k]\!]^i_{\Sigma^i_\ell} = (\{[\![E_1]\!]^i_{\Sigma^i_{\ell\downarrow_1}}, \cdots, [\![E_r]\!]^i_{\Sigma^i_{\ell\downarrow_1}}\}_k, \lfloor\{[\![E_1]\!]^i_{\Sigma^i_{\ell\downarrow_2}}, \cdots, [\![E_r]\!]^i_{\Sigma^i_{\ell\downarrow_2}}\}^b\rfloor_d)$$
$$\text{where } b = Enc_\tau([\![E_1]\!]^i_{\Sigma^i_{\ell\downarrow_2}}, \cdots, [\![E_r]\!]^i_{\Sigma^i_{\ell\downarrow_2}})$$

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

$$\frac{\tau(i,\ell) \in \Theta(\ell)(a)}{(\hat{\Sigma},\Theta) \vDash_\ell i^a} \qquad \frac{\tau(v,\ell) \in \Theta(\ell)(a)}{(\hat{\Sigma},\Theta) \vDash_\ell v^a} \qquad \frac{\tau(x,\ell) \otimes \hat{\Sigma}_\ell(x) \sqsubseteq \Theta(\ell)(a)}{(\hat{\Sigma},\Theta) \vDash_\ell x^a}$$

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

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

Table 3: Analysis of labelled terms $(\hat{\Sigma},\Theta) \vDash_\ell M^a$.

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) that drive asynchronous multi-communications are discussed in Section 3. 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 $\{E_1',\cdots,E_j';x_{j+1},\cdots,x_r\}_{k_0}$. As for 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.

A.2 Control Flow Analysis of IoT-LySA

Our CFA is specified in a logical form through a set of inference rules expressing the validity of the analysis results, where the function $\lfloor - \rfloor_d$ to cut all the terms with a depth greater than a given threshold d (with the special abstract values $\top^b$) is defined as follows.

$$\lfloor \top^b \rfloor_d = \top^b \quad \lfloor \nu^b \rfloor_d = \nu^b \quad \lfloor \{\hat{v}_1,\cdots,\hat{v}_r\}_{k_0}^b \rfloor_0 = \top^b \quad \lfloor \{\hat{v}_1,\cdots,\hat{v}_r\}_{k_0}^b \rfloor_d = \{\lfloor \hat{v}_1 \rfloor_{d-1},\cdots,\lfloor \hat{v}_r \rfloor_{d-1}\}_{k_0}^b$$

The result or *estimate* of our CFA is a triple $(\hat{\Sigma},\kappa,\Theta)$ (a pair $(\hat{\Sigma},\Theta)$ when analysing a term) that satisfies the judgements defined by the axioms and rules of Tables 3 and 4.

We do not comment the clauses discussed in Section 4. The judgement $(\hat{\Sigma},\Theta) \vDash_\ell M^a$, defined by the rules in Table 3, 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, $\Theta(\ell)(a)$ must include $\tau(i,\ell)$ and $\tau(v,\ell)$, respectively, i.e. the corresponding taint classifications decided by the designer. 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 $\{\hat{v}_1,\cdots,\hat{v}_r\}_{k_0}^b$, cut at depth d, belongs to $\Theta(\ell)(a)$, where $b = Enc_\tau(\hat{v}_1,..,\hat{v}_r)$. The special abstract value $\top^b$ 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 abstract value ν^b belongs to $\Theta(\ell)(a)$, where $b = F_\tau(f,\hat{v}_1,..,\hat{v}_r)$.

$$\frac{}{(\hat{\Sigma},\kappa,\Theta) \vDash 0} \qquad \frac{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell B}{(\hat{\Sigma},\kappa,\Theta) \vDash \ell : [B]} \qquad \frac{(\hat{\Sigma},\kappa,\Theta) \vDash N_1 \wedge (\hat{\Sigma},\kappa,\Theta) \vDash N_2}{(\hat{\Sigma},\kappa,\Theta) \vDash N_1 \mid N_2}$$

$$\frac{\forall\, i \in \mathcal{I}_\ell.\ \tau(i,\ell) \in \hat{\Sigma}_\ell(i)}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell \Sigma} \qquad \frac{}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell S} \qquad \frac{}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell A}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{k} (\hat{\Sigma},\Theta) \vDash_\ell M_i^{a_i} \ \wedge\ (\hat{\Sigma},\kappa,\Theta) \vDash_\ell P \ \wedge \\ \forall \hat{v}_1,\cdots,\hat{v}_r :\ \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \ \Rightarrow \forall \ell' \in L : (\ell, \langle\!\langle \hat{v}_1,\cdots,\hat{v}_r \rangle\!\rangle) \in \kappa(\ell')\end{array}}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell \langle\!\langle M_1^{a_1},\cdots,M_r^{a_r} \rangle\!\rangle \rhd L.\, P}$$

$$\frac{\begin{array}{c}\bigwedge_{i=1}^{j} (\hat{\Sigma},\Theta) \vDash_\ell M_i^{a_i} \ \wedge \\ \forall (\ell', \langle\!\langle \hat{v}_1,\cdots,\hat{v}_r \rangle\!\rangle) \in \kappa(\ell) :\ Comp(\ell',\ell) \Rightarrow \left(\bigwedge_{i=j+1}^{r} \hat{v}_i \otimes \tau(x_i,\ell) \in \hat{\Sigma}_\ell(x_i) \ \wedge\ (\hat{\Sigma},\kappa,\Theta) \vDash_\ell P\right)\end{array}}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell (M_1^{a_1},\cdots,M_j^{a_j};\, x_{j+1}^{a_{j+1}},\cdots,x_r^{a_r}).\, P}$$

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

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

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

$$\frac{}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell 0} \qquad \frac{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell P}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell \mu h.\, P} \qquad \frac{}{(\hat{\Sigma},\kappa,\Theta) \vDash_\ell h}$$

Table 4: Analysis of nodes $(\hat{\Sigma},\kappa,\Theta) \vDash N$, and of node components $(\hat{\Sigma},\kappa,\Theta) \vDash_\ell B$.

The judgements for nodes with the form $(\hat{\Sigma},\kappa,\Theta) \vDash N$ are defined by the rules in Table 4. 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 $(\hat{\Sigma},\kappa,\Theta) \vDash_\ell B$, where ℓ is the label of the enclosing node. The rule connecting actual stores Σ with abstract ones $\hat{\Sigma}$ 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 to analyse the immediate sub-processes. 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 $\hat{\Sigma}_\ell(x)$, once combined with the variable taint with the operator $\otimes$. 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).