On Continuously Verifying Device-level Functional Integrity by Monitoring Correlated Smart Home Devices

Shiva Sunar
Concordia University
Montréal, Canada
er.shiva.sunar@gmail.com

Paria Shirani
University of Ottawa
Ottawa, Canada
pshirani@uottawa.ca

Suryadipta Majumdar
Concordia University
Montréal, Canada
suryadipta.majumdar@concordia.ca

J. David Brown
Defence Research and Development Canada
Ottawa, Canada
DAVID.BROWN10@forces.gc.ca

ABSTRACT

The correct functionality (can also be called as *functional integrity*) from a smart device is essential towards ensuring their safe and secure operations. The functional integrity of a device can be defined based on its correctness in sensing and actuating on the physical environment as well as in reporting to the users. As evident from several practical threats (e.g., event spoofing attacks, event masking attacks, sensor failure, vulnerabilities, and misconfigurations), this functional integrity of a device are often breached to cause severe security and safety impacts to their users. To make things worse, such integrity breaches might stay stealthy (due to their non-existence at the user-side) as well as be caused from both devices and apps (due to their vulnerability and misconfigurations at both physical and cyber spaces). Existing works mainly focus on detecting specific attacks without aiming at verifying functional integrity as a security property. In this paper, we bridge this gap by proposing a continuous approach for smart homes to verify functional integrity at the device-level while monitoring correlated devices. Specifically, our main idea is to learn the correlations among various sensors and actuators in a smart environment, and continuously monitor all the correlated devices to verify functional integrity breaches against various real-world attacks, including spoofing, masking, sensor failure, and device misconfigurations/vulnerabilities. We implement our approach in the context of smart home and evaluate its effectiveness (e.g., for sensors, R2 score of 0.98, and for actuators, accuracy up to 100%) using a public dataset.

CCS CONCEPTS

• **Security and privacy → Systems security**.

KEYWORDS

IoT security, smart home, correlated device, functional integrity

WiSec '24, May 27–30, 2024, Seoul, Republic of Korea

ACM ISBN 979-8-4007-0582-3/24/05…$15.00
https://doi.org/10.1145/3643833.3656132

ACM Reference Format:
Shiva Sunar, Paria Shirani, Suryadipta Majumdar, and J. David Brown. 2024. On Continuously Verifying Device-level Functional Integrity by Monitoring Correlated Smart Home Devices. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '24), May 27–30, 2024, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3643833.3656132

1 INTRODUCTION

Smart devices with the promises of many automation opportunities and convenience are being adopted on a large scale [22]. This growing expectations include the heavy reliance on the correctness of those device functionalities to manage automation in various critical infrastructures (including intimate and personal environments such as smart homes). However, due to many reasons (e.g., cheap and unreliable sensors [11, 13], misconfigurations [22], vulnerabilities [6], attacks [35, 37, 40]), the so called *functional integrity* of those IoT devices (i.e., correct measurement/action and correct reporting; defined in Section 2) might be compromised. As a consequence of such integrity breaches, the safety and security of mass population might be affected as it involves life threatening impacts (e.g., arson, intrusion, and other hazardous smart home conditions) [6, 22, 37].

The existing works (e.g., [8, 9, 14, 20, 21, 26, 27, 37, 40, 46, 47, 50, 54]) in smart homes mostly focus on detecting specific attacks on sensors and actuators in an IoT device. As a result, those works might find some of the integrity breaches, but they do not focus on continuously verifying the functional integrity in those devices regardless of focusing on the specific causes of integrity breaches. A detailed review of the related work is presented in Section 6. Specifically, the prominent works on detecting event spoofing and event masking attacks (e.g., [9–11, 13, 20, 37, 46, 47, 54]) in smart homes might be used to verify actuator integrity mainly if it is breached by those attacks. The other works (e.g., [8, 11, 13, 14, 21, 26, 27, 50]) on sensor failure detection might be used to verify sensor integrity relying on either redundant devices (i.e., multiple instances of same type of sensor) or dedicated devices (e.g., surveillance cameras). However, for their different objectives, each of those existing works either: (i) does not cover the entire functional integrity property (as detailed in Section 2), (ii) does not provide a continuous verification support, or (iii) is infeasible due to the overhead of deploying additional sensors and/or limited to specific attacks. Using the following

example use case, we further highlight the limitations of existing works and provide the motivation for our proposed solution.

Motivating Example. A typical smart home scenario, including a smart home environment with devices and apps (on the left) and a remotely located homeowner (on the right), is depicted in the upper part of Figure 1. Under normal operation (not shown in the figure), the actual status (`open`) of a smart door is the same as its reported status to the homeowner. However, due to a vulnerability (e.g., CVE-2023-50124 [2]) in the door, the reported status is wrong (`close`); as a result, that breaches the functional integrity of a smart door (a.k.a. *actuator integrity*) and hides a burglary. Also, a smoke detector wrongly reports its status (`no_fire`) during an arson (due to an attack in fire app similar as in [15]), a thermostat wrongly reads a temperature (`23°C`, instead of `40°C`), due to a failure in the thermostat [15]; both breach the functional integrity in those sensors (a.k.a. *sensor integrity*), and might lead to an arson and unauthorized intrusion, respectively, in a stealthy manner (i.e., not leading an obvious trace at the user end). Thus, the overall functional integrity in smart home devices might be breached. Due to the possibilities of such desperate situation, the homeowner needs to verify the functional integrity in a smart home.

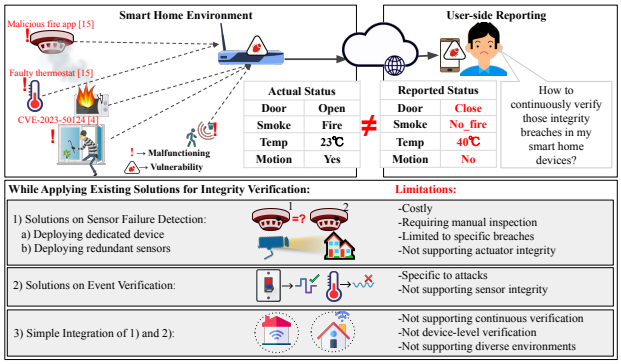

Figure 1: A motivating example showing functional integrity breaches in a smart home and how existing solutions approach them

The lower part of Figure 1 shows the limitations of applying the existing works for functional integrity verification as follows.

- *Applying sensor failure detection solutions for sensor integrity:* The existing sensor failure detection approaches (e.g., [8, 11, 13, 14, 21, 26, 27, 50]) might be able to detect some sensor integrity breaches (e.g., reporting `no_fire` status when there is an arson). However, those solutions cannot detect the other sensor integrity breaches, like reporting `23°C` when the actual temperature is `40°C`, as well as any actuator integrity breaches. Also, they require either redundant sensors that might incur additional cost and overhead to the homeowner or a dedicated monitoring device, such as CCTV, that requires manual verification from homeowners and is limited to specific breaches (e.g., verified visually).
- *Applying event verification solutions for actuator integrity:* The existing event verification solutions (e.g., [9, 10, 37]) might be able to detect some actuator integrity breaches (e.g., reporting `close` status when the door is actually `open`). However, they are insufficient to verify sensor integrity breaches, such as reporting `no_fire` when there is an arson by a smoke detector sensor and wrong reading by a temperature sensor (`23°C` instead of `40°C`).
- *Simply integrating both existing approaches:* Moreover, a simple integration of those existing solutions is not enough, as there exist more challenges (that are not addressed by those works), including (i) the need of continuous verification (e.g., without waiting for the actuators to report an action), (ii) conducting device-level verification (where threats are stealthy enough not to target an entire smart home behavior), and (iii) lack of ground truth for validation in a single smart home, while risk of privacy leak in case of aggregating the data from multiple smart homes.

In this paper, we address the aforementioned limitations and propose an approach to continuously verifying the device-level functional integrity while monitoring correlated smart home devices. Specifically, we first identify the correlated devices in a smart home and extract influential features for continuous monitoring and integrity verification. Second, we learn selective features and parameters for continuous monitoring as well as two specific models for predicting for functional integrity verification. Finally, we continuously monitor selective features from a selective set of devices and only based on an early symptom (i.e., an initial mismatch from the integrity property) from monitoring, conduct functional integrity verification using the learned predictive models for both sensors and actuators. We implement our solution for a smart home network using Home Assistant [7] platform, and evaluate its effectiveness (e.g., accuracy and efficiency) using a public dataset.

Contributions. Our main contributions are as follows:

- As per our knowledge, this is the first framework to continuously verifying functional integrity (i.e., sensing integrity, actuating integrity, and reporting integrity) of a smart home device by monitoring correlated devices to complement the existing works that mostly focus on detecting specific attacks in a smart home.
- In designing this framework, we learn both sensor-sensor relationship (how a sensor can affect other sensors) and event-sensor relationship (how an event from an actuator can affect other sensors) so that we can monitor the correlated devices to offer continuous verification of functional integrity in those devices.
- We implement and evaluate three variants (centralized, isolated, and federated) of our solution in the context of smart home to facilitate its adaptation to diverse customized smart environments (e.g., with no privacy issues among them, with privacy issues among them, and with no similarity among them, respectively). The evaluation results show the effectiveness of our solution (e.g., the accuracy of up to 100% for verifying actuating integrity, and an R2 score of 0.98 for verifying sensing integrity).

2 PRELIMINARIES

This section provides necessary backgrounds and defines the functional integrity property for IoT devices as well as our threat model.

Smart Home Networks. A typical smart home architecture [39, 56, 57] (shown in Figure 12 in Appendix) with various transducers are connected to the smart hub either directly or through a cloud (using various low powered and low bandwidth protocols, such as ZigBee, Z-Wave, Thread, and Matter). Smart home transducers (which include both sensors and actuators) generate two types of data: sensor readings and event notifications from their actuators. Both sensor readings and event notifications are transmitted from devices to the users through smart hubs (i.e., home automation systems, such

as Home Assistant [7], openHAB [1], and OpenMotics [3]); from where this work collects the data (as described in Section 3), like most other related works (e.g., Maverick [35] and ARGUS [40]).

Relationships between Smart Home Devices. Our preliminary study (in Figure 13 in Appendix) shows that there is a relationship between the outputs of IoT devices; which indicates that any state of the physical environment (either reported as a sensor reading or an event by an actuator) might have effects on the sensor readings of other devices that are situated at the same environment. Figure 13 shows an example where `door-opening` and `door-closing` events have direct impacts on nearby accelerometer, gyroscope, and magnetometer sensors and less or no impact on pressure and humidity sensors. In our methodology (in Section 3), we automatically find the correlations between smart home devices and then leverage those correlated devices to verify their functional integrity.

Functional Integrity for Smart Home Devices. We define the functional integrity property for an IoT device based on the definitions in NIST 8228 [44]. The *functional integrity* property is to ensure the correctness of an IoT *device* in performing its intended *functions*. A *device* can be defined as an embedded system that consists of a set of sensors and actuators. The *functions* of a device are dictated by its sensors and actuators. Therefore, we divide the functional integrity property of a device in two following categories.

(1) **Sensor Functional Integrity:** The sensor integrity is to check the correctness of sensor functionality (i.e., sensing or measuring various aspects of a physical environment and converting it to a digital signal). This integrity includes both the (a) *correctness in sensing a physical environment* (e.g., a temperature sensor must sense temparature accurately, where "accurately" can be defined with some tolerance as shown in Section 4) and (b) *correctness in reporting sensor readings to the users*.
(2) **Actuator Functional Integrity:** The actuator integrity is to check the correctness of actuator functionality (i.e., converting a digital signal to various physical actions, such as emitting light, producing sound, locking a door). This integrity includes both the (a) *correctness in actuating on a physical environment* (following two rules: i) an actuator performs only the action that a command asks for, and ii) an actuator only performs an action when there is a command) and (b) *correctness in reporting an event to the users*.

We consider these integrity properties in our threat model.

Threat Model. This work primarily aims at continuously verifying functional integrity (as defined above) in smart home devices. By continuous, it means that this work does not only rely on the status reports from the sensors and actuators but also continuously monitor correlated sensors to verify any integrity breach in between reports. Even though our work is not to detect specific attacks (e.g., spoofing [9, 37], and masking [10, 37]), we can find a functional integrity breach resulting from those attacks (in addition to misconfigurations and vulnerabilities). For verification, we rely on the availability of the device data from the smart hubs (i.e., home automation systems, such as Home Assistant [7], openHAB [1], and OpenMotics [3]) that support multi-vendor IoT products. Our out-of-scope threats include the threats that might affect the correlations between the devices and we assume that attackers main intention is to breach the functional integrity of the devices and not to alter the correlation between them (where all other existing works also fail). Like other related works (e.g., [9, 10, 20]), we cover the scenarios where the attacker wants to remain stealthy from the users and do not take over all the devices (as in those other cases, even though an automated solution might be harder to offer, the attack would be more noticeable by a user) Additionally, any threats at the cyberspace (e.g., modifying the sensor and actuator data in transit or through a smart app) is beyond our scope and can be tacked by the solutions (e.g., [40, 57]) at cyberspace.

3 METHODOLOGY

This section describes our proposed methodology.

3.1 Overview

As shown in Figure 2, our methodology for continuously verifying functional integrity in IoT devices consists of three major steps, namely, *identifying correlated devices*, *learning*, and *monitoring and verifying*. First, we identify a set of correlated device for each sensor/actuator measurement/action, and extracts statistical features for each actuator action reported by a device using the sensor readings of its correlated devices (detailed in Section 3.2). Second, we learn for monitoring (not shown in Figure 2) as well as for verification by learning two predictive models for the sensor-sensor relationship (that captures how one sensor reading impacts the other correlated sensors) and event-sensor relationship (that captures how one event in a device impacts the other correlated sensors), respectively, (detailed in Section 3.3). Third, we continuously monitor correlated devices, and based on an initial mismatch from the integrity property, conduct functional integrity verification using the learned predictive models for both sensors and actuators (detailed in Section 3.4). We provide more details on each step as follows.

3.2 Correlated Device Identification

This step is to first identify correlated devices in a smart home and then extract the most influential features from those correlated devices to facilitate the following steps (in Sections 3.3 and 3.4).

Identifying Correlations among Devices. In this work, we first identify the correlation among the devices in a smart home so that later we can leverage the most correlated devices during our continuous verification step to keep it efficient without affecting accuracy. This step aims to find the sensors that (i) have correlation with other sensor readings, and (ii) are more affected by an actuator event. More specifically, in this step (as shown on the top left in Figure 2), we first collect sensor readings and actuator events for a period of time (similarly as in Figure 12). Second, for each target device, we calculate the correlation between a given target device and all other sensors. To that end, for sensor readings (which are continuous data), we leverage the Pearson correlation coefficients [38] and for actuator notifications (which are discrete events), we use the Point-biserial correlation coefficient [29]. Third, we consider all sensors that have a correlation coefficient higher than a threshold value (which is chosen by our evaluation in Section 4) as the correlated sensors to a target device. Finally, we identify a set of correlated sensors for each given target sensor and actuator event.

More formally: let $S_{sens} = \{S_1, S_2, \ldots, S_k\}$ be a set of sensor readings and $S_{evts} = \{S_{E1}, \ldots, S_{Em}\}$ be a set of actuator events. CS_{S_E} and CS_{S_T} contain a set of sensors that are correlated with

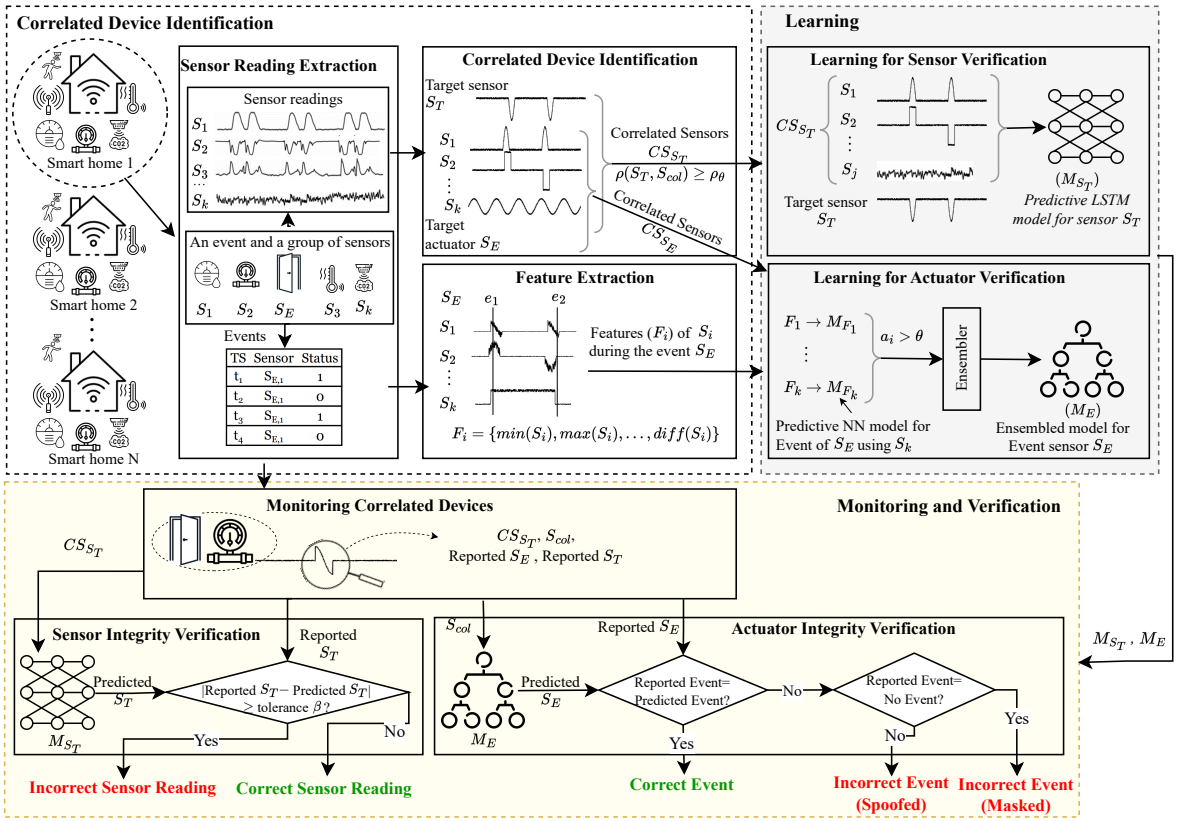

Figure 2: An overview of our proposed system. S_T represents sensors, S_E represents actuators, and e_i represents an event.

an actuator or a sensor, respectively. For a given target sensor, S_T and $\forall S_i \in \{S_{sens} - S_T\}$, S_i is correlated with S_T, if the absolute Pearson coefficients, $\rho_1(S_T, S_i)$, is greater than a threshold value ($\rho_{\theta 1}$), $\rho_1(S_T, S_i) \geq \rho_{\theta 1}$. Similarly, for a given target actuator, S_E and $\forall S_i \in \{S_{sens}\}$, S_i is correlated with S_E, if the Point-biserial coefficient, $\rho_2(S_E, S_i)$, is greater than a threshold value ($\rho_{\theta 2}$), $\rho_2(S_E, S_i) \geq \rho_{\theta 2}$.

Extracting Features. After identifying correlated devices, it is also important to find and extract most influential features that might provide the best representation of a correlated device during the learning step (in Section 3.3), as the effects of different events on a correlated sensor can be different. For example, the reading of magnetometer has opposite effects on two door events: increasing for `door-opening` at t_o and decreasing for `door-closing` at t_c, as shown in Figure 3. This difference in effect can be used to differentiate and identify various event types. To capture and quantify the effects of an event type on its correlated sensors, we extract various statistical features/metrics, including *minimum*, *maximum*, *mean*, *median*, *standard deviation*, *skewness*, *kurtosis* and *difference* (i.e., differences between the first and last data points of the event window).

To extract the features representative of a target event from its correlated sensor readings, we first consider two timestamps: *event timestamp* t_e, which indicates that an event e might have occurred and *non-event timestamp* t_{ne}, which means no event might have occurred at a given timestamp. Then, we select two time periods: the period before t_e is called *pre-event window*, and the period after t_e is called *post-event window*. The duration of these periods is obtained through our empirical studies as evaluated in Appendix. Finally, we extract the features for these two periods.

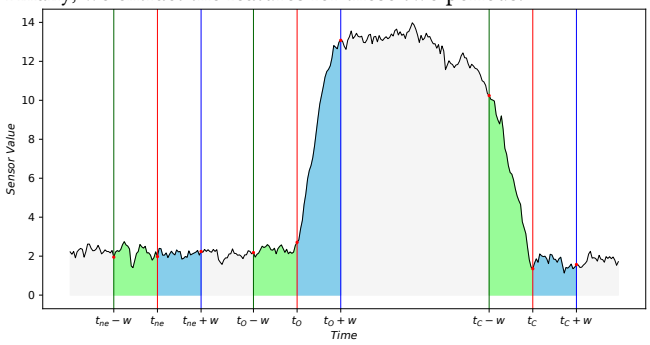

Figure 3: The sensor readings of a magnetometer correlated with `door` event during `door-opening` (t_o) and `door-closing` (t_c) events is different; it increases during `door-opening` and vice versa. The t_{ne}, and w represent a non-event timestamp and an event window length.

Example 3.1. Figure 3 shows the magnetometer sensor readings during two events occurring at event timestamps, t_o and t_c, where the *pre-event window* (e.g., $t_o - w$, t_o) is indicated by the area with light-blue shade, while the *post-event window* (e.g., $t_o + w$, t_o) is shown by the area with light-green shade, and w represents the *event window length*. Thus, for both pre/post event windows, we extract eight statistical features from the sensor readings.

More formally: Let $S_E \in S_{evts}$ be an event actuator and t_e be the timestamp for an event. Let $S_i \in S_{sens}$ be one of S_E correlated sensors, which is affected by S_E's event during a period of

w_i. The w_i is called the *event window length* of sensor S_i. The *pre-event window* (T^-) and the *post-event window* (T^+) at a given timestamp t_e are defined as $[t_e - w_i, t_e]$ and $[t_e, t_e + w_i]$, respectively. Let function, $F_{S_E,S_i}(T) = \{min(), max(), mean(), median(), standard\_deviation(), skewness(), kurtosis(), difference()\}$, extracts the features during an event time window T. The final set of features is calculated by $F_{S_E,S_i}(w_i) = F_{S_E,S_i}(T^-) \parallel F_{S_E,S_i}(T^+)$.

3.3 Learning

This section is to learn prediction models for both verifying functional integrity and continuously monitoring correlated sensors.

Learning for Verification. For verifying functional integrity using correlated devices, it is essential to predict sensor values or actuator events from other sensors. Therefore, in the following, we build two models (specifically trained for sensor and actuator verification, respectively) that will later be used for such predictions.

(i) Learning for Sensor Verification: This step finds how the readings from a specific sensor are related to other sensors' readings and how that relationship can be modeled to predict a sensor reading from others. For this purpose, we use the correlated devices identified in the previous step and create a model, M_{S_T}, which predicts the value of a target sensor S_T. Specifically, we train a long short-term memory (LSTM) model using all of its correlated sensors (S_i) obtained in Section 3.2. The reason behind using the LSTM model is that the sensor reading data is a time-series data and LSTM models are known to work well on time-series data, which can predict future values based on previous sequential data. Investigating other models (e.g., Transformers) is the subject of our future work.

(ii) Learning for Actuator Verification: This step finds how an action (a.k.a. *event*) from a specific actuator affects other sensors' readings and how that relationship can be modeled to predict an event from others. For this purpose, using the extracted features, $F_{S_E,S_i}(w_i)$, in the previous step, we create a model for each correlated sensor and then ensemble them to obtain a single model to predict an event from a specific actuator. To learn the relationships between the events (e) of an actuator, S_E, and each of its correlated sensor S_i, a model M_{F_i} is built for each $S_i \in CS_{S_E}$ (as defined in Section 3.2). A model M_{F_i} is trained to predict the events of S_E using the features generated with the data of sensor S_i for that event and its accuracy (a_i) is measured. For this purpose, we use feed-forward neural network (FFNN) since our input is a set of features and we aim to keep our learning overhead lightweight. Afterwards, the models with an accuracy of more than a threshold value (a_θ) are selected for the ensemble learning. Based on our analysis, we consider the threshold values of $a_\theta = 0.5$. On these selected models, we apply the weighted majority voting ensemble (WMVE) method [30] to combine them and generate an ensemble model (M_E). As such, the weight (α_i) of each model M_i is calculated as $\alpha_i = \log[\frac{a_i}{1-a_i}]$ [30]. Then, the ensemble prediction (p_{ens}) [30] is calculated as follows:

$$p_{ens} = \begin{cases} 1\,, if\ \left(\sum w_{pred=1} - \sum w_{pred=0}\right) > 0 \\ \\ 0\,, if\ \left(\sum w_{pred=1} - \sum w_{pred=0}\right) \leq 0 \end{cases}$$

where $\sum w_{pred=x}$ is the sum of weights of all the models that predicted $x \in \{0, 1\}$ for the event. The ensemble learning is described in Algorithm 1 in Appendix.

Depending on different smart home (and beyond smart home) setups (as discussed in Section 5), these learning steps can be implemented using different methods such as isolated, centralized and federated learning (as evaluated in Section 4).

Learning for Monitoring. To enable continuous monitoring of the correlated devices in the next step, the above-mentioned models are not sufficient, as they take more time to predict and continuous monitoring demands quick response in indicating an early suspect of change in sensor readings and actuator events. To address this issue, we alternatively classify event and non-event timestamps to quickly identify an event timestamp for which later we conduct verification (in Section 3.4). Therefore, in the following, we learn to answer the following two questions: which selective feature(s) from which specific sensor(s) are to be monitored and what would be the threshold value of the selected feature for timestamp classification.

(i) Selecting Devices and Features for Monitoring: To detect if an event of a target actuator (S_E) has happened at a certain timestamp (t), we need a function that takes the *event window length* of a correlated sensor at timestamp t and detects if t is an event or non-event timestamp. Hence since an event may have several correlated sensors, first we need to select one of them. As such, we consider a correlated sensor with the highest accuracy to predict its event timestamp. More specifically, the *event detecting sensor*, S_{ed}, returns a sensor $S_i \in CS_{S_E}$, where $S_i = \max_{S_i}(\alpha_i)$.

To later monitor a change in an actuator through sensor readings at a certain timestamp, we need a function with as few computations as possible to keep the process lightweight, as this step is performed very frequently. Thus, after identifying the *event detecting sensor*, we aim to find the best function from several candidate functions (e.g., *max(), min()*) to identify the type of a given timestamp (i.e., *event* or *non-event*). To this end, we first evaluate the Kernel Density Estimation (KDE) [41] for each of the candidate functions for the sets of event timestamps (t_e) and non-event timestamps (t_{ne}) using the sensor readings of the event detecting sensor (S_{ed}). We further confirm the results by their high Kolmogorov-Smirnov Score (KS). Then, we choose the most discriminating functions between the event and non-event timestamps to later use it during our device monitoring and verification step. More specifically, the *event detection function*, f_{ed}, returns the best function $f_i \in \{min(), max(), mean(), median(), standard\_eviation(), skewness(), kurtosis(), difference()\}$, where $f_i = \max_{f_i}(KS_n(KDE(n)))$, and n is a datapoint in the *event window length*.

Example 3.2. We collect a set of data from a correlated sensor and calculate the KDE of each candidate functions within its *event window length* for the sets of event timestamps and non-event timestamps. As shown in Figure 4, the *standard_deviation()* and *difference()* are the most discriminating functions between the event and non-event timestamps. This is also confirmed by their high Kolmogorov-Smirnov Score of 0.96, which determines the difference or similarity of distributions of two random variables. Finally, we select the *difference* as the *event detection function* (f_{ed}) due to its lower time complexity, i.e., $O(1)$ for the difference compared to $O(n)$ for the standard deviation.

(ii) Finding Threshold Value for Monitoring: This step is to select a threshold value for the above-identified feature to classify event and non-event timestamps. We observe that when there is no event

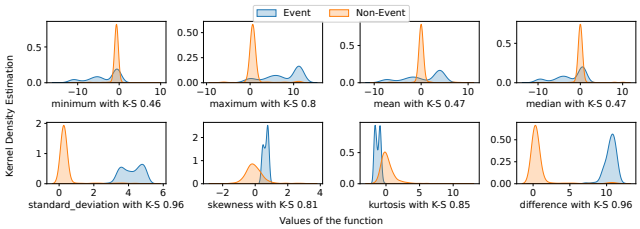

Figure 4: Kernel density estimation of each candidate function for "event" and "non-event" timestamps and their corresponding Kolmogorov-Smirnov scores. The *standard_deviation* and *difference* functions have the most non-overlapping distributions.

occurring in the event source (S_T), the readings of the event detecting sensor (S_{ed}) is uniform, which means that the average of the *difference()* function for non-event timestamps (μ_{ne}) within the event window length will be close to zero. However, the value of sensor data changes abruptly when an event occurs, which means that the average of the *difference()* function for event timestamps (μ_e) is much higher than zero. These observations can be seen in Figure 4. Thus, during our monitoring step, we use the *mean* of μ_{ne} and μ_e to separate the event and non-event timestamps. More formally: Let S_T be a target actuator with event window length of w, S_{ed} be its event detecting sensor, and f_{ed} be the *event detection function*. For each timestamp $t_{ne}, t_e \in w$, μ_{ne} = average of $f_{ed}(t_{ne})$ and μ_e = average of $f_{ed}(t_e)$. Thus, the threshold value to classify if a timestamp is an event or non-event of S_T is calculated as $\mu_{S_T} = \frac{\mu_{ne}+\mu_e}{2}$.

3.4 Monitoring and Verification

This section is to continuously monitor the correlated devices and based on monitoring result to verify functional integrity in a device.

Monitoring Correlated Devices. To detect an indication of a change in an actuator event (S_E), we continuously monitor its detecting sensor (S_{ed}) within the even window length w using sliding window techniques. At each timestamp, t, (depending on the reporting interval for each S_{ed}, as discussed in Section 5), the event detection function (f_{ed}) is computed for S_{ed} (i.e., $f_{ed}(S_{ed}(t))$. If its value is greater than μ_{S_E} (i.e., the μ_e specific to event S_E), then this change is considered as a potential integrity breach in actuator S_E within period of $[t-w, t+w]$, and t is considered as an indication of an integrity breach. As a result, we conduct a verification for timestamp t conducting the following step.

Verifying Functional Integrity. In the following, we describe both sensor and actuator integrity verification steps.
(i) Verifying Sensor Integrity: The value of the sensor readings of a given target sensor is predicted using the sensor readings of its correlated sensors. The predicted value and the reported value of the target sensor are compared and if their absolute difference is more than a threshold value (β) then the reported value is considered to be incorrect. For each sensor, the value of threshold β_i is adjusted on the basis of how much error a user may tolerate and should be tuned as per the individual use cases as described in Section 4. More formally: Let S_T be a target sensor and CS_{S_T} be a set of its correlated sensors. Let $S_{T,pred} \leftarrow P_{ens}(CS_{S_T})$ be the predicted value through our model, and $S_{T,rep}$ be the reported value of S_T. If $|S_{T,pred} - S_{T,rep}| > \beta$ then the reported value is considered to be *incorrect*, where β is a predefined tolerance threshold value.

Example 3.3. As shown in Figure 5, there are four sensors, *magnetometer*, *light* sensor, *temperature* sensor and *pressure* sensor (noted as S_{sens}={mag, $light$, $temp$, $press$}) and we aim to predict the value of *pressure* sensor (meaning $S_T = S_{press}$). First, we calculate the correlation between *pressure* and each of the remaining sensors as follows: $\rho(press, mag) = 0.6$, $\rho(press, light) = 0.7$, and $\rho(press, temp) = 0.8$. Suppose that the threshold value of ρ_θ to choose the correlated sensor is 0.65. Then, the set of its correlated sensors are *light* and *temperature*: $CS_{press} = \{light, temp\}$. Thus, the ML model is trained using correlated sensors $CS_{press} = \{light, temp\}$ as input and then it is used to predict *press* as output ($S_{T,pred}$). If $|S_{T,pred} - S_{T,rep}| > \beta$, then the reported value ($S_{T,rep}$) is considered to be incorrect.

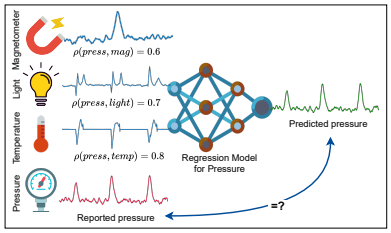

Figure 5: An example for the prediction of *pressure* sensor readings utilizing its correlated sensors and the regression model. Only correlated *light* and *temperature* sensors are used for regression.

(ii) Verifying Actuator Integrity: We describe the two steps for event verification as follows.
Predicting Events from Timestamp. This step is to predict the occurrence of an event at a given timestamp through its correlated sensors. The timestamp (t) is either obtained from a reported event or from the previous monitoring step for unreported events. To this end, the set of feature $F_{S_E,S_i}(w_i)$ are extracted for each correlated sensor S_i and then are fed into their corresponding models (M_i) which in turn produces their corresponding predictions (p_i). Obtained prediction results are combined as described in Section 3.3 to produce ensemble prediction (p_{ens}).

Example 3.4. Suppose there are three sensors, *accelerometer*, *magnetometer* and *gyroscope*, with their respective models $\{M_{acc}, M_{mag}, M_{gyro}\}$ to predict the door event (e) with accuracy $\{a_{acc} = 0.7, a_{mag} = 0.8, \text{and } a_{gyro} = 0.9\}$ and their predictions $\{p_{acc} = 1, p_{mag} = 0, \text{and } p_{gyro} = 1\}$. The weight of each model is calculated by $\alpha_i = \log[\frac{a_i}{1-a_i}]$: $\alpha_{acc} = 0.847$, $\alpha_{mag} = 1.38$ and $\alpha_{gyro} = 2.19$. We then calculate the sum of weights, $\sum w_{pred=1} - \sum w_{pred=0} = 0.847 - 1.386 + 2.197 = 1.658 > 0$. The ensemble prediction (p_{ens}) is one, meaning that the predicted event (E_{pred}) is door-open.

Matching Reported and Predicted Events. The predicted event (E_{pred}) is compared with the reported event (E_{rep}). If both indicate the same event, then the reported event is considered as *correct*. Otherwise, the reported event is flagged as *incorrect*. This incorrect reporting of an event is later further analyzed to decide their specific integrity breach (as explained in the following subsection).

Specifically, we utilize our defined event/non-event timestamps for this purpose. If E_{rep} indicates no event and E_{pred} indicates an event, then the predicted event (E_{pred}) is considered to be the breach of *correct reporting* property, whereas if $E_{rep} = some\_event$ but $E_{pred} = no\_event$ then the E_{rep} is considered to be a breach of *correct actuating* property. Further by matching with the command, we verify if it is a breach of *exact command* and *only by commmand* sub-properties.

Verification Results. Table 1 summarizes how our solution verifies the functional integrity property (covering all sub-properties) defined in our threat model (in Section 2). Specifically:

- *Sensor functional integrity:* This property is ensured by verifying two sub-properties (*correct sensing* and *correct reporting*) for sensors. The *correct sensing* sub-property might be breached by sensor failure (e.g., [11, 13]) and is verified by following the specific steps of our methodology as shown in the table. Also, the *correct reporting* sub-property might be breached by masking and spoofing sensor readings (e.g., [10, 20, 37, 57]) in addition to sensor failure and is verified by the steps in the table (where the step in parenthesis is only needed for masking-like breaches).
- *Actuator functional integrity:* This property is ensured by verifying two sub-properties (*correct actuating*, which further covers the case where actuator actions are exactly same as the command (*exact command*) and only triggered by a command (*only by command*) and *correct reporting*) for actuators. The *exact command* and *only by command* sub-properties in a actuator might be breached by device malfunctions (e.g., [6, 25]) are verified by following the steps mentioned in the table, respectively. Also, the correct reporting sub-property might be breached by masking and spoofing attacks (e.g., [10, 20, 37, 57]) in actuators and is verified by the steps mentioned in the table (where the step in parenthesis is only needed for masking-like breaches).

4 EXPERIMENTS

Experimental Setups. We conduct our experiments on a Windows 10 host machine with Intel i7 10700 CPU and 32 GB of memory. The *Docker* 4.17 was used to containerize the development environment. *Python* 3.8.10 was used for coding, *PyTorch* 1.13.1 was used to implement machine learning, *Flower* 1.3.0 library was used for federated learning, and *sdv* 0.18.0 was used to generate synthetic data. We implement our learning steps (as discussed in Section 3.3) leveraging three methods: isolated, centralized and federated. *Isolated learning* is where the model for each smart home is trained individually. *Centralized learning* is where one model is built by combining historical sensor data from multiple smart homes. *Federated learning* is where the model for each smart home is built by aggregating the learning results from multiple smart homes.

Dataset: To conduct the experiments, we utilize the Peeves real-world dataset* [9] containing *sensor data* from 48 sensors and event data from 12 different *event sources* and 12 raspberry pis collected over the period of 12 days in an office environment. A subset of Peeves dataset (which includes event notifications and sensor readings) are used in our experiments. The raspberry pi_8 senor data also includes three derived values (*yaw*, *pitch*, *roll*) that are utilized in our experiments. We exclude those raspberry pis that do not have event sensor data from our experiments. To prepare our datasets for different learning models, we spilt our data into different training and testing sets. As for sensor readings verification, the data is divided into 75% of train data and 25% of test data. The train data is further partitioned equally into three train sets for three smart homes (A, B and C). With respect to the event data, the number of events or data-points in our dataset is low not being sufficient for the training or testing of three smart homes. Thus, we leverage the Synthetic Data Vault (SDV) framework [4] and generate synthetic train dataset for smart homes A, B and C using the original data. We then use the original data as the test set. We utilize *SDMetrics* library to evaluate the generated synthetic data.

Description of Used Models: As for event notification verification, we use a typical feed forward neural network (FFNN) with fully connected layers and the *Cross_Entropy* loss function. With respect to the sensor readings verification, we utilize an LSTM neural network with the *MSE* loss function. In both models, the *Adam* optimizer with `lr=0.01`, `betas=(0.9,0.999)`, `eps=1e-8`, `weight_decay=0`, `amsgrad=False` are used.

Prediction Accuracy for Sensing Integrity. To measure the performance of the prediction model for sensor readings (discussed in Sections 3.3 and 3.4), we measure the R2 score between the reported and the predicted values, as it is more informative than other metrics for regression analysis evaluation [12]. Figures 6 and 7 show the R2 scores between the reported and predicted values for various correlated sensors attached to the Pi_8 and Pi_{10} raspberry Pis, respectively. As shown, in this set of experiments, we consider isolated, centralized, and federated learning for each scenario. The results of different machine learning models are very close for each sensor. We observe the highest R2 score of 0.98 in Figure 6 and 1.0 in Figure 7; which demonstrates the benefit of using these sensors in our verification.

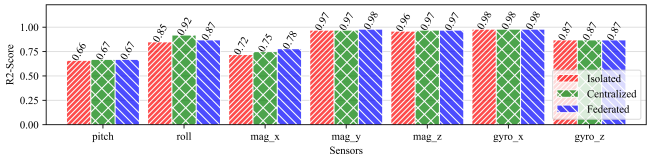

Figure 6: R2 score of reported data and predicted values for various sensors of Pi_8.

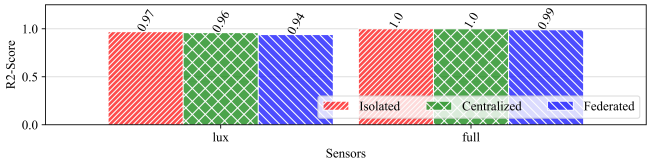

Figure 7: R2 score of reported data and predicted values for various sensors of Pi_{10}.

Prediction Accuracy for Actuating Integrity. To examine the accuracy of our actuator event verification solution (discussed in Section 3.3 and 3.4), we select `door`, `window`, `fan`, `fridge`, and `light` actuators as our target event sources. For each actuator, we first predict its events (e.g., `door-open/door-close`, `window-open/window-close`, `fan-on/fan-off`, `fridge-door-open/ridge-door-close`, and `light-on/light-off`) using the trained model of each correlated sensor through the isolated, centralized and federated learning models. Then, we perform ensemble

*PEEVES dataset homepage

Properties		Sub-properties		Sample Breaches	Steps by Our Solution (Sections)
Functional integrity	Sensor	Correct sensing		Sensor failure [11, 13]	3.2.a → 3.3.a → 3.4.a → 3.4.b
		Correct reporting		Masking and spoofing [9, 10, 20, 37]	3.2.a → 3.3.a → (3.4.a) → 3.4.b
	Actuator	Correct actuating	Exact command	Device malfunction [6, 25]	3.2.b → 3.3.b → 3.4.c
			Only by command	Device malfunction [6, 25]	3.2.b → 3.3.b → 3.4.a → 3.4.c
		Correct reporting		Masking and spoofing [9, 10, 20, 37]	3.2.b → 3.3.b → (3.4.a) → 3.4.c

Table 1: Showing verification results of our solution.

learning using all the models having an accuracy of more than 0.5. We run each of the experiments ten times and the average accuracy of each individual correlated verification sensor and ensemble prediction is calculated. Obtained results for both individual and ensemble learning are presented in Table 2. As seen, in most of the cases, isolated learning shows lower accuracy than other two methods and centralized method obtains slightly higher accuracy than federated. Moreover, in some cases, one sensor data is sufficient to detect events (e.g., *mag_y* sensor to detect `door` events). We further measure the f1-score and obtain similar results. In what follows we provide our insights on the obtained results.

Door Events: Several sensors including the magnetometers *mag_x*, *mag_y*, gyroscope *gyro_x*, and rotation sensors (*yaw*, *pitch*, *roll*) can detect `door-opening` and `door-closing` events with an accuracy of almost %100. This is expected as these sensors are attached to the door itself. *Pressure* sensor can also predict the `door` events which implies that opening and closing the door changes the pressure inside the room. *Temperature* sensor predicts the `door` events with lower accuracy than aforementioned sensors. Opening the door brings in outside air which might be of different temperature. However, as the changes in temperature takes some time (whereas for other sensors changes are seen instantly), and the outside-inside temperature difference can be high sometimes but low other times. Therefore, the accuracy to predict `door` events might have been lower. Some sensors such as *humidity* sensor were not able to predict `door` events. This implies that opening or closing the door does not change the humidity of the room instantly, which might be due to the fact that the humidity outside the room was the same as inside, or even if it was different it would take some time much longer than our event window.

Window Events: The *sound* sensor (*rms* and *rms_db*) is the most reliable predictor of the `window` events (`window-opening` and `window-closing`) with an accuracy of around 99%. It may have been the case that the window produced sound while being opened and closed, or it can also be the case that outside was much more noisier than inside the room. The *light* sensor was also able to predict the `window` events with around 70%-80% accuracy. It might be the case that opening the window let more light come in from outside. *co2* and *voc* sensors were also able to predict the `window` events. One possible explanation could be the difference in the amount of the gas inside and outside the room. *Temperature* sensor in BME sensor module is able to predict the `window` events with higher accuracy (∼90%) than that of temperature sensor in MPU (∼70%). One possible explanation is that the BME sensor module might have been closer to the window. Other sensors such as *pressure* and *humidity* were not able to predict the `window` event as *humidity* take time to change a lot longer than our event window. Pressure inside and outside the room might have been the same. However, we achieve the highest accuracy of 100% using ensemble learning.

Fridge Door Events: Most of the sensors can predict `fridge` events (`fridge-door-opening` and `fridge-door-closing`) with high accuracy. For *pressure*, *temperature*, *accelerometer*, and *gyroscope* sensors, the results can be explained with reasoning the same as the `door` events. *Light* sensor (placed inside the fridge) might be able to predict it as opening the fridge door lets lights to get in from inside, and also from the light-bulb inside the fridge whereas closing the door will make inside completely dark. Inside the fridge is very humid compared to outside, therefore the *humidity* sensor might be able to pick that while opening and closing the fridge door.

Fan Events: Only the *sound* sensor is able to predict `fan` events (`fan-on`/`fan-off`) with the highest accuracy of 90%, while the *temperature* sensor could predict `fan` events with lowest accuracy. Other sensors in Table 2 are not able to detect it. This is expected as fan mainly produces sound without a major effect on other sensors.

Light Events: As expected, `light-on`/`light-off` events were reliably predicted with the light sensor with an accuracy of up to 99% using federated learning.

Prediction Accuracy for Reporting Integrity. We further evaluate the performance of our verification step (discussed in Section 3.4), that is used to check the correctness of reports from sensors and actuators. For each actuator, we select a time window according to its correlated sensor and monitor the corresponding timestamps to detect if there has been a missing reporting of an event. For each of those timestamps, we predict whether it is an event or a non-event timestamp using the method described in Section 3.3. Based on the true events and predicted events for the timestamps, we evaluate various performance metrics (accuracy, precision, recall and f1-score). Obtained results for the detection of `door`, `window`, `light` and `fridge` events are presented in Figure 8 per actuator. We achieve high score in all these performance metrics except for the `fan` events that we receive relatively lower accuracy due to less effects of `fan` events on its correlated sensors.

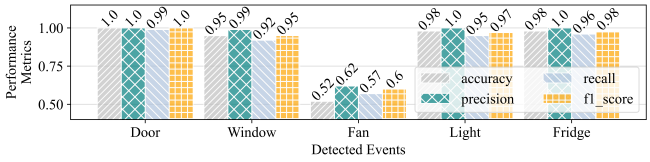

Figure 8: Performance of event detection for various actuators.

Impact of Parameters. *Event Window Length Selection:* To examine the effects of event window length (w) for event verification, we vary the length of windows starting from 0.5 to 5 seconds (using incremental steps of 0.5s) and measure the accuracy for each of the values. An example for the correlated sensors on the Pi_8 is shown in Figure 9. Window length with the highest accuracy is chosen as the final value of w; if multiple window lengths have the same highest accuracy, the longest window length is chosen for w. For instance, a window size of 3 is chosen for the *yaw* sensor, while the window size of 1 is selected for the *acc_y* sensor.

Tolerance Threshold Value Selection: To choose the value of the tolerance threshold (β), we measure the accuracy results for different values of β as plotted in Figure 10. If the chosen β is too high there will be too many false negative (i.e., largely incorrect sensor reading identified as correct), whereas if it is too low there will be too

Door	tempH	tempP	press	yaw	pitch	roll	mag_x	mag_y	mag_z	acc_x	acc_y	acc_z	gyro_x	gyro_y	gyro_z	Ensem
Isolated	0.50	0.50	0.50	0.74	0.51	0.65	0.69	1	**0.99**	0.75	**0.81**	**0.92**	0.99	0.90	0.78	0.98
Centralized	**0.75**	**0.84**	**0.96**	0.87	**0.94**	**0.99**	1	1	**0.99**	0.73	0.80	**0.92**	0.99	0.81	0.76	1
Federated	0.65	0.74	0.91	**0.94**	0.86	**0.99**	0.99	1	**0.99**	**0.80**	0.79	0.90	1	**0.91**	**0.85**	1

Window	temp	humidity	acc_x	acc_y	acc_z	gyro_x	gyro_y	gyro_z	temp	co2	voc	rms	rms_db	lux	full	Ensem
Isolated	**0.94**	**0.81**	0.89	0.89	**0.93**	0.74	**0.94**	**0.8**	0.55	**0.81**	0.8	0.97	0.98	0.69	**0.82**	0.99
Centralized	0.89	0.76	**0.90**	0.89	0.85	**0.82**	0.92	0.76	**0.77**	0.72	**0.82**	**0.99**	**0.99**	**0.81**	0.73	1
Federated	0.93	0.74	0.89	**0.90**	0.88	**0.82**	0.92	0.78	0.62	0.76	0.81	**0.99**	**0.99**	0.76	0.8	0.99

Fridge	temp	press	humidity	acc_z	gyro_x	gyro_y	gyro_z	lux	full	ir	Ensem
Isolated	0.91	1	**0.97**	0.96	0.6	0.90	**0.98**	**0.99**	**0.99**	0.96	1
Centralized	**0.98**	1	0.96	**0.97**	0.74	**0.95**	0.95	**0.99**	**0.99**	**0.97**	1
Federated	0.96	1	**0.97**	0.96	**0.76**	0.90	0.97	**0.99**	**0.99**	**0.97**	1

Fan	temp	rms	rms_db	Ensem	**Light**	lux	full	Ensem
Isolated	0.47	0.84	0.88	0.85	Isolated	0.77	0.82	0.64
Centralized	**0.72**	**0.88**	0.90	**0.89**	Centralized	0.87	**0.89**	0.54
Federated	0.61	0.86	0.88	0.87	Federated	0.91	0.84	**0.75**

Table 2: Comparing the accuracy results of different event detection models for each of the actuators and their correlated sensors. The values in bold show the highest accuracy per sensor for each learning model. The cells shaded in grey represent the highest accuracy for each actuator.

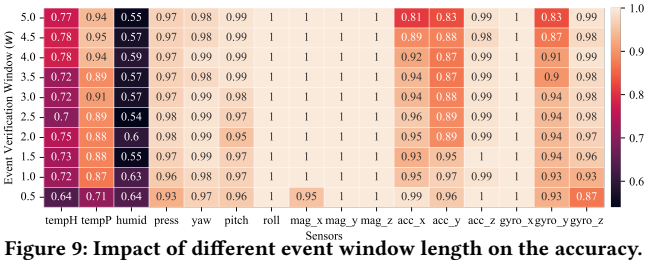

Figure 9: Impact of different event window length on the accuracy.

many false positive (i.e., almost correct sensor reading identified as incorrect). Thus, we need to select the minimum value of β without getting much false positive or giving up accuracy of the detection. For instance, the optimum values of β for door and fridge are 50% and 35%, respectively.

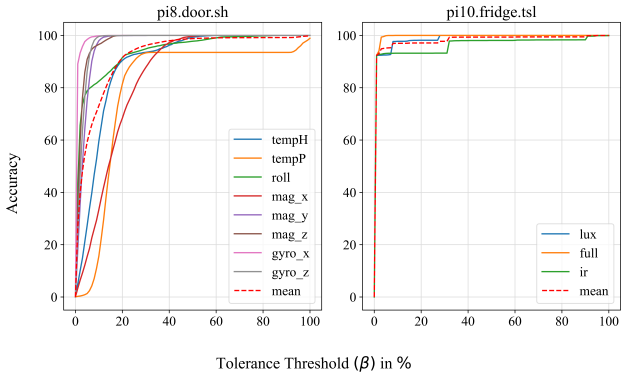

Figure 10: The impact of β on our sensor reading verification.

Resource Overhead. We further compare the CPU and memory overhead of federated and non-federated (Centralized) methods for training a model for door events verification. Obtained results shown in Figure 11 suggest that the federated learning uses around 1.57 (for *datapoints = 100*) to 2.1 (for *datapoints = 600*) times more CPU time utilizing almost the same amount of memory. With respect to the non-federated method, we use centralized because federated and centralized has the same number of data points whereas isolated has less number of data-points. It can be seen that the federated method uses around 1.57 (at *datapoints = 100*) to 2.1 (at *datapoints = 600*) times more CPU time and around 1.1 times more memory than the non-federated (centralized) method.

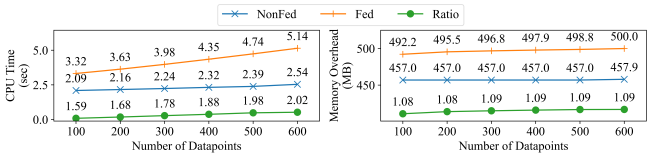

Figure 11: Comparing the CPU and memory overhead of federated and non-federated (centralized) methods for event verification.

5 DISCUSSION

Supporting Different Smart Environment Setups. Even though our solution is currently implemented for smart homes, our proposed approach is applicable to other similar smart environments (e.g., smart office, smart city). With the broader outreach, the diversity (especially in the nature of learning data) among multiple smart environments will also increase. Therefore, our current learning approach is implemented using three different techniques: centralized (where data from multiple smart environments are aggregated before learning), federated (where data from multiple smart environments are aggregated through learning), and isolated (where data from each smart environment is only used for learning) to support various scenarios. Specifically, centralized learning might be suitable for a smart city like scenario where data from its different components (e.g., buildings, transportation, etc.) can be aggregated without any privacy concerns. Federated learning might be suitable for an office environment where different departments within an organization might collaborate as long as their private data is not leaked. Finally, isolated learning might be useful for a smart environment where data sharing is impossible (due to privacy/security issues) or useless (due to significant difference in nature).

Varying Reporting Intervals. Based on the interval time for sensing in different IoT devices, the granularity of our verification step might vary. In most cases, this interval is chosen to optimize the energy efficiency of those devices. For our experiments, we chose the default interval time and no change in configuration was performed (to replicate a typical smart home environment where homeowners rarely modify its settings). However, in most current IoT devices, it is possible to configure their interval time; which can be used to adjust the frequency of our verification.

The Impact of Simultaneous Events. If two actuators are nearby then event in one actuator might result in a wrong unreported

event detection for the other, as described in [37]. For example, if a door and window are nearby and opening both of them produce similar effect on a sound sensor, then an incorrect reporting of the `door-opening` event might be undetected, if our solution would only be using sound sensor data. Using multiple sensor readings (as long as not affected by the other event(s)) and their ensemble predictions (as discussed in Section 3.3) might help to reduce this effect. In the future, we plan to develop [37] like solution for functional integrity verification to fully overcome this issue.

The Impact of Distance between Devices. Like most event verification system [9, 10], our actuator verification step also relies on how an event affects the sensors physically. This effect is typically dependent on the distance between an actuator and correlated sensors. Therefore, in our future work we intend to explore the effect of the distance between devices on our solution (inspired by [43]).

The Impact of Outliers and Poisonous Data. If the training data is poisoned by an adversary or includes outliers (e.g., opposite nature data from a smart home) then our learning step might be affected. One possible way to reduce this effect could be to assign weights to models based on their performance and then perform weighted aggregation of the models during the federation step.

6 RELATED WORK

Verification in Smart Homes. There exist several works (e.g., [9, 10, 20, 37] to verify smart home security. For instance, PEEVES [9] and Haunted-House [10] verify the correctness of the reported event notifications using the effects of an event on nearby sensors. However, unlike ours, they do not provide a verification system that can support all sensing integrity, actuating integrity, and reporting integrity. HAWatcher [20] constructs correlation rules for monitoring event and command manipulation, as well as detecting device malfunctions. However, it does not perform any sensor integrity verification. Ozmen et al. [37] on the other hand build event fingerprints to facilitate reporting integrity. However, they do not include the sensing integrity. ARGUS [40] supports actuator integrity by detecting spoofing events, masking events, spoofing commands, and masking commands. However, the authors have not evaluated its ability to verify sensing integrity. ThingsDND [11] proposes a context-aware method to detect and diagnose IoT device failures for multi-resident smart homes. *ElHady et al.* [18] detect sensor failure in ambient assisted living using association rule mining. However, they do not provide reporting integrity.

CLEAN [55] identifies faults in binary sensors in a smart home environment, but is unable to verify sensing integrity. Horus-Eye [17] is a two-stage anomaly detection system designed to manage the growing traffic from IoT devices. However, it dose not perform functionality integrity. Dice [13] detects incorrect sensor states and state transitions by learning the possible sets of states of all the sensors in a system and the probability of transition of states from one set to another during the learning phase. [18, 28, 34] detect sensor faults using the association rules which capture the relationship among sensors. However, they do not support actuating integrity and reporting integrity. Detectif [34] and Idea [28] detect missing events but their solution is only applicable to smart homes which are customized to infer activities of daily living (ADL), but it is not applicable in general to smart home scenario.

There are works [8, 21, 26, 27, 50] on the verification of the sensor readings, however, all of these solutions are based on either hardware redundancy or dedicated verification devices. For instance, the authors in [21] detect sensor faults by utilizing the data gathered by nearby sensors of similar types. There are several works (e.g., [14, 27, 49, 51]) that use camera to perform various security operations (such as intrusion detection, remote monitoring, etc.). However, for those works, users need to monitor captured videos to detect the events of interests, which can be challenging [33]. Additionally, those works can only detect attacks that have visual effects (captured by a camera).

Missing Event Detection in Smart Home. SecureHouse [32] detects `door` events by measuring the vibrations through a smartphone accelerometer mounted near the door. Similarly, [23] detects `user-passing` event primarily by measuring the changes in magnetic intensity using the on-board magnetometer sensor of a smartphone. The authors in [16] detect `door` events using the sound impulse generated by a smartphone and analysing the Doppler shift of the received response caused by the moving door. Some other works [31, 52, 53] detect events by utilizing spatial and temporal patterns in the sensor data (e.g., slope, jump, and spike). However, all of them only focus on detecting missing events.

Other Security Solutions for Smart Home. DÏoT [36] detects compromised IoT devices through network traffic packets. The SmartThings system is investigated in [19] where a number of vulnerabilities introduced by privileged smart home applications are discovered. One of those vulnerabilities enables event spoofing. Shuvo et al. [45] verify security policies on smart home devices using existing security standards. In [48], malicious mobile phone applications can infect home networks leaving smart home devices vulnerable to attacks from distant. However, none of those works verify sensor readings and event notifications in a smart home.

7 CONCLUSION

Even though the functional integrity property in an IoT device is critical for the secure and safe operations of smart homes, none of existing works in smart homes aim to verify this integrity property. In this paper, we proposed an approach to continuously verifying the device-level functional integrity by monitoring correlated smart home devices. To that end, we first learned the relationships among correlated devices based on various events and sensor readings. Then we leveraged this relationship to verify the correctness of reported sensor readings and event notifications as well as to detect unreported events. We implemented our solution in the context of smart home and evaluated its effectiveness using a public dataset. As future work, we plan to apply our method to other IoT applications such as autonomous vehicle and smart city, where verification of functional integrity is needed. Also, we plan to extend our device-level solution for the cyberspace to cover wider range of threats.

Acknowledgement. The authors thank the anonymous reviewers for their valuable comments. We also acknowledge Piyush Adhikari's help in representing the experiment results. This material is based upon work supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Department of National Defence Canada (DND) under the Discovery Grants RGPIN-2021-04106 and DGDND-2021-04106.

REFERENCES

[1] openHAB 2024. *Empowering the smart home.* openHAB. https://www.openhab.org/

[2] NIST 2024. *National Vulnerability Database - CVE-2023-50124 Detail.* NIST. https://nvd.nist.gov/vuln/detail/CVE-2023-50124

[3] OpenMotics 2024. *OpenMotics makes building automation relevant.* OpenMotics. https://www.openmotics.com/en/

[4] DataCebo 2024. *The Synthetic Data Vault.* DataCebo. https://sdv.dev/

[5] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. 2017. Internet of Things (IoT) communication protocols. In *2017 8th International conference on information technology (ICIT).* IEEE, 685–690.

[6] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security evaluation of home-based IoT deployments. In *IEEE Symposium on S&P.* 1362–1380.

[7] Home Assistant. 2024. Awaken your home. http://www.home-assistant.io/.

[8] David S Bayard and Scott R Ploen. 2005. High accuracy inertial sensors from inexpensive components. US Patent 6,882,964.

[9] Simon Birnbach, Simon Eberz, and Ivan Martinovic. 2019. Peeves: Physical Event Verification in Smart Homes. In *ACM CCS.* ACM, 1455–1467.

[10] Simon Birnbach, Simon Eberz, and Ivan Martinovic. 2022. Haunted house: physical smart home event verification in the presence of compromised sensors. *ACM TIOT* 3, 3 (2022), 1–28.

[11] Alireza Borhani and Hamid R Zarandi. 2022. ThingsDND: IoT Device Failure Detection and Diagnosis for Multi-User Smart Homes. In *2022 18th European Dependable Computing Conference (EDCC).* IEEE, 113–116.

[12] Davide Chicco, Matthijs J Warrens, and Giuseppe Jurman. 2021. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Computer Science* 7 (2021), e623.

[13] Jiwon Choi, Hayoung Jeoung, Jihun Kim, Youngjoo Ko, Wonup Jung, Hanjun Kim, and Jong Kim. 2018. Detecting and identifying faulty IoT devices in smart home with context extraction. In *48th Annual IEEE/IFIP International Conference on DSN.* IEEE, 610–621.

[14] Shruti Dash and Pallavi Choudekar. 2022. IoT-Based Smart Home Surveillance System. In *Applied Information Processing Systems.* Springer, 417–427.

[15] Wenbo Ding, Hongxin Hu, and Long Cheng. 2021. IoTSafe: Enforcing Safety and Security Policy withReal IoT Physical Interaction Discovery. In *Network and Distributed System Security Symposium.*

[16] Thilina Dissanayake, Takuya Maekawa, Daichi Amagata, and Takahiro Hara. 2018. Detecting door events using a smartphone via active sound sensing. *ACM IMWUT* 2, 4 (2018), 1–26.

[17] Yutao Dong, Qing Li, Kaidong Wu, Ruoyu Li, Dan Zhao, Gareth Tyson, Junkun Peng, Yong Jiang, Shutao Xia, and Mingwei Xu. 2023. {HorusEye}: A Realtime {IoT} Malicious Traffic Detection Framework using Programmable Switches. In *32nd USENIX Security Symposium (USENIX Security 23).* 571–588.

[18] Nancy E ElHady, Stephan Jonas, Julien Provost, and Veit Senner. 2020. Sensor failure detection in ambient assisted living using association rule mining. *Sensors* 20, 23 (2020), 6760.

[19] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In *IEEE symposium on S&P.* IEEE, 636–654.

[20] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. 2021. HAWatcher: Semantics-aware anomaly detection for appified smart homes. In *30th USENIX Security Symposium.* 4223–4240.

[21] Saurabh Ganeriwal, Laura K Balzano, and Mani B Srivastava. 2008. Reputation-based framework for high integrity sensor networks. *ACM TOSN* 4, 3 (2008), 1–37.

[22] Aniketh Girish, Tianrui Hu, Vijay Prakash, Daniel J Dubois, Srdjan Matic, Danny Yuxing Huang, Serge Egelman, Joel Reardon, Juan Tapiador, David Choffnes, et al. 2023. In the Room Where It Happens: Characterizing Local Communication and Threats in Smart Homes. In *ACM IMC.* 437–456.

[23] Liangyi Gong, Yiyang Zhao, Chaocan Xiang, Zhenhua Li, Chen Qian, and Panlong Yang. 2018. Robust light-weight magnetic-based door event detection with smartphones. *IEEE Transactions on Mobile Computing* 18, 11 (2018), 2631–2646.

[24] Grant Hernandez, Orlando Arias, Daniel Buentello, and Yier Jin. 2014. Smart nest thermostat: A smart spy in your home. *Black Hat USA* 2015 (2014).

[25] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart locks: Lessons for securing commodity internet of things devices. In *ACM ASIACCS.* 461–472.

[26] Mahdi Jafari and Jafar Roshanian. 2013. Inertial navigation accuracy increasing using redundant sensors. *Journal of Science and Engineering* 1, 1 (2013), 55–66.

[27] Lee Han Keat and Chuah Chai Wen. 2018. Smart indoor home surveillance monitoring system using Raspberry Pi. *JOIV* 2, 4-2 (2018), 299–308.

[28] Palanivel A Kodeswaran, Ravi Kokku, Sayandeep Sen, and Mudhakar Srivatsa. 2016. Idea: A system for efficient failure management in smart IoT environments. In *ACM MobiSys.* 43–56.

[29] Diana Kornbrot. 2014. Point biserial correlation. *Wiley StatsRef: Statistics Reference Online* (2014).

[30] Ludmila I Kuncheva. 2014. *Combining pattern classifiers: methods and algorithms.* John Wiley & Sons. 123 pages.

[31] Mo Li, Yunhao Liu, and Lei Chen. 2008. Nonthreshold-based event detection for 3D environment monitoring in sensor networks. *IEEE Transactions on Knowledge and Data Engineering* 20, 12 (2008), 1699–1711.

[32] Michael A Mahler, Qinghua Li, and Ang Li. 2017. SecureHouse: A home security system based on smartphone sensors. In *PerCom.* IEEE, 11–20.

[33] N Malarvizhi, Arun Kumar Dash, V Manikanta, and Athreayasa Kalyan. 2022. AI-Based Tracking System from Real-Time CCTV Captures. In *Artificial Intelligence and Sustainable Computing.* Springer, 739–747.

[34] Madhumita Mallick, Archan Misra, Niloy Ganguly, and Youngki Lee. 2020. DETECTIF: Unified detection & correction of IoT faults in smart homes. In *WoWMoM.* IEEE, 78–87.

[35] M Hammad Mazhar, Li Li, Endadul Hoque, and Omar Chowdhury. 2023. Maverick: An app-independent and platform-agnostic approach to enforce policies in IoT systems at runtime. In *ACM WiSec.* 73–84.

[36] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N Asokan, and Ahmad-Reza Sadeghi. 2019. DÏoT: A federated self-learning anomaly detection system for IoT. In *IEEE ICDCS.* 756–767.

[37] Muslum Ozgur Ozmen, Ruoyu Song, Habiba Farrukh, and Z Berkay Celik. 2023. Evasion attacks and defenses on smart home physical event verification. In *NDSS 2023.* NDSS.

[38] K Pearson. 1895. Notes on regression and inheritance in the case of two parents proceedings of the royal society of London, Vol. 58. , 240-242 pages.

[39] Abhay Kumar Ray and Ashish Bagwari. 2020. IoT based Smart home: Security Aspects and security architecture. In *IEEE CSNT.* IEEE, 218–222.

[40] Phillip Rieger, Marco Chilese, Reham Mohamed, Markus Miettinen, Hossein Fereidooni, and Ahmad-Reza Sadeghi. 2023. ARGUS: Context-Based Detection of Stealthy IoT Infiltration Attacks. In *USENIX Security.* 4301–4318.

[41] Murray Rosenblatt. 1956. Remarks on some nonparametric estimates of a density function. *The annals of mathematical statistics* (1956), 832–837.

[42] Álvaro San-Salvador and Álvaro Herrero. 2012. Contacting the devices: a review of communication protocols. In *Ambient Intelligence-Software and Applications: 3rd International Symposium on Ambient Intelligence (ISAmI 2012).* Springer, 3–10.

[43] Rahul Anand Sharma, Elahe Soltanaghaei, Anthony Rowe, and Vyas Sekar. 2022. Lumos: Identifying and Localizing Diverse Hidden {IoT} Devices in an Unfamiliar Environment. In *31st USENIX Security Symposium.* 1095–1112.

[44] V.K. Shen, D.W. Siderius, W.P. Krekelberg, and H.W. Hatch. 2019. *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks.* Technical Report. National Institute of Standards and Technology.

[45] Md Wasiuddin Pathan Shuvo, Md Nazmul Hoq, Suryadipta Majumdar, and Paria Shirani. 2023. On Reducing Underutilization of Security Standards by Deriving Actionable Rules: An Application to IoT. In *International Conference on Research in Security Standardisation.* Springer, 103–128.

[46] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. 2019. Aegis: A context-aware security framework for smart home systems. In *ACSAC.* 28–41.

[47] Amit Kumar Sikder, Leonardo Babun, and A Selcuk Uluagac. 2021. Aegis+ a context-aware platform-independent security framework for smart home systems. *Digital Threats: Research and Practice* 2, 1 (2021), 1–33.

[48] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. 2016. Smartphones attacking smart-homes. In *ACM WiSec.* 195–200.

[49] Tanin Sultana and Khan A Wahid. 2019. IoT-Guard: Event-driven fog-based video surveillance system for real-time security management. *IEEE Access* 7 (2019), 134881 – 134894.

[50] Thomas George Thuruthel, Josie Hughes, Antonia Georgopoulou, Frank Clemens, and Fumiya Iida. 2021. Using redundant and disjoint time-variant soft robotic sensors for accurate static state estimation. *IEEE Robotics and Automation Letters* 6, 2 (2021).

[51] Lingshan Xu, Xianghan Zheng, Wenzhong Guo, and Guolong Chen. 2012. A Cloud-based monitoring framework for Smart Home. In *4th IEEE CloudCom.* IEEE, 805–810.

[52] Wenwei Xue, Qiong Luo, Lei Chen, and Yunhao Liu. 2006. Contour map matching for event detection in sensor networks. In *ACM SIGMOD.* 145–156.

[53] Wenwei Xue, Qiong Luo, and Hejun Wu. 2012. Pattern-based event detection in sensor networks. *Distributed and Parallel Databases* 30, 1 (2012), 27–62.

[54] Rozhin Yasaei, Felix Hernandez, and Mohammad Abdullah Al Faruque. 2020. IoT-CAD: Context-aware adaptive anomaly detection in IoT systems through sensor association. In *ICCAD.* 1–9.

[55] Juan Ye, Graeme Stevenson, and Simon Dobson. 2015. Fault detection for binary sensors in smart home environments. In *PerCom.* IEEE, 20–28.

[56] Xiaojing Ye and Junwei Huang. 2011. A framework for cloud-based smart home. In *ICCSNT*, Vol. 2. IEEE, 894–897.

[57] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. 2018. HoMonit: Monitoring smart home apps from encrypted traffic. In *ACM CCS.* 1074–1088.

A BACKGROUNDS

A smart home is typically equipped with devices and appliances that are connected to and can be controlled remotely using a smartphone or computer via the Internet. A typical smart home architecture is shown in Figure 12, where various smart home transducers are connected to the smart hub. A hub is connected to the devices either through a third-party cloud or directly using various communication protocols.

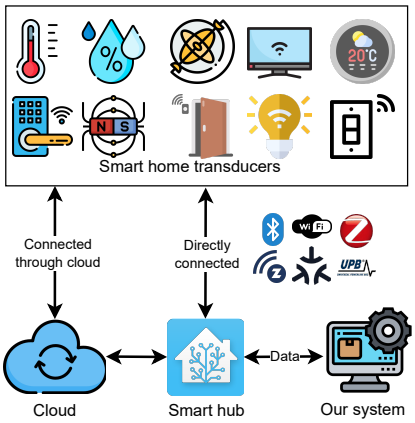

Figure 12: An overview of a smart home architecture [39, 56, 57].

The functionality of each IoT device might be different and according to their functionalities, the corresponding requirements of bandwidth, physical ranges, latency, and power consumption can vary. For example, a security camera might need a high bandwidth connection and its power consumption does not matter much, as it can be mains powered. But for a battery-powered device such as a thermometer or a contact sensor, low bandwidth is acceptable but they must be power efficient. Therefore, to cater to these different needs of different devices there are different communication protocols with their own strength and weakness. For low-powered and low-bandwidth use cases, there are protocols such as ZigBee, Z-Wave, Thread, and Matter. Whereas, for more bandwidth, there are protocols such as Bluetooth Low Energy (BLE), and for even more bandwidth at the cost of more power, there are protocols such as Wi-Fi [5]. There are also power-line communication protocols that use existing AC powerlines to communicate between devices such as X-10, UPB, and LonWorks [42]. Some IoT devices, nest thermostats, for example, store their data directly to the cloud platform of their manufacturers [24]. Different IoT devices using different protocols may not be able to communicate directly with each other. Home automation systems, such as Home-Assistant, openHAB, and Open-Motics, can be utilized to integrate all the devices in a single platform which enables them to communicate with each other, as shown in Figure 12.

Figure 13 shows an example showing the relationship between smart home devices which is leveraged in our solution (in Section 3).

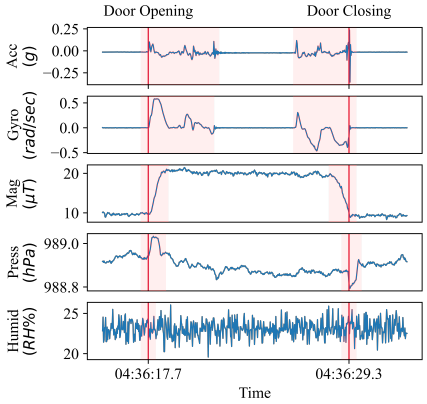

Figure 13: An example showing relationships between smart devices.

B ENSEMBLE LEARNING ALGORITHM

Event prediction using ensemble learning of event-sensor relationships is detailed in Algorithm 1.

Algorithm 1 Compute ensemble prediction

Input:
$\mathcal{A} \leftarrow$ *set of accuracies for each model,*
$\mathcal{P} \leftarrow$ *set of predictions of each model*
Output: $p_{ens} \leftarrow$ *ensemble prediction*

$sum \leftarrow 0$, $p_{ens} \leftarrow \emptyset$
for all (p_i, a_i) s.t. $p_i \in \mathcal{P}$ and $a_i \in \mathcal{A}$, **do**
  $\alpha_i \leftarrow \log[a_i/(1-a_i)]$
  if $p_i = 1$ **then**
    $sum \leftarrow sum + \alpha_i$
  else
    $sum \leftarrow sum - \alpha_i$
  end if
end for
if $sum > 0$ **then**
  $p_{ens} \leftarrow 1$
end if
return p_{ens}