Trustworthy Privacy Policy Translation in Untrusted IoT Environments

Mamadou H. Diallo, Nisha Panwar, Roberto Yus and Sharad Mehrotra
Department of Computer Science, University of California at Irvine, Irvine, U.S.A.

Keywords: IoT Trust, Security and Privacy, Trusted Computing, Privacy Policies, Contracts, Encryption, Remote Attestation.

Abstract: Internet of Thing (IoT) systems, such as smart buildings and smart cities, provide services to users (individuals and organizations) in various aspect of our lives. To provide such services, IoT systems need to handle data captured from multiple devices/sensors, and translation of data processing policies agreed by users (high-level) into commands for devices (device-level). The underlying assumption is that users trust IoT systems in honoring their policies. However, this trust assumption is incorrectly positioned since IoT systems may not be honest or may fall victim to cyberattacks. We address such concerns by providing mechanisms to help in ensuring trust and accountability at the time of translating a contract (agreed and signed policies). The objective of the proposed scheme is two fold, (1) translation of contracts from a high-level to device-level, (2) attestation of the translation. We have implemented the proposed scheme for contract translation and attestation of translation as a module and integrated it with the TIPPERS system (our IoT testbed under development). The results of our experiments highlight the feasibility of our proposed schemes.

1 INTRODUCTION

IoT systems influence all aspects of modern life such as health care, transportation, buildings, infrastructure, and emergency response, among others (Madakam and Date, 2016; Gubbi et al., 2013). In this study, we focus particularly in user-centric IoT systems wherein sensors and devices of diverse types – cameras, cell phones, WiFi access points (APs), beacons, occupancy sensors, temperature sensors, light sensors, and acoustic sensors – are used to collect data and to create awareness about user interactions in the IoT space. While the data collection about users is necessary for the IoT system to tailor services toward users, it raises several security and privacy concerns (Varadharajan and Bansal, 2016; Aikins, 2016; Zhao and Ge, 2013; Farooq et al., 2015).

To address security and privacy concerns when collecting and processing users data, IoT systems can leverage privacy policies (Pappachan et al., 2017), which enable users to decide how their data should be collected and processed. Note that IoT systems are typically designed using a layered approach to abstract out data from a lower layer to a higher layer. However, while this data abstraction is useful in interacting with users through applications using a higher level of data abstraction (e.g. application-level) where concepts and properties that are semantically more meaningful to users are used, it introduces a semantic gap between high-level concepts and low-level concepts. Note that this semantic gap extends to policies and introduces a complexity of translating the policies from high-level to device-level. For instance, consider a WiFi AP in a smart building system, which can record MAC addresses of devices connected to it. This raw data may not be meaningful for users. However, when this data is interpreted in the context of the rooms in the building, the time, and individuals coming in and out of the building, then more meaningful information can be inferred. From the list of MAC addresses, it would be possible to learn about the habits of the building's inhabitants such as frequencies of individual movements.

To address the policy semantic gap in translating policies from application-level to the device-level, specific information about the IoT application domain is needed. This specific information, which we refer to as IoT domain knowledge, needs to encompass information about the IoT smart space, devices and sensors deployed in the space, and inhabitants of the space. There have been many approaches in the literature in building IoT smart space knowledge bases (Nambi et al., 2014; Balaji et al., 2016; Wang et al., 2012; Han et al., 2014). For instance, Brick

Diallo, M., Panwar, N., Yus, R. and Mehrotra, S.
Trustworthy Privacy Policy Translation in Untrusted IoT Environments.
DOI: 10.5220/0006765501320143
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 132-143
ISBN: 978-989-758-296-7

(Balaji et al., 2016) is an example of an ontology to model smart buildings. This complex task of modeling and generating this knowledge base is compounded by the need for managing static data as well as dynamic data.

Given the specific domain knowledge of a system, translating application policies to device policies is challenging since the mapping is not one-to-one and there may be different choices with different levels of privacy. One high-level policy may needs to be translated into a number of device policies as multiple devices may be required to check the contextual conditions of the policy. For example, consider a high-level policy stating that "Deliver my work related messages only when I'm in my office". The context in this policy is the user being in the office. Therefore, one or more devices (with the capability of precise localization of the user) are required to verify this context.

To address the aforementioned concerns, we propose a trustworthy contract translation approach, which empowers users to retroactively attest contract translation by the IoT space. A contract is a mutually agreed and signed policy between the IoT system and one or more users. Consider a set of devices $(D_1, D_2, ..., D_n)$ in an IoT infrastructure and a policy P that dictates the condition under which the devices D_i can or cannot capture data (Yan et al., 2014). P may be the policy dictated by the IoT environment or a result of an agreement between the IoT space and the subject. In any case, the resulting agreed policy, when signed by both parties, infrastructure and user, represents a *mutual contract* or simply *contract* that dictates the norms of data capture about any individual, especially, in cases where a user consent is necessary from the privacy perspective.

The attestation of contract translation involves checking the integrity of the domain knowledge and the integrity of the translation procedure. We use a continuous logging mechanism to keep track of the availability of infrastructure devices at anytime, which are then used to check the integrity of the domain knowledge. If the integrity of the domain knowledge is certified, then the contract translation procedure is guaranteed to produce correct and complete device contracts. However, maintaining the domain knowledge in accordance with the translation of policies is challenging. For instance, consider a device in a smart building that is used to locate individuals in a room. If the device stops working after a policy is translated to the device before enforcement, then the translation would not be correct.

Therefore, the main contributions of this paper are as follows:

- We introduce IoT policy and contract (agreed policies) model and a translation approach for translating high-level contracts into device-level contracts.
- We propose an attestation of contract translation approach, based on tamperproof audit logs to attest the integrity of the contract translation procedure.
- We implemented the contract translation and attestation of translation approaches as a module and integrated it with TIPPERS (Mehrotra et al., 2016). We performed a number of experiments to analyze the feasibility of the overall proposed approach.

2 IoT SYSTEM MODEL

Our IoT system model follows a layered design approach to abstract out the complexity of defining and enforcing mutual contracts for the devices. Essential for this design approach are a model of the devices and the smart space, which we explain in the following.

2.1 Device Model

We consider two types of devices, physical devices that capture raw data and virtual devices that transform the raw data to semantic data. The physical devices include user devices and infrastructure.

Physical Device. We model a physical device D_i (e.g., a camera) "used to collect data in an IoT space" as consisting of the following three sets.

- *Device states* ($D_i.states$) are the states a device can be in. For example, a video camera can be in an "active" state or "inactive" state. Similarly, a WiFi AP device can be in an "active" state providing signals to enable wireless devices to connect to it or it could be "inactive".
- *Device actions* ($D_i.actions$) are actions that trigger a device's state transition (e.g., "*switching on*" or "*switching off*") from one state to another. In addition, a *Device action* also includes a *function* that the device could perform on its payloads. For instance, after recording a data stream, if a camera is connected to a server, it can send the video stream to the server. It can also encrypt the data, if supported, before streaming it to the server. If equipped with a logging software, the camera can even log its own states and actions. In this example, the actions include "encrypting data", "sending data", and "logging states and actions".

- *Device payloads* ($D_i.payloads$) are data elements (observations) generated by the device. The type of a payload is specific to the device that generates it. A *Device payload* is further associated with the set of functions ($D_i.payload.functions$) that can be performed on it. For instance, a video stream can be associated with a face detection function and a face recognition function. Likewise, a WiFi AP's payload can be associated with a function that determines how much time users spend in a location. These functions can be provided by devices or the server that processes the payloads.

Note that, while a physical device may contain multiple sensors (e.g. a video camera has an image recording sensor, and a sound recording sensor), for the purpose of defining and enforcing policies, we consider the whole device as a basic unit.

Virtual Devices. In addition to physical devices, we consider the concept of a virtual device, which is essentially a software program used to transform data from a lower to a higher data abstraction. An example of virtual device is *People Locator*, which reads data from one or more physical devices (e.g. video cameras, WiFi APs, beacons, etc.) in a space and locates an individual. Note that one virtual device can include the data from other virtual devices. For instance, a *People Counter* virtual device can take as input the location information produced by the *People Locator* virtual device to count the number of people present in the space. A *Time Spent* virtual device can use the same information to determine the time an individual spent in the space.

2.2 Domain Knowledge Model

The domain model encapsulates the knowledge specific to an IoT domain into a domain knowledge (*DK*). This *DK* is necessary in bridging the semantic gap between the high-level concepts (e.g. application-level) and device-level concepts. *DK* comprises the entities that define the IoT smart space for the domain. We model entities as consisting of two types: concepts and properties. The concepts are essentially the space ($DK.space$) and its inhabitants ($DK.users$) and can be hierarchically represented. For instance, consider the smart building domain, a building contains floors and a floor contains rooms, which can be represented as $Building.Floors.Rooms$. Likewise, a university campus community can be hierarchically organized (e.g., $community.students.undergrad$, $community.faculty.associates$). Properties further refine concepts by giving more information about the concepts. For instance a room may have the property of capacity, occupancy, or temperature ($room.capacity$, $room.occupancy$, $room.temperature$). A student concept *student* may have properties $student.id$, $student.name$, $student.phone$, and $student.gpa$. Properties can be categorized as static or dynamic. Note that the static properties change rarely while dynamic properties change often overtime (e.g., $room.name$ is static – whereas $room.temperature$ is dynamic). This domain knowledge is necessary in bridging the semantic gap between the high-level concepts and properties and device-level concepts and properties.

2.3 Application Model

The application model defines concepts and properties for representing an application and its environment. An application makes use of the semantic data generated by the IoT system through virtual sensors based on the domain knowledge to provide services to users. Therefore, the application concepts and properties need to be in line with the domain concepts and properties. Application concepts include the application semantic data, the services/functionalities that operate on the data. We model the application semantic data as set of objects (O), where each object o ($o \in O$) comprises a set of attributes/properties ($o.A$). For instance, in a smart concierge application, an event (e) may be represented an an object and the properties of the event (what, when, where, etc.) as the attributes ($e.what, e.when, e.where, etc.$). These abstract application level properties can then be mapped to the domain properties. For example, $e.where$ corresponds to the physical location in domain model where the needs to take place.

3 POLICY AND CONTRACT MODEL

In this section, we formally define our policy and contract model, and summarize our logging mechanism for ensuring that logs cannot be tampered without detection. Table 1 summarizes the notations used throughout the paper.

In our IoT system model, a policy defines the conditions under which the IoT system manages users' data (Pappachan et al., 2017). Policies can be for capture/collection, analysis, sharing, retention, etc. In this paper, we focus on capture policies, and from now on, policies refer to capture policies. A policy determines the data that can be collected, specifies the form (e.g., encrypted or plaintext) of

data, and mandates the conditions (e.g., videos record individuals only when they are detected in a specific location or only during a specific period of time) under which the data can be collected.

A policy is first specified at a high-level of data abstraction using high-level concepts and properties (*Application Model*) before being translated into one or more device-level policies using the domain knowledge (*DK*). The device policies are expressed using the *Device Model* and are enforced by the IoT system. As an example, consider a high-level policy for a smart building stating that the system will not track an individual in a given space. In this case, any device D_i within the vicinity of the space registered in *DK* ($D_i \in DK.devices$), which can be used to track the individual, will be subject to policy enforcement. This high-level policy will need to be translated to device-level policies for all such devices to restrict their operations to enforce the policy.

Table 1: Notations.

U_i	User	D_U	User devices
I	Infrastructure	D_I	Infrastructure devices
E	Entity	$Cert$	Public key certificate
D_i	Devices	D_C	Context devices
$D_i.id$	Device identity	$D_i.owner$	Device owner
pk	Public key	sk	Secret key
P_I	Infrastructure policy	P_U	User policy
PR	Reconciled policy	CT	Contract
CS	Contract Server	TS	Trusted Server
DK	Domain Knowledge	LM	Log Manager
P_i^h	High-level policy	P_i^d	Device-level policy
S_i^h	Subject	C_i^h	Policy context
R_i^h	Policy resource	A_i^h	policy action
C_i	Device context	S_i	Device state
A_i	Device action	ts	timestamp
$State$	Hardware configuration	$Action$	Active operation
O	Data object set	o	Data object

3.1 High-Level Policies

A high-level policy or application-level policy defines the context (conditions) under which an IoT system can collect and process users data using high-level concepts and properties described using the application model. The policy defines which *subject* can perform what *actions*, on which *resources* of users or the infrastructure, and under what *context*. We model a high-level policy P_i^h as a tuple $\langle S_i^h, C_i^h, R_i, A_i^h \rangle$, where S_i^h is the *subject* of the policy, C_i^h is the *context* in which the policy applies, R_i is the *resource* subject to access restriction, and A_i^h is the *action* the subject S_i^h is allowed to performed on the resource R_i. The high-level policy model can be interpreted as follows.

> *If the context C_i^h is true, permit the subject S_i^h to perform the action A_i^h on the user or infrastructure resource R_i.*

For instance, consider the high-level policy "Allow smart concierge application to acquire Bob's fine grained location for directions." This policy can be mapped to the high-level policy model as follows.

> *If using for directions, permit the smart concierge to access Bob's fine grained location data.*

The context dictates what an subject/application is permitted or not permitted to do with users data. The context itself is modeled as a *set of conditions* on contextual variables including location, time, and activity. A contextual condition is a boolean function on the observations generated from the devices payloads.

3.2 Device-Level Policies

We consider two types of device policies – *device state policies* that restrict the states a device can sustain (as per the policy) in different conditions, and *device payload policies* that specify which actions must be executed on the payload generated by the device. A device state policy P_i^d is modeled as a triple $\langle D_i, C_i, S_i \rangle$, where D_i is the device to which the policy P_i^d applies, C_i is a *context* in which the policy applies, and S_i ($S_i \in D_i.states$) is the *state* of the device. The device state policy is to define that the device must be in state S_i if the context C_i is *true*. Similarly, a device payload policy P_i^d is modeled as a triple $\langle D_i, C_i, A_i \rangle$, where D_i is the device to which the policy P_i^d applies, C_i is a *context* in which the policy applies, and $A_i = a_{i1}a_{i2}\ldots a_{im}$ is a sequence of actions $a_{ij} \in D_i.actions$ that must be applied to the payload if the context C_i is *true*.

In both, state and payload policies, the context dictates what the device is allowed to do in the IoT space. The context itself is modeled as a *set of conditions* on contextual variables including location, time, and activity. A contextual condition is a boolean function on the payload of the device D_i (specified in the policy) or the payload of other device(s) in the IoT infrastructure. For a given policy P_i^d, we refer to the device D_i for which the policy is specified as $P_i^d.device$, and context C_i as $P_i^d.context$. For $P_i^d.context$, we further define $P_i^d.context.devices$ as a set of devices $\{D_j\}$ whose payloads are referred to in $P_i^d.context$.

3.3 Contracts

A contract is an agreed and signed policy between the IoT system and one or more users, which represents an undeniable agreement between them. On the one hand, users use high-level policies (preferences) to restrict access to their private data. On the other hand, the IoT infrastructure uses high-level policies to define users data collection and processing practices. An *agreed high-level contract* is the result of reconciling users preferences and an infrastructure policy, which is agreed by both parties, users and the infrastructure. We model a contract between a user U_i and the infrastructure I as

$$CT = [P_j^h, Sign_I(P_j^h, DK_l), Sign_{U_i}(P_j^h, DK_l)],$$

where P_j^h is the agreed policy, DK_l the DK to use for the translation, $Sign_I$ the infrastructure's signature, and $Sign_{U_i}$ the user's signature.

Before agreeing to a policy, a negotiation process needs to happen between the system and users (Ramakrishna, 2008; Sun et al., 2017). This negotiation process not only needs to be interactive, but also needs to accommodate the dynamic changes in the IoT system. The negotiation process may involve trade-offs wherein both, a user and the system, has to give away some privileges in order to gain some benefits. For example, in a smart concierge application, users may have to give away location privacy to receive other services such as receiving mails related to work. Note that, since our goal in this study is about contract translation and translation attestation, we will not describe any solution for policy negotiation.

Our attestation of contracts translation requires verifiable domain knowledge, which in turn requires verifiable device states and actions logs. In the following section, we describe our tamper-proof audit logs mechanism for ensuring that all logs are verifiable.

3.4 Tamper-Proof Audit Logs

A tamper-proof logging mechanism enables the creation of audit logs that cannot be tampered without detection. The logs are kept consistent with respect to the time they were recorded. We consider the case of untrusted devices, but connected to a trusted server for generating the tamper-proof audit logs. In addition, we consider a trusted contract server for generating public/private key pairs and for creating and attesting contracts for entities in the IoT space. The trusted server securely encrypts the generated log events based on the contracts between users and infrastructure.

4 SYSTEM OVERVIEW

We explain a brief overview of our IoT framework regarding policy negotiation, translation and attestation. The overall framework comprises four layers: Infrastructure, System, Application, and Trust. The core of the framework comprises the following seven main components: *User Devices*, *Infrastructure Devices*, *Contract Server*, *Trust Server*, *Log Manager*, and *Domain Knowledge Manager*.

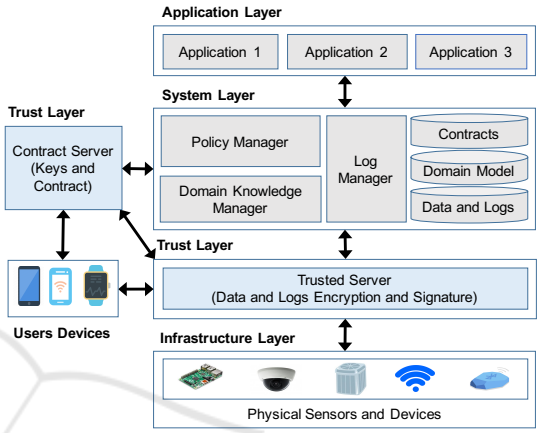

Figure 1: IoT System High-Level Design.

The *Infrastructure Devices* component contains all the stationary and mobile devices D_I owned and controlled by the IoT infrastructure, which are used to sense the IoT environment. Recall that, for each policy P_j^d, a subset $P_j^d.context.devices$ (*Context Devices*) of D_I is used to check the conditions that form the context $P_j^d.context$ in which the policy P_j^d must be in effect. The *User Devices* component comprises all the user devices (D_U) registered with the IoT system such as cellphones or tablets. These devices are used to identify users in the system. Through these devices, users interact with the system to define their data collection preferences, sign mutual contracts with the infrastructure, and attest contracts translation. The *Trusted Server* component is used to generate audit logs from the device payloads and encrypt the logs before sending them to *LM* for storage. The *Contract Server* (*CS*) is a third party entity responsible for reconciling infrastructure policies and users' preferences into agreed policies and mediating the signing of the agreed policies into mutual contracts through a contract negotiation mechanism. The *Log Manager* (*LM*) component stores contracts negotiated between users and the infrastructure, logs of devices states and actions. The logs are used during the remote attestation to verify contracts translation adherence. The *Domain Knowledge Manager* component manages the rapidly

changing domain knowledge, i.e., static and dynamic data produced by the sensors and about the sensors. It is also used to retrieve *DK* during the contract translation and attestation of contract translation.

4.1 Trust Model

Our trust model here is unidirectional such that each type of device has to possess *trustworthy attributes* in order to gain trust of the system. In particular, we can enumerate the course of trust building as below:

- **Assumption 1** (Trust origination): All devices must be validated initially through a registration phase. In this phase, devices produce trustworthy attributes, i.e., device identity, functionality, intended durability, purpose, permission, etc. After the verification of these attributes, the device gains an initial trust from the system manager.
- **Assumption 2** (Trust expansion): Beyond the initial phase of trust origination, all devices are then allowed to function (as verified for) and gradually build the trustworthiness through verifiable logs, with respect to the verifying controller system.

We provide the details of the trust origination as the device setup phase in Section 6. We assume that there must exist a trust building protocol as an auxiliary module. For example, there exists a variety of checkpoint protocols that enable the fault tolerance property. A checkpointing protocol would record the essential snapshots (in the form of device activity logs) of the system over the course of time. Therefore, in case of attestation the system would assure an intact trustworthiness until most recent snapshot in the past was checkpointed accurately. However, we do not propose any checkpointing method in this paper. We assume that by integrating an efficient checkpointing protocol we can leverage the trust expansion from the point of trust origination. We considers attack scenarios that have the potential to be queried out by a malicious infrastructure against the contract translation, and device states and action logs, which can result in compromising users private data.

- **Scenario 1** (Interconnected devices). In this scenario, we consider the set of devices that interconnects with peer devices across the network. In particular, the presence of these devices can be detected hence verified through the unique identity used for communication over wireless radio channels. For example, a device would use a physical or MAC address and connect through a WiFi AP. In this case, the AP can securely bind a communication activity across the channel with respect to a unique MAC identity. Therefore, the payload at the AP can be used to detect and prove the presence of a specific device. Our contract translation and attestation schemes consider only the interconnected devices and guarantee the consistency claims regarding *DK*.
- **Scenario 2** (Standalone devices). In this scenario, we consider the set of devices that remain standstill and does not connect to the network. In particular, we consider the portable devices that can sense the environment while residing as a passive device. Clearly, the aim is to sense/capture any sensitive information without sharing any information about itself or the environment. Therefore, the presence of these devices must be revealed so that each device is accountable regarding any sensitive information that it captures.
- **Scenario 3** (Unfound devices). In this scenario, we consider the set of devices that *intentionally delay* the trust origination phase, i.e., initial registration with the host network, even if the device possess trustworthy attributes as mentioned above. The small interval for which a device was not registered with the system, the device was still present inside the environment similar to a standalone device. In this case, there is no accountability for this interval in which device acted passively, i.e., what if the device had first recorded some sensitive information and registered later on as a device abiding to policy? The detection of a delayed registration is not straightforward without using device activity claims from the device hardware itself. In this paper, we do not consider this scenario in which device can record sensitive information in a subliminal manner.
- **Scenario 4** (Active threat to attestation of contract translation). In this scenario, we consider an active threat to the contract translation, which can actively manipulate *DK*, i.e., changing *DK* (information about device layout, positioning, counting, identity, functioning, status, etc.) to affect the policy actuation. In this case, the active adversary gains the access to *DK* server (or already has the access) and performs a malicious write on the domain knowledge server before the read command (from policy manager) to actuate a fabricated policy. The adversary either misuses the access privileges or manipulates the access privileges (to wrongfully gain access to otherwise inaccessible *DK*).

5 HIGH-LEVEL POLICY TRANSLATION

Let us consider our high-level policy model (P_i^h), device policy model (P_j^d), and the domain knowledge (DK) of the IoT space, as explained in Section 3. The goal is to map the set of elements of the source policy P_i^h to the set of elements of one or more of the destination policy P_j^d using DK. This requires finding the set of policy devices ($\{D_1, D_2, \ldots, D_m\}$) and/or set of virtual devices ($\{V_1, V_2, \ldots, V_n\}$) that the policy applies to. For each policy device (D_i) or virtual device (V_i), find its contextual conditions (C_i), context devices (D_C), and state (S_i) or actions (A_i). In the following, we describe the main steps of our approach for the translation. We use the example of Figure 2 to illustrate the process. In this example, Bob's policy statement "*Do not video record me when I am in room 2065.*" is translated into our high-level policy model as follows:

If [(User.id = "Bob") AND (User.location = "room 2065")] THEN (Concierge.Action = "No_Video") ON (Owner.Resource = "Bob_Face").

Explicitly, we have the following mappings to the elements of the policy model: (S_i^h = "*Concierge*"), (C_i^h = "*room*2065"), (A_i^h = "*No_Video*"), (R_i = "*Bob_Face*"), ($P_i^h.C\_attribute$ = "*Location*").

This policy is translated into two device policies using the knowledge of the smart building space.

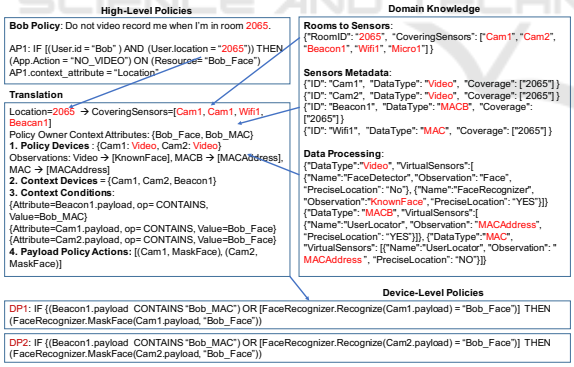

Figure 2: Policy Translation Example.

Step 1: Finding Policy Devices. Recall that a high-level policy P_i^h is defined in terms of a context attribute $P_i^h.C\_attribute$ such as location and activity. In our running example of Figure 2, we have $P_i^h.C\_attribute$ = "*Location*". Devices subject to policy are those which are located at the location specified at $P_i^h.C_i^h$ and which can capture data (payload), which can be enriched to produce the high-level observation/resource $P_i^h.R_i$. In our running example, $P_i^h.R_i$ corresponds to "Bob_Face" and "Bob_MAC", and $P_i^h.C_i^h$ to "room 2065". Then, the policy devices will correspond to all video cameras (which capture video that can be processed to retrieve Bob's face), which are $\{Cam1, Cam2\}$. The result of this step is a set of policy devices $\{P_i^h.device\}$.

Step 2: Finding Context Devices. Given a policy device D_j from Step 1 and the high-level policy context attribute $P_i^h.C\_attribute$, we first determine the set of context devices C_j for D_j by finding the set of all virtual devices V in $DK.vdevices$ that can produce values $P_i^h.C\_value$ for the policy owner. In our example, $P_i^h.C\_value$ = "*Bob_Face*" is produced by *FaceRecognizer*, and $P_i^h.C\_value$ = "*Bob_MAC*", is produced by *PeopleLocator*. So, $V = \{FaceRecognizer, PeopleLocator\}$, We then, identify all physical devices D_C in $DK.devices$, which collect data needed by the virtual devices in V. In our example, $D_C = \{Cam1, Cam2, Wifi1, Beacon1\}$. The result of this step is a list of context devices ($P_i^d.context.devices$) for each device policy D_j.

Step 3: Generating Context for Policy Devices. Given a policy device $P_j^d.device$ and a set of context devices $P_j^d.context.devices$ associated with it, the context of P_j^d is defined as set of conditions, where each condition is a boolean function of the form $\langle Attribute\ \ Operation\ \ Vale \rangle$, where $Attribute = D_l.payload$, $Operation = D_l.payload.operation$, and $Value = P_i^h.C\_value$. In our example, the set of policy devices is $\{Cam1, Cam2\}$ and the set of context devices is $\{Cam1, Cam2, Wifi1, Beacon1\}$. The set of corresponding conditions for $Cam1$ is $\{Cam1.payload$ = "*Bob_Face*", $Beacon1.payload$ = "*Bob_MAC*"$\}$ and for $Cam2$ $\{Cam2.payload$ = "*Bob_Face*", $Beacon1.payload$ = "*Bob_MAC*"$\}$. Note that $Wifi1$ is not used. This is because alone, it cannot provide precise location of a user. The result of this step is a set of conditions for each device policy.

Step 4: Device Policy State and Actions. A device policy can be either state or payload policy. The type of policy to use is determined by the action A_i^h of the high-level policy P_i^h. Our domain knowledge includes a mapping between the actions of a given application to the states and actions of the devices. Using this mapping as a lookup table, for each application action, we can determine the corresponding state or set of actions for the policy devices. In our example, this mapping is follows. ($Concierge$, No_Video) corresponds to ($Camera.State$ = "*OFF*" or $Camera.Action$ = "*MaskFace*").

Contract Translation Algorithm. The policy translation algorithm (*ContractTranslate*) describes the overall approach of the contract translation. It

takes a high-level policy P_i^h and a domain knowledge DK as input, and then generates a set of device policies $\{P_1^d, P_2^d, \ldots, P_k^d\}$ as output, where k is the number of device policies and d is the device identity.

Algorithm 1: High-Level Contract Translation Algorithm Pseudocode.

$contractTranslate()$ {

1. Through the Log Manager (LM), the algorithm retrieves the high-level policy (P_i^h) and the current Domain Knowledge (DK_j).
2. Then it retrieves the $context$ from $P_i^h.context$.
3. If $C\_attribute$ is equal to "Location", then it retrieves the space ID, $Location.ID$, from $P_i^h.C_i^h$.
4. Using $Location.ID$ and DK, it runs the procedure, *findPolicyDevices()* [**Step 1**], which outputs the list of policy devices,
   $PD = \{D_1, D_2, \ldots, D_m\}$,
   $PD = findPolicyDevices(Location.ID, DK)$.
5. For each policy device D_j in PD
   (a) It runs the function, *findContextDevices()* [**Step 2**], to find the set of context devices,
   $CD_j = \{D_1, D_2, \ldots, D_m\}$,
   $CD_j = findContextDevices(P_i^h, DK, D_j)$.
   (b) It runs the function, *generateContextConditions()* [**Step 3**], to generate contextual conditions C_j,
   $C_j = generateContextCondition(P_i^h, CD_j, D_j)$.
   (c) It determine the type of device policy to generate using A_i from P_i^h and DK.
   (d) For device state policy, it runs the function, *getState()* [**Step 4**], to output the permitted state S_j,
   $S_j = getState(A_i, P_i^h, DK)$.
   (e) For device payload policy, it runs the function, *getActions()* [**Step 4**], to output the actions allowed to be run on $D_j.Payload$.
   $A_j = getActions(A_i, P_i^h, DK)$
   (f) It then composes the device policy P_j^d for D_j,
   $P_j^d =< D_j, C_j, S_j >$ for device state policy or
   $P_j^d =< D_j, C_j, A_j >$ for device payload policy.
6. It returns the sets $\{P^d\}$ and $\{CD\}$.

}

6 ATTESTATION OF CONTRACT TRANSLATION SCHEME

A high-level contract is an agreed and signed high-level policy between the IoT system and users. Since the translation of a high-level policy to a set of device policies is controlled by the IoT system, our goal is to ensure that the translation of a signed policy is accountable and verifiable through DK. More precisely, our solution guarantee the integrity of the translation process in a retroactive manner such that any privacy violations are detectable. The overall translation attestation schema consists of sequential phases, i.e., setup phase, log collection phase, attestation of domain knowledge phase, attestation of policy translation phase.

6.1 Setup Phase

During the setup phase, all entities (E) involved with the IoT system are first registered with the Contract Server (CS) viz., Trusted Server (TS), Infrastructure (I), set of Infrastructure Devices (D_I), and set of User Devices (D_U). We assume that each user U_i has a dedicated device D_i ($D_i \in D_U$) such as a smart phone to interact with the IoT space. We use Public Key Infrastructure (PKI) to secure communications between all entities in the IoT system. Initially, we assume that CS has a certificate with a public/private key pair ($CS.cert, CS.sk, CS.pk$), and has distributed the public key $CS.pk$ to all entities in E. This enables all entities to encrypt all messages with $CS.pk$ before sending them to CS. CS plays the role of the PKI and we assume that it cannot be tampered with.

Entities Registration to CS. After an entity E_i ($E_i \in E$) sends a request for registration to CS, CS generates a certificate with a public/private key pair ($E_i.cert, E_i.sk, E_i.pk$) and sends back to the entity. Then, CS distributes the key $E_i.pk$ to all other entities. This certificate is revoked or renewed only by CS. After all the entities are registered with CS, then secure channels are created between all entities.

Users Registration to Infrastructure. Following the creation of secure channels between all entities is the registration of the user devices D_U with I. Each D_i of a user U_i sends its certificate ($U_i.cert$), public key (pk_{D_i}), and unique identity ($D_i.id$) to I for registration viz, ($D_i.id, U_i.id, U_i.cert, D_i.pk$). We assume that the infrastructure devices (D_I) are not powerful enough to handle certificates management and this task is left to I. Upon receiving the request for registration, I verifies the public key $D_i.pk$ of D_i using $U_i.cert$. Then, the infrastructure registers D_i and sends an acknowledgment message back to D_i. The registration enables the infrastructure to create a user to device identity binding and to uniquely identify the user. In addition, all registered D_I will be recorded inside DK for a subsequent contract translation procedure.

6.2 Device States and Actions Logs Collection

We use TS to generate and secure logs of states and actions of all devices in the IoT space. TS is responsible for authenticating all devices using their identities, generating audit logs using their payloads, and encrypting logs before forwarding them to LM. A continuous stream of data produced by each device D_i is collected by TS to generate the log format as $e^j_{D_i}$=$(j, D_i.id, D_i.state, D_i.action, ts)$, where i is the device's index, j the log sequence number, $D_i.id$ the unique device identity, $state$ the device state, $action$ the action performed by the device, and ts the timestamp. TS then encrypts each log entry using the key K_l ($encrypt(K_l, e^j_{D_i})$), indexes the encrypted log before forwarding the resulting tamper-proof audit log to LM for storage as $e^j_{D_i}$=$(seq, D_i.id, ts, encrypt(K_l, e^j_{D_i}))$, where seq is the global sequence number.

6.3 Attestation of Domain Knowledge Integrity

Ensuring the integrity and consistency of the translation algorithm requires the verification of current domain knowledge DK regarding resource addition, modification, and deletion from the IoT system. DK contains information about the IoT space ($DK.space$), users ($DK.users$), devices ($DK.devices$), and virtual devices ($DK.vdevices$). We first assume that DK_i at time t_i is verified. We also assume a high-level policy P^h_l was translated at time t_j using DK_j. Then, we use logs (Log) of states and actions of devices to check the integrity of DK.

The general idea behind domain knowledge attestation is to check currently available device logs and then search inside those logs for any *foreign device* trails by using methods (Srinivasan et al., 2008) such as wireless fingerprinting, Domain Name Server (DNS) query, traffic rates, etc. A *foreign device* is the one that is not registered as part of DK. However, it is cumbersome to quantify the scope, i.e., both spatially and temporally, of searching the dynamically streaming logs over the channel and the DK logs, respectively. The spatial scope of search inside the dynamic streaming of logs is to localize a physical space, meaning, if more than half of the infrastructure devices belongs to a hall/office/cabin then the system can speculate on the state of the rest of the devices in that same physical space; at least as long as the system collects enough evidences. Similarly, the temporal scope of the search inside the DK logs is to search for a time-proof (tp) verifying the latest updates as a closest link to previous history. In particular, (tp) provides a time bound on the staleness of DK, the higher is the deviation from the current time, the higher is the chance that contracts translated during that period ($t_{current}$ - t_{tp}) were inconsistent.

6.4 Attestation of Contract Translation

We perform the attestation of the translation of a high-level policy P^h_i to a set of device policies $(P^d_1, P^d_2, \ldots, P^d_m)$ by replaying the contract translation algorithm using the associated, verified domain knowledge DK_l. The overall attestation process comprises the following verification tasks: (1) Check the integrity of the translation procedure, and (2) Check the integrity of the translated device policies as advertised by the IoT system.

Attestation of Contract Translation Algorithm. We assume that whenever the IoT system translates a policy, it will inform the Contract Server (CS) about DK used and resulting device policies (P^{d*}), and mutually sign DK and P^{d*} with CS. The algorithm for contract translation ($ContractAttest$) receive inputs as a high-level policy (P^h_i), verifiable domain knowledge (DK_j), a hash MAC on the most recently upgraded version of translation algorithm ($ContractTranslate$), the log of devices (Log_j), and the set of device policies (P^{d*}) translated by the system. It then outputs either contract violated or not violated.

The followings summarize the steps of $ContractAttest$

Algorithm 2 : Attestation of High-Level Contract Translation Algorithm Pseudocode.

$attestContractTranslation()$ {

1. It runs the function for checking DK integrity
   $IntegrityDK = checkIntegrityDK(DK_j, Log_j)$
2. If ($IntegrityDK = FALSE$)
   (a) It returns "Violation"
3. ELSE
   (a) It runs the translation algorithm
   $contractTranslate(P^h_i, DK_j)$,
   to acquire the set of device policies,
   $P^d = \{P^d_1, P^d_2, \ldots, P^d_m\}$
   (b) It compares P^d and P^{d*}
   (c) If $P^d = P^{d*}$
   It returns "No Violation"
   (d) ELSE
   It returns "Violation"

}

The function *checkIntegrityDK()* implements the attestation of domain knowledge integrity as described in the above sections.

7 EXPERIMENTAL EVALUATION

We implemented prototypes of the contract translation algorithm and the attestation of contract translation algorithm as a module to be integrated with TIPPERS (Mehrotra et al., 2016). TIPPERS consists of a six-story campus building, which has classrooms, laboratories, conference rooms, offices, kitchens, and bathrooms. It is thoroughly instrumented with many types of devices (beacons, WiFi APs, video cameras, microphones) as well as various sensors (occupancy/motion, light, temperature and HVAC). It has a large infrastructure development component designed to allow plug-n-play integration of diverse security and privacy methods. It currently works under the assumption that infrastructure devices are trusted by users to collect data solely based on the policies established by the IoT environment, which makes it an ideal system for testing and validating real-time contract translation and attestation related experiments.

We performed a number of experiments to analyze the feasibility of our approach. In particular, we studied the performance overhead associated with the translation of contracts, the attestation of the contract translation procedure, the storage of encrypted logs. All the data sets used in the experiments are generated by TIPPERS.

7.1 Contract Translation Overhead

We performed two experiments to analyze the contract translation overhead: the impacts of the number of policy devices and the number of context devices. For the former, we fixed the number of context devices to 2 and vary the number policy devices as shown in Table 2. For each set of devices, we measured the time it took to run the contract translation (*ContractTranslate*) to translate one policy. Table 2 summarizes the results of this experiment.

Table 2: Contract translation execution time in function of the number of policy devices.

Policy Devices	**10**	**20**	**30**	**40**	**50**
Time (ms)	304	360	437	439	474
Policy Devices	**60**	**70**	**80**	**90**	**100**
Time (ms)	523	562	572	722	745

For the later, we fixed the number of policy devices to 2 and vary the number context devices as shown in Table 3. For each set of devices, we measured the time it took to run the contract translation (*ContractTranslate*) to translate one policy. Table 3 summarizes the results of this experiment.

The results of these two experiments indicate that the contract translation overhead is linear in function of the number of policy devices as well as the number context devices.

Table 3: *ContractTranslate* execution time in function of the number of context devices.

Policy Devices	**10**	**20**	**30**	**40**	**50**
Time (ms)	245	252	255	257	260
Policy Devices	**60**	**70**	**80**	**90**	**100**
Time (ms)	263	265	267	273	274

7.2 Attestation of Contract Translation Overhead

In this experiment, we analyzed the computation overhead associated with the attestation the contract translation algorithm, *ContractAttest*. For a given *DK*, we considered a varying number of device states logs to use to attest the translation of a given contract as shown in Table 4. For each set of logs, we recorded the running time of *ContractAttest*. The results of this experiment are summarized in Table 4.

The result of this experiment also shows that the computation overhead associated with the *ContractAttest* algorithm is linear in function of the number of device states logs.

Table 4: *ContractAttest* execution time in function of the size of the device states logs.

Logs	**10,000**	**20,000**	**30,000**	**40,000**
Time (ms)	643	847	1,023	1,133
Logs	**50,000**	**60,000**	**70,000**	**80,000**
Time (ms)	1,244	1,391	1,534	1,617

7.3 Log Generation and Storage Overheads

In this experiment, we analyzed the overheads associated with encrypting logs and storing the encrypted logs. We considered a set of 5 devices and their workloads collected for a period of 6 hours. For each device, we measured the time it took to create logs from the payload and encrypt the logs. We also measured the size of the logs. Table 5 summarizes the results of the experiment.

From these results, we can observe that the average time to process a log entry is about 0.32 ms. We can also observe that the average size of an encrypted log entry is 128 bytes. Both are not very significant.

Table 5: Device states and actions logs generation and storage overheads.

Device	Logs	Time (ms)	Size(KB)
Device 1	20,601	8,839	2,636
Device 2	17,247	5,157	2,208
Device 3	17,251	5,018	2,208
Device 4	17,252	4,963	2,208
Device 5	17,242	5,149	2,207

8 RELATED WORK

IoT frameworks are generally modeled through ontology creation (Wang et al., 2012; Nambi et al., 2014). In (Wang et al., 2012), IoT domain ontology is developed, which facilitate IoT service discovery, resource management, service testing, composition, and adaptation. Similarly, in (Nambi et al., 2014), an ontology based unified semantic knowledge base is introduced that captures the complete dynamics of the entities. It supports semantic definition and representation of IoT entities, dynamic service discovery and matching based on user request, service composition and orchestration in dynamic environments. However, we emphasize on data abstraction regarding privacy-policy through mutual contract creation and a remote attestation to verify policy enforcement in a retroactive manner.

In our model, the IoT infrastructure collecting and storing the logs is not fully trusted. Therefore, we propose a method of policy translation and the attestation of translation through log verification. Secure audit logging techniques (Schneier and Kelsey, 1999; Waters et al., 2004; Ma and Tsudik, 2009; Chadwick and Fatema, 2012) detect any tampering with the logs and guarantee integrity property. Basically, audit logging techniques cannot prevent tampering of the logs, rather, they guarantee that any attempts in tampering of the audit logs can be detected provably. In general, cryptographic methods ensure that logs stored by a logging facility remain intact. However, the log processing must be precise to maintain the efficiency during integrity verification. For example, in (Waters et al., 2004) hash chains are used for integrity protection and identity-based searchable encryption.

Remote attestation is being considered as viable mechanism for ensuring trust in IoT devices. It can be used to establish a static or dynamic root of trust in embedded and cyber-physical systems such as IoT systems (ElDefrawy et al., 2017). Three approaches have been proposed in remote attestation: hardware-based (based on the security provided by a Trusted Platform Module (TPM), a secure co-processor), software-based (doesn't rely on any secure co-processor or CPU-architecture extensions), and hybrid (software-hardware co-designs) (ElDefrawy et al., 2017). In our IoTtrust framework, we consider a hybrid remote attestation, which is implemented through the *Contract Server* and *Trusted Server*.

The emerging blockchain technology makes use of a shared ledger between the members of a network to maintain an immutable record of every transaction being executed (Swan, 2015). A blockchain doesn't rely on a central authority or trusted intermediary to validate transactions; rather, a consensus protocol is used by its members to guarantee trust, accountability, and transparency across the network. The recent design of private blockchains includes the concept of "channel" to enable granular data isolation. Only authenticated peers to a specific channel can share data in that channel. However, our system model is different from the peer-to-peer model assumed in most of the blockchain based solutions.

9 CONCLUSION AND FUTURE WORK

In this paper, we propose contract translation and attestation of contract translation schemes, which ensure trust and confidence in the translation of data collection and processing contracts from the high-level of data abstraction to the device-level. Both, users and the IoT infrastructure, can undeniably check mutual contracts between them, which defines the norms of data capture about the users. The translation attestation depends on the audit logs generated during data collection. In order to guarantee that audit logs are tamper-proofed, our solution rely on a trusted server, which securely generates and encrypts the audit logs. To evaluate our proposed contract attestation and translation attestation schemes, we implemented them as a module to be integrated with TIPPERS (Mehrotra et al., 2016). The results of the preliminary experiments highlight the feasibility of the overall approach.

With the emerging innovations in processors' design technology, such as modern ARM processors, TrustZone and hardware virtualization are being incorporated into those processors (Winter, 2008; Choi et al., 2014). These hardware based security mechanisms enable secure computation at the hypervisor level of the processors (Mirzamohammadi et al., 2017). In the future, this technology can be used

to move the logs sealing inside the devices, thereby removing the need of a trusted server.

ACKNOWLEDGEMENTS

This material is based upon work supported by DARPA under grants FA8750-16-20021 and FA8750-15-2-0277.

REFERENCES

Aikins, S. K. (2016). Connectivity of smart devices: Addressing the security challenges of the internet of things. In *Connectivity Frameworks for Smart Devices: The Internet of Things from a Distributed Computing Perspective*.

Balaji, B., Bhattacharya, A., Fierro, G., Gao, J., Gluck, J., Hong, D., Johansen, A., Koh, J., Ploennigs, J., Agarwal, Y., Berges, M., Culler, D., Gupta, R., Kjærgaard, M. B., Srivastava, M., and Whitehouse, K. (2016). Brick: Towards a unified metadata schema for buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, BuildSys '16, pages 41–50, New York, NY, USA. ACM.

Chadwick, D. W. and Fatema, K. (2012). A privacy preserving authorisation system for the cloud. *Journal of Computer and System Sciences*, 78(5):1359 – 1373.

Choi, H.-M., Jang, C.-B., and Kim, J.-M. (2014). Efficient security method using mobile virtualization technology and trustzone of arm. *Journal of Digital Convergence*, 12(10):299–308.

ElDefrawy, K., Rattanavipanon, N., and Tsudik, G. (2017). Hydra: Hybrid design for remote attestation (using a formally verified microkernel). *arXiv preprint arXiv:1703.02688*.

Farooq, M. U., Waseem, M., Khairi, A., and Mazhar, S. (2015). A critical analysis on the security concerns of internet of things (iot). *International Journal of Computer Applications*, 111(7).

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation of Computer Systems*, (7):1645–1660.

Han, S. N., Lee, G. M., and Crespi, N. (2014). Semantic context-aware service composition for building automation system. *IEEE Transactions on Industrial Informatics*, 10(1):752–761.

Ma, D. and Tsudik, G. (2009). A new approach to secure logging. *ACM Transactions on Storage (TOS)*, 5(1):2.

Madakam, S. and Date, H. (2016). Security mechanisms for connectivity of smart devices in the internet of things. In *Connectivity Frameworks for Smart Devices: The Internet of Things from a Distributed Computing Perspective*.

Mehrotra, S., Kobsa, A., Venkatasubramanian, N., and Rajagopalan, S. R. (2016). Tippers: A privacy cognizant iot environment. In *IEEE Pervasive Computing and Communications Workshops*.

Mirzamohammadi, S., Chen, J. A., Sani, A. A., Mehrotra, S., and Tsudik, G. (2017). Ditio: Trustworthy auditing of sensor activities in mobile & iot devices. In *The 15th ACM Conference on Embedded Networked Sensor Systems (SenSys 2017)*.

Nambi, S. N. A. U., Sarkar, C., Prasad, R. V., and Rahim, A. (2014). A unified semantic knowledge base for iot. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 575–580.

Pappachan, P., Degeling, M., Yus, R., Das, A., Bhagavatula, S., Melicher, W., Naeini, P. E., Zhang, S., Bauer, L., Kobsa, A., Mehrotra, S., Sadeh, N., and Venkatasubramanian, N. (2017). Towards privacy-aware smart buildings: Capturing, communicating, and enforcing privacy policies and preferences. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 193–198.

Ramakrishna, V. (2008). *Policy management and interoperation through negotiation in ubiquitous computing*. University of California, Los Angeles.

Schneier, B. and Kelsey, J. (1999). Secure audit logs to support computer forensics. *ACM Transactions on Information System Security*, 2:159–176.

Srinivasan, V., Stankovic, J., and Whitehouse, K. (2008). Protecting your daily in-home activity information from a wireless snooping attack. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pages 202–211.

Sun, Y., Wu, T. Y., Li, X., and Guizani, M. (2017). A rule verification system for smart buildings. *IEEE Transactions on Emerging Topics in Computing*, 5(3):367–379.

Swan, M. (2015). *Blockchain: Blueprint for a New Economy*. O'Reilly Media, Inc., 1st edition.

Varadharajan, V. and Bansal, S. (2016). Data security and privacy in the internet of things (iot) environment. In *Connectivity Frameworks for Smart Devices: The Internet of Things from a Distributed Computing Perspective*.

Wang, W., De, S., Toenjes, R., Reetz, E., and Moessner, K. (2012). A comprehensive ontology for knowledge representation in the internet of things. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1793–1798.

Waters, B. R., Balfanz, D., Durfee, G., and Smetters, D. K. (2004). Building an encrypted and searchable audit log. In *NDSS*, volume 4, pages 5–6.

Winter, J. (2008). Trusted computing building blocks for embedded linux-based arm trustzone platforms. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 21–30. ACM.

Yan, Z., Zhang, P., and Vasilakos, A. V. (2014). A survey on trust management for internet of things. *Journal of Network and Computer Applications*, 42:120 – 134.

Zhao, K. and Ge, L. (2013). A survey on the internet of things security. In *Computational Intelligence and Security (CIS), 2013 9th International Conference on*, pages 663–667. IEEE.