

A Modeling Language for Security Threats of IoT Systems

Delphine Beaulaton, Ioana Cristescu, Axel Legay, Jean Quilbeuf

▶ **To cite this version:**

Delphine Beaulaton, Ioana Cristescu, Axel Legay, Jean Quilbeuf. A Modeling Language for Security Threats of IoT Systems. Formal Methods for Industrial Critical Systems - 23rd International Conference, FMICS 2018, 11119, Springer, pp.258-268, 2018, LNCS, 978-3-030-00243-5. 10.1007/978-3-030-00244-2_17 . hal-01962080

HAL Id: hal-01962080
https://inria.hal.science/hal-01962080

Submitted on 20 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Modeling Language for Security Threats of IoT Systems

Delphine Beaulaton[1], Ioana Cristescu[2], Axel Legay[2], and Jean Quilbeuf[2]

[1] Univ. South Brittany, Irisa, Vannes, France
[2] INRIA Rennes France

Abstract. We propose a security-based modeling language for IoT systems with two important features: vulnerabilities are explicitly represented and interactions are allowed or denied based on the information stored on the IoT devices. An IoT system is transformed in BIP, a component-based modeling language, in which can execute the system and perform security analysis. As proof-of-concept for our approach we model an attack on the Amazon Smart-Key system.

Keywords: IoT systems · Component-based Specifications · Security.

1 Introduction

IoT systems are part of our daily lives, as we are surrounded by computing device that communicates through the Internet. The IoT devices often have access to personal, confidential information that needs to be shared with other devices in order to provide *smart* services. Security attacks on IoT systems exploit the vulnerabilities of the different devices, and their interactions, to steal the sensitive data [2, 13].

We propose a modeling language for IoT systems in which vulnerabilities are explicitly represented. A malicious *entity*, that we usually call an Attacker, tries to break a security property of the system by using different attack scenarios. The other entities in the system can accidentally help the Attacker by *leaking* sensitive data. If one attack scenario violates the security property, our analysis concludes that the system is vulnerable to security threats.

Another feature of our language is that the interactions are permitted only between entities that share some *knowledge*. It is often the case that IoT devices require a password or share security keys to ensure their identity and to communicate with the rest of the system. In our approach, *protocols* supervise the interactions and verify that the two communicating entities have some common knowledge. The Attacker assumes the identity of the other entities by obtaining their knowledge.

Running Example. In the Amazon Smart-Key [1] system, shown in Figure 1, Amazon provides Home Owners with a Smart Lock and a Camera. The Camera can communicate with the Amazon Server to send all its recordings. A Home

Owner asks for a delivery at a time when she is not home (step 1). The Delivery Guy sends the package number to the Amazon server (step 2) from which it receives a temporary code (step 3) that, if communicated to the Smart Lock, can open the door (step 4). The Smart Lock is also in charge for turning on the Camera as soon as the door is unlocked (step 5). The Delivery Guy leaves the package inside and asks the Smart Lock to close the door (step 6) and to turn off the Camera (step 7). The Camera sends the video of the delivery to the Amazon Server (step 8) which forwards it to the Home Owner (step 9).

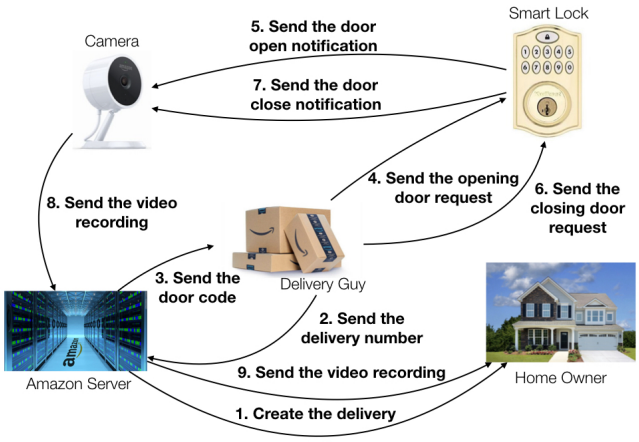


Fig. 1. Amazon Smart-Key

In our example, the security property is that the camera is recording as long as the door is open, to prevent thefts. Let us suppose that an Attacker intercepts the communication between the Delivery Guy and the Smart Lock. The Delivery Guy leaks the code for closing the door, which never reaches the Smart Lock. The communication between the Smart Lock and the Camera also has to be intercepted. The Attacker assumes the identity of the Smart Lock and sends a message to the Camera to turn off the recording. If this two-steps scenario succeeds the home is vulnerable to thefts as the door is open and the camera is turned off.

Outline. In Section 2 we introduce our language and show how we can model the Amazon Smart-Key system. An IoT model is translated in BIP in Section 3, which is equipped with an execution framework. We use the system's executions in Section 4 to verify whether a system is vulnerable to security attacks. As this is preliminary work for integrating probabilities and performing more complex analysis on IoT systems, we sketch the future works and conclude in Section 5.

2 A Modeling Language for IoT Systems

We model interconnected devices in an IoT systems as *entities*, that have each an identifier, ranged over by $e_1, \cdots, e_n$. Entities communicate using protocols and exchange values. Formally we write E for the set of identifiers, C for the set of protocols and *Val* for the set of values.

Protocols supervise the interactions, by verifying that some knowledge is shared between the communicating entities. We model the *knowledge* of an entity as a function $k : E \times C \to \mathit{PowerSet}(\mathit{Val})$ which associates to an entity e and a protocol c a set of values. For simplicity we write k_i^c for the application of k to the entity e_i and the protocol c. Then an interaction between entities e_1 and e_2 is permitted by the protocol c if there exists one common value between k_1^c and k_2^c.

Each entity has a CCS-like process [11] defined by the grammar in Figure 2. A process $a.P$ executes an action a and continues as P. The process $a.P + b.Q$ chooses between two execution branches, either $a.P$ or $b.Q$. We use A to denote (recursive) definitions, and 0 to denote the inactive process.

Two entities communicate through a pair of actions (Send, Receive), that represent "safe" interactions or (Leak, Collect) which signal an inadvertent interaction with a malicious entity. Actions have to specify the identifiers of the sender and the receiver and a value v which is exchange during the interaction. Moreover safe interactions also specify a *protocol* c. Entities can also compute *internally*, without involving any interaction, represented by the action τ.

$$
\begin{array}{rl}
\textit{Process} & P, Q \ ::= \ a.P \mid a.P + b.Q \mid A \mid 0 \\
\textit{Action} & a, b \ ::= e_1 \xrightarrow[v]{c} e_2 \text{ (Send)} \mid e_1 \xleftarrow{c} e_2 \text{ (Receive)} \\
& \qquad \mid e_1 \overset{c}{\underset{v}{\rightarrowtail}} e_2 \text{ (Leak)} \mid e_1 \overset{c}{\leftarrowtail} e_2 \text{ (Collect)} \mid \tau \text{ (Internal)} \\
\textit{Definition} & A \stackrel{\text{def}}{=} P
\end{array}
$$

Fig. 2. The Syntax of the IoT calculus

IoT transition systems An entity's state $\langle P, k \rangle$ consists of a running process and a current knowledge. A global state (of an IoT system) s is defined by the following grammar:

$$s \ ::= \emptyset \ \Big| \ \langle P, k \rangle \ \Big| \ s \mid s.$$

that is it can be either empty, a local state or the composition of the local states of each of its constituents. An IoT transition system consists of the tuple (S, T, i) where S is a set of global states, $i \in S$ is an initial state and $T \subseteq S \times S$ is a set

$$\textsc{SendReceive}\ \frac{k_1^c \cap k_2^c \neq \emptyset \qquad c' = \mathsf{protocol}(v)}{\langle e_1 \xrightarrow[v]{c} e_2.P_1, k_1\rangle | \langle e_2 \xleftarrow{c} e_1.P_2, k_2\rangle \rightarrow \langle P_1, k_1\rangle | \langle P_2, k_2^{c'} \uplus \{v\}\rangle}$$

$$\textsc{LeakCollect}\ \frac{c' = \mathsf{protocol}(v)}{\langle e_1 \underset{v}{\rightarrowtail} e_2.P_1, k_1\rangle | \langle e_2 \leftarrowtail e_1.P_2, k_2\rangle \rightarrow \langle P_1, k_1\rangle | \langle P_2, k_2^{c'} \uplus \{v\}\rangle}$$

$$\textsc{Internal}\ \langle \tau.P, k\rangle \rightarrow \langle P, k\rangle \qquad \textsc{Congruence}\ \frac{s \equiv_s t \rightarrow s' \equiv_s t'}{s \rightarrow s'}$$

$$\textsc{Sum}\ \frac{\langle P_i, k_i\rangle | \langle P_j, k_j\rangle \rightarrow \langle P_i', k_i'\rangle | \langle P_j', k_j'\rangle}{\langle P_i + Q_i, k_i\rangle | \langle P_j + Q_j, k_j\rangle \rightarrow \langle P_i', k_i'\rangle | \langle P_j', k_j'\rangle} \qquad \textsc{Par}\ \frac{s \rightarrow s'}{s \mid t \rightarrow s' \mid t}$$

Fig. 3. The Operational Semantics of the IoT calculus

of transitions that are derived by the rules in Figure 3. We write $s \rightarrow s'$ for a transition and $s_0 \rightarrow s_1 \cdots s_{n-1} \rightarrow s_n$ for an *execution* of the system.

The rules SendReceive and LeakCollect describe the communication between two entities e_1 and e_2 with the knowledge functions k_1 and k_2, respectively. The two entities exchange a value v which is added to the knowledge of e_2 under protocol c'. The SendReceive interaction also verifies that the two entities share a common value in their knowledge for a protocol c. An internal computation is derived by the rule Internal. The rules Par and Sum allow derivations inside the state's composition and the sum constructors. Lastly, the rule Congruence rewrites a state or a process into a syntactic form that is suitable for a derivation. It uses two *congruence* relations $\equiv_P$ and $\equiv_S$, defined as the smallest equivalence relations on processes and states, respectively, such that

- $\equiv_P$ includes the abelian monoid laws for $+$ and the unfolding law for definitions:

$$P + Q \equiv_P Q + P \quad (P + Q) + R \equiv_P P + (Q + R) \quad P + 0 \equiv_P P$$
$$A \equiv_P P \text{ if } A \stackrel{\text{def}}{=} P.$$

- $\equiv_S$ includes the abelian monoid laws for $|$ and generalizes $\equiv_P$ to states:

$$s|t \equiv_s t|s \quad (s|t)|q \equiv_s s|(t|q) \quad s|\emptyset \equiv_s s \quad \frac{P \equiv_P Q}{\langle P, k\rangle \equiv_S \langle Q, k\rangle}$$

The Amazon Smart-Key example In modeling our example, we have the following entities: the Home Owner H, the Amazon Server S, the Delivery Guy D, the Smart Lock L, the Camera C and the Attacker A. The initial process for

each entity is given in Figure 4. The protocols used in the example are delivery, doorControl, cameraCom and customerCom.

$$
\begin{aligned}
\text{AmazonServer} =& S \overset{\text{customerCom}}{\longleftarrow} H.S \overset{\text{delivery}}{\longleftarrow} D.S \underset{\text{doorCode}}{\overset{\text{delivery}}{\longrightarrow}} D.S \overset{\text{cameraCom}}{\longleftarrow} C. \\
& S \underset{\text{video}}{\overset{\text{customerCom}}{\longrightarrow}} H.\text{AmazonServer} \\
\text{HomeOwner} =& H \underset{\text{askDelivery}}{\overset{\text{customerCom}}{\longrightarrow}} S.H \overset{\text{customerCom}}{\longleftarrow} S.\text{HomeOwner}
\end{aligned}
$$

$$
\begin{aligned}
\text{DeliveryGuy} =& D \underset{\text{deliveryArrived}}{\overset{\text{delivery}}{\longrightarrow}} S.D \overset{\text{delivery}}{\longleftarrow} S.D \underset{\text{openDoor}}{\overset{\text{doorControl}}{\longrightarrow}} L. \\
& (\tau.\text{normal_delivery} + \tau.\text{hijack_delivery}) \\
\text{normal_delivery} =& D \underset{\text{closeDoor}}{\overset{\text{doorControl}}{\longrightarrow}} L.\text{DeliveryGuy} \\
\text{hijack_delivery} =& D \underset{\text{lock1}}{\rightarrowtail} A.\text{DeliveryGuy}
\end{aligned}
$$

$$
\begin{aligned}
\text{SmartLock} =& L \overset{\text{doorControl}}{\longleftarrow} D.L \underset{\text{startRecording}}{\overset{\text{cameraCom}}{\longrightarrow}} C.(\tau.\text{normal_lock} + \tau.\text{hijack_lock}) \\
\text{normal_lock} =& L \overset{\text{doorControl}}{\longleftarrow} D.L \underset{\text{stopRecording}}{\overset{\text{cameraCom}}{\longrightarrow}} C.\text{SmartLock} \\
\text{hijack_lock} =& L \overset{\text{doorControl}}{\longleftarrow} A.L \underset{\text{camera1}}{\rightarrowtail} A.\text{SmartLock}
\end{aligned}
$$

$$
\begin{aligned}
\text{Camera} =& C \overset{\text{cameraCom}}{\longleftarrow} L.(C \overset{\text{cameraCom}}{\longleftarrow} L.C \underset{\text{video}}{\overset{\text{cameraCom}}{\longrightarrow}} S.\text{Camera} + \\
& C \overset{\text{cameraCom}}{\longleftarrow} A.C \underset{\text{video}}{\overset{\text{cameraCom}}{\longrightarrow}} S.\text{Camera}) \\
\text{Attacker} =& A \leftarrowtail D.A \underset{\text{getCameraId}}{\overset{\text{doorControl}}{\longrightarrow}} L.A \leftarrowtail L.A \underset{\text{stopRecording}}{\overset{\text{cameraCom}}{\longrightarrow}} C.\text{Attacker}
\end{aligned}
$$

Fig. 4. The Amazon Smart-Key system

We define the initial knowledge of each entity such that all safe communications are possible. For instance, the Amazon Server and the Home Owner can communicate because they both know the identity of the Home Owner: $k_S^{\text{customerCom}} = k_H^{\text{customerCom}} = \text{homeOwner1}$. However, initially the Attacker only knows the identity of the Delivery Guy and it is through the LeakCollect interactions that the Attacker acquires information that allows him to communicate with the Smart Lock and the Camera. To model this behaviour, the Delivery Guy and the Smart Lock have two execution branches, one called the *normal* behaviour, when the Attacker does not interfere and a second one, the *hijacked* one, where the system is vulnerable to the attack.

3 BIP: A Component-based Modeling Language

BIP [12, 10, 3] is a component-based modeling language where a system is modeled as the *composition* of a set of interacting components. BIP stands for *Behaviour* (each component has an abstract behaviour), *Interaction* (components interact with each other) and *Priorities* (interactions have a priority order, and higher priority interactions occur first). We start by introducing the components, and then move on to the interactions and the priorities.

A component has an abstract behaviour, represented by a labeled transition system (LTS). Components communicate with each other through *ports*. Transitions are labeled by ports and the set of ports used by a components is called the *interface* of the component.

Definition 1. *A component K is an LTS, denoted as (P, Q, q^0, T), where P is a set of* ports *constituting the interface of K, Q is the set of states, $q^0 \in Q$ is the initial state, and $T \subseteq Q \times P \times Q$ is the set of transitions labeled by ports from the interface P.*

We write $q^1 \xrightarrow{a} q^2$ if the tuple (q^1, a, q^2) is in T. The previous transition is said to be enabled when the component is at state q^1. An execution is a sequence of transitions that starts from the initial state $q^0 \xrightarrow{a_1} q^1 \cdots q^{n-1} \xrightarrow{a_n} q^n$.

The definition above can be extended by including variables in components. In such an extended version, there are three additional mechanism:

- *guards*: each transition has a predicate, name guard, defined on the variables. For a transition to be enabled, the guard must evaluate to true.
- *variables exchange*: each port exports a set of variables. Any interaction through that port can read and write these variables.
- *update functions*: each transition is labeled with an update function that set new values to the variables according to the previous ones.

These mechanisms interact as follows. First the guards are evaluated to list the enabled transitions. Then, when an interaction takes place, the variables exported by the associated port are potentially modified. Finally, the update function is applied to modify the variables of the component. Sometimes there are no variables to verify or update, in which case the guard function is the constant true or the update function is the identity.

In our case, we use variables to encode the knowledge of the entities. As an example, consider the component Amazon Server from Figure 5, representing the entity with the same name from our running example. The states $S_0, \cdots S_4$ represent the reachable processes from the *AmazonServer* of Figure 4. The protocols used in *AmazonServer* become the ports **customerCom, develivery, cameraCom**. Initially the

Composition of BIP components BIP systems consists of several components that interact with each other. An *interaction* consists of the synchronization of some local transitions labeled by ports. In the following we define a BIP system of n components. We write (P_i, Q_i, q_i^0, T_i) for the transition system of the component K_i for $i \leq n$.

Definition 2. *Let $(K_i)_{i\leq n}$ be n components such that their interfaces are pairwise disjoint, that is $i \neq j \implies P_i \cap P_j = \emptyset$. We define the set of all ports by $Ports = \bigcup_{i\leq n} P_i$. An interaction (a, α) consists of an* exported port *$a \notin Ports$, a non-empty set of ports $\alpha \subseteq Ports$ such that there exists at most one port from each interface P_i in α.*

Interactions involve at least one component. In the case where only one component participate in the interaction, we say that it is *internal* transition and we denote it with τ.

As for transitions in the atomic components, interactions can be extended to handle data, using guards and update functions. For each interaction, the set of variables exported by the participating ports defines the set of variables that are visible to the interaction. The interaction can only take place if the guard evaluates to true. When the interaction takes place, the update function modifies the value of the variables exported by the port.

In Figure 5 there is an interaction between the Amazon Server and the Home Owner defined on the ports **customerCom** of two components. These ports export the knowledge related to the corresponding protocol. The guard of interaction checks the existence of a common value between the the Amazon Server's knowledge and the Home Owner's knowledge. The update function simply propagates to the Amazon Server a message signaling a delivery request.

An *interaction model* γ is a set of interactions with distinct exported ports. An useful notation for the next definition is $I_a = \{i : \exists a \in P_i, a \in \alpha_i\}$ for the set of indexes of the components K_i that participate in the interaction. Note that, because in an interaction model the exported ports are distinct, we can distinguish an interaction (a, α) by the port a.

Definition 3. *Let γ be a set of interactions defined on the components $(K_i)_{i\leq n}$. We denote with $\langle\gamma\rangle(K_i)_{i\leq n}$ the component (P_γ, Q, q^0, T) where*

- *$P_\gamma = a : (a, \alpha) \in \gamma$ is the set of exported ports of γ;*
- *$Q = \prod_{i\leq n} Q_i$ with $q^0 = (q_1^0, \cdots q_n^0)$;*
- *T is the least set of transitions such that:*

$$\frac{(a, \alpha) \in \gamma \quad \forall i \in I_a, \exists a_i \in \alpha, q_i^1 \xrightarrow{a_i} q_i^2 \quad \forall i \notin I_a, q_i^2 = q_i^1}{(q_1^1, \cdots, q_n^1) \xrightarrow{a} (q_1^2, \cdots, q_n^2)}.$$

Priorities A priority order on a set of ports is a partial order, where each element $a < b$ of the order is called a *priority*. Whenever the system has a choice between the two interactions on ports a or b, the interaction on b is chosen. Formally, we introduce priorities as in Definition 3, following the approach in [10].

Definition 4. *Let $<$ be a priority order on the ports P of a component $K = (P, X, Q, q^0, T)$. We define $\langle< \rangle(K)$ as the component (P, X, Q, q^0, T') where only T' is changed to be the least set of transitions such that*

$$\frac{q^1 \xrightarrow{a} q^2 \in T \quad \nexists b \in P, \exists q \in Q, (a < b \wedge q^1 \xrightarrow{b} q)}{q^1 \xrightarrow{a} q^2}.$$

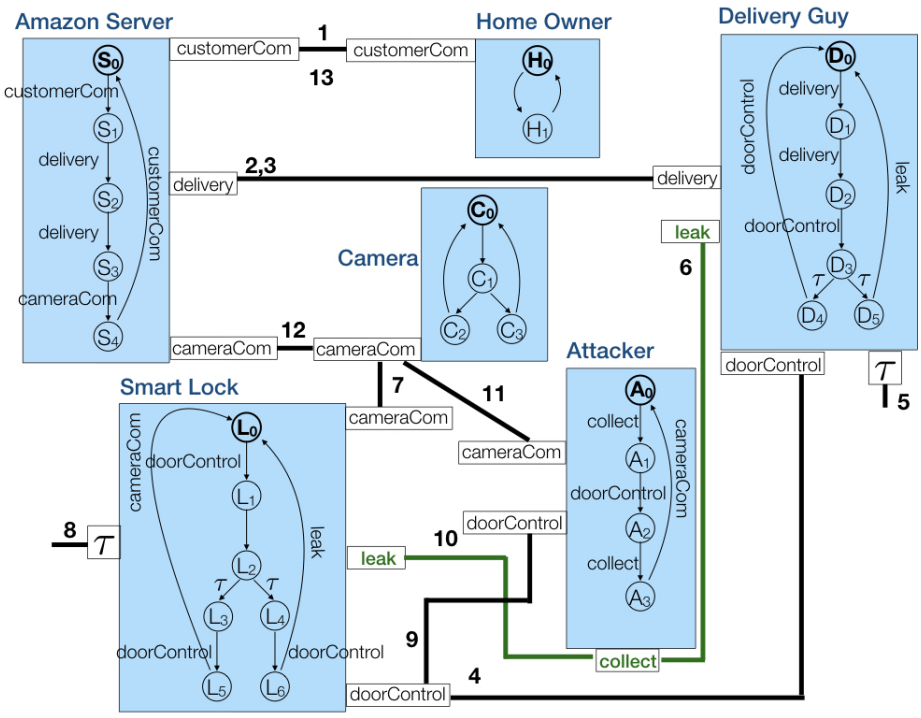


Fig. 5. The initial state of the Amazon Smart-Key transformed to BIP. Each IoT entity is transformed into the BIP component with the same name. The LTS of a BIP component represents the process of the corresponding IoT entity while the variables represent the knowledge. Each state in the LTS represents a reachable process. Actions connect the different states of the LTS. For simplicity, here the ports are the protocols, for the SendReceive interactions, τ denote the internal transitions and *leak*, *collect* the LeakCollect. Variables are sets of values, where each protocol has its own variable. Interactions are defined between ports that have the name, or are internal transitions, denoted τ.

We use a priority order, denoted $\ll$, which gives priority to the internal transitions over the binary interactions. We can also use the method in [10] to infer a priority order that avoids deadlocks (the system reaches states from which no transition is possible) as much as possible.

Giving more power to the Attacker In our model, we explicitly model the knowledge of the Attacker by using a set of values to represent it. Furthermore, this knowledge focuses mainly on actual data rather than the state of a component. However, some attacks might require to send a message when a component is in a particular state, which makes it vulnerable to an attack. To that end we could use techniques similar to the ones used for the *distributed controller* in [8, 9]. A distributed controller knows the behaviour of a system and can observe its

executions to infer the global state of the system. An Attacker can analyze the behaviour of a system to detect in which state it is most vulnerable. For example, an Attacker that knows how Amazon Smart-Home works can detect when the Delivery Guy opens the door of a user's home and proceed with an attack at that moment. A more powerful Attacker can also observe the data exchanged during communications. An attack in our case could consists in reusing the code for opening and closing a door that was used during a "safe" execution of the system.

Definition 5. *A* process π *is a subset of states in a component. A* property ϕ *is defined as a set of value for a variable* x *and it* holds *in a state* q *if* x *evaluates to a value of* ϕ *in state* q*. We say that* π *knows* ϕ *if* ϕ *holds in all reachable states of* π*.*

These notions are well studied in BIP, and therefore if integrated in the IoT language can enrich the security analysis of an IoT system.

Transformation of an IoT system to BIP An IoT system is translated into a BIP system. We do not show the transformation here, but is based on [4], along with the proof of the following theorem.

Theorem 1. *Let* $e_1, \cdots e_n$ *be a set of IoT entities with the initial states* $\langle P_i, k_i \rangle$*,* $i \leq n$*. The transformation in [4] from IoT to BIP produces a BIP component* K_i *for each* e_i *entity and a* γ *an interaction model such that there exists a bisimulation relation between* $\langle \ll \rangle \big(\langle \gamma \rangle (K_i)_{i \leq n} \big)$ *and* $\prod_{i \leq n} \langle P_i, k_i \rangle$*.*

The theorem allows us to analyze the system transformed in BIP, instead of the original one which is modeled in the IoT language. Any analysis on the executions of the BIP system holds for the IoT system. We do this in the next section.

4 Verification

The BIP language is equipped with a simulator that we use to obtain executions of an IoT system. As future work we will use the simulations for more complicated security analysis, based on statistical model checking [6]. For the moment, we simply use the simulation framework to guide the executions of a system to discover execution paths leading to security failures.

By simulating the system we can extract two executions for the Amazon Smart-Key system. To clarify notations, in the following execution we write the global state as a list of the local states of the different BIP components.

Let us informally describe an execution where the Attacker succeds. Each step of the attack also annotates the interactions in Figure 5. The initial state is $(S_0, H_0, L_0, C_0, D_0, A_0)$.

1. the Home Owner contacts the Amazon Server using the protocol customerCom, which verifies that $k_S^{\text{customerCom}} \cap k_H^{\text{customerCom}} = \{\text{homeOwner1}\}$, that is that both entities know the value *homeOwner1*. The Home Owner sends the message *askDelivery* to the Amazon Sever. The global state becomes $(\mathbf{S_1}, \mathbf{H_1}, L_0, C_0, D_0, A_0)$;
2. when the delivery arrives, the Delivery Guy sends the message *deliveryArrived* to the Amazon Server on protocol delivery, which verifies that both entities know the Delivery Guy identity, $k_D^{\text{delivery}} \cap k_S^{\text{delivery}} = \{\text{deliveryGuy1}\}$. The state changes to $(\mathbf{S_2}, H_1, L_0, C_0, \mathbf{D_1}, A_0)$;
3. the Amazon Server sends back the value *doorCode1* to the Delivery Guy, using the same protocol and the state is now $(\mathbf{S_3}, H_1, L_0, C_0, \mathbf{D_2}, A_0)$;
4. the Delivery Guy has the value *doorCode1* needed to communicate with the Smart Lock on procotol doorControl, $k_D^{\text{doorControl}} \cap k_L^{\text{doorControl}} = \{\text{doorCode1}\}$. The Delivery Guy sends the message *openDoor* and the state changes to $(S_3, H_1, \mathbf{L_1}, C_0, \mathbf{D_3}, A_0)$;
5. the Delivery Guy accidentally agrees to help the Attacker by choosing the internal transition leading to the global state $(S_3, H_1, L_1, C_0, \mathbf{D_5}, A_0)$;
6. the Delivery Guy leaks to the Attacker the value *doorCode1*. The state becomes $(S_3, H_1, L_1, C_0, \mathbf{D_0}, \mathbf{A_1})$;
7. the Smart Lock opens the door and sends the message *startRecording* to the Camera using the protocol cameraCom, for which the guard $k_L^{\text{cameraCom}} \cap k_C^{\text{cameraCom}} = \{camera1\}$ holds. The state is now $(S_3, H_1, \mathbf{L_2}, \mathbf{C_1}, D_0, A_1)$;
8. the Smart Lock is also recruited by the Attacker, when the Smart Lock chooses the internal transition leading to the global state $(S_3, H_1, \mathbf{L_4}, C_1, D_0, A_1)$;
9. the Attacker communicates with the Smart Lock using a safe communication, as now the Attacker knows the *doorCode1*. The condition of the protocol doorControl: $k_A^{\text{doorControl}} \cap k_L^{\text{doorControl}} = \{\text{doorCode1}\}$ holds. Then Attacker sends the message *getCameraId* and the state changes to $(S_3, H_1, \mathbf{L_6}, C_1, D_0, \mathbf{A_2})$;
10. the Smart Lock leaks the value *camera1* to the Attacker. The states is now $(S_3, H_1, \mathbf{L_0}, C_1, D_0, \mathbf{A_3})$;
11. the Attacker sends the message *stopRecording* to the Camera using the protocol cameraComwhich verifies that $k_A^{\text{cameraCom}} \cap k_C^{\text{cameraCom}} = \{camera1\}$. The global state is now $(S_3, H_1, L_0, \mathbf{C_3}, D_0, \mathbf{A_0})$;
12. unaware of the attack, the Camera sends an uncompromising *video* to the Amazon Server using the protocol cameraCom. The state is now $(\mathbf{S_4}, H_1, L_0, \mathbf{C_0}, D_0, A_0)$;
13. lastly, the Amazon Server forwards the *video* to the Home Owner using the protocol customerCom. The system is back in its initial state $(\mathbf{S_0}, \mathbf{H_0}, L_0, C_0, D_0, A_0)$.

The Smart Lock never received a message for closing the door, while the Camera did received a message to stop the recording. Therefore the door is open, the Camera switched off, and the home is vulnerable to thefts.

The Attacker proceeds in two steps: it first compromises the Delivery Guy and then compromises the Smart Lock. If either one of these two steps fails,

the attack fails. An example of a safe execution is one in which everything is as above, except for the step 5, where the Delivery Guy does an internal transition to the local state D_4 (instead of state D_5). The Attacker cannot then collect the value *doorCode1* and cannot communicate with the Smart Lock. Another possibility is for the Delivery Guy to leak the value *doorCode1* (and thus proceed as in step 5 and 6 described above), but the Smart Lock does not collaborate with the Attacker in step 8.

5 Conclusion

We introduced a modeling language for IoT systems, where the malicious entities are explicitly part of the system. The malicious entities interfere with the rest of the system to steal confidential data and leave the system vulnerable. A system modeled in our language is transformed into a BIP system which is then executed. Execution traces leading to successful attacks are proofs of the vulnerabilities of a system.

As future work we plan to add probabilities to the IoT language, similarly to the probabilistic CCS [7] and use *statistical* BIP [5] to simulate these systems. In the plasma tool [6] we can then apply statistical model checking techniques to provide more complex security analysis.

References

[1] *Amazon Key*. `https://www.amazon.com/key`. Accessed: 2018-06-22.

[2] Manos Antonakakis et al. "Understanding the mirai botnet". In: *26th USENIX Security Symposium*. 2017. URL: `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis`.

[3] Ananda Basu, Marius Bozga, and Joseph Sifakis. "Modeling Heterogeneous Real-time Components in BIP". In: *4th SEFM Conference*. 2006. DOI: `10.1109/SEFM.2006.27`.

[4] Delphine Beaulaton et al. "A Language for Analyzing Security of IOT Systems". In: *13th SOSE Conference*. 2018.

[5] Saddek Bensalem et al. "Statistical Model Checking Qos Properties of Systems with SBIP". In: *5th ISoLA Conference*. 2012. DOI: `10.1007/978-3-642-34026-0_25`.

[6] Benoît Boyer et al. "PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library". In: *QEST Conference*. 2013. DOI: `10.1007/978-3-642-40196-1_12`.

[7] Rob Van Glabbeek, Scott Smolka, and Bernhard Steffen. "Reactive, Generative, and Stratified Models of Probabilistic Processes". In: *Information and Computation* 121.1 (1995). DOI: `https://doi.org/10.1006/inco.1995.1123`.

[8] Susanne Graf. "Distributed Implementation of Constrained Systems Based on Knowledge". In: *13th ISPDC Conference*. 2014. DOI: 10.1109/ISPDC.2014.32.

[9] Susanne Graf and Sophie Quinton. "Knowledge for the Distributed Implementation of Constrained Systems". In: *Softw. Syst. Model.* Vol. 15. 4. 2013. DOI: 10.1007/s10270-014-0451-z.

[10] Imene Ben Hafaiedh, Susanne Graf, and Sophie Quinton. "Building Distributed Controllers for Systems with Priorities". In: *J. Log. Algebr. Program.* 80.3 (2011), pp. 194–218.

[11] Robert Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., 1982.

[12] Joseph Sifakis. "A Framework for Component-based Construction Extended Abstract". In: *3th SEFM Conference*. 2005. DOI: 10.1109/SEFM.2005.3.

[13] TrapX Security Inc. TrapX LAbs. *Anatomy of an Attack, MEDJACK (Medical Device Attack)*. Tech. rep. May 2015.