One Key to Rule Them All: Secure Group Pairing for Heterogeneous IoT Devices

Habiba Farrukh*, Muslum Ozgur Ozmen*, Faik Kerem Ors, and Z. Berkay Celik
Purdue University
{hfarrukh, mozmen, fors, zcelik}@purdue.edu

***Abstract*—Pairing schemes establish cryptographic keys to secure communication among IoT devices. Existing pairing approaches that rely on trusted central entities, human interaction, or shared homogeneous context are prone to a single point of failure, have limited usability, and require additional sensors. Recent work has explored event timings observed by devices with heterogeneous sensing modalities as proof of co-presence for decentralized pairing. Yet, this approach incurs high pairing time, cannot pair sensors that sense continuous physical quantities and does not support group pairing, making it infeasible for many IoT deployments. In this paper, we design and develop IoTCupid, a secure group pairing system for IoT devices with heterogeneous sensing modalities, without requiring active user involvement. IoTCupid operates in three phases: (a) detecting events sensed by both instant and continuous sensors with a novel window-based derivation technique, (b) grouping the events through a fuzzy clustering algorithm to extract inter-event timings, and (c) establishing group keys among devices with identical inter-event timings through a partitioned group password-authenticated key exchange scheme. We evaluate IoTCupid in smart home and office environments with 11 heterogeneous devices and show that it effectively pairs all devices with only 2 group keys with a minimal pairing overhead.**

1. Introduction

Internet of Things (IoT) devices need secure wireless communication channels to protect the confidentiality and integrity of the data they exchange (e.g., sensor measurements, actuator states). This protection is critical to prevent various attacks (e.g., man-in-the-middle (MitM) [1], [2] and protocol manipulation [3], [4]), provide user privacy and ensure the trustworthiness of IoT systems [5], [6]. Therefore, IoT devices require a *pairing* mechanism, which establishes shared cryptographic keys between devices to enable secure wireless communication among them.

Traditional pairing methods employ a centralized approach where a user pairs each device with a trusted IoT gateway/hub through an external helper device (e.g., user typing a password on their smartphone to pair a smart light with the IoT hub). Yet, central gateways/hubs (a) are prone to temporary or permanent failures due to operational malfunctions [7], and (b) can be compromised due to their vulnerabilities [4], [8]. In such cases, devices need a secure mechanism for directly communicating with each other.

To illustrate, consider a smart home that turns on the lights and unlocks the door when smoke is detected for fire safety. If the hub is not present or experiences a failure, these devices cannot communicate with each other, resulting in severe consequences, e.g., residents being trapped in a fire.

Consequently, IoT platforms have recently been pushing towards decentralized IoT networking protocols (e.g., OpenThread [9]). Such decentralized protocols have applications in smart homes and industrial automation due to their reliability (always-on), scalability (easy device addition), and adaptability (supporting devices from different vendors). Past efforts at decentralized secure IoT device pairing have explored two primary approaches: (1) human-in-the-loop-based and (2) context-based pairing.

In human-in-the-loop-based approaches, a user needs to be physically involved to facilitate the pairing process. A line of work requires users to contact devices by exploiting the fact that the movement pattern is correlated in multiple devices [10], [11]. For instance, a user with a wristband touches a device [12] or shakes two devices at the same time for pairing [13]. Another line of work relies on the user to enter passwords, read QR codes, or press buttons [14]. For example, OpenThread requires a user to scan the QR code of each device with a mobile phone [9]. These approaches, however, require human involvement that affects usability and scalability with an increasing number of devices.

To address these limitations, there is a growing interest in context-based pairing schemes [1]. In this, co-located sensors establish shared keys based on the entropy extracted when they observe common events. Yet, these approaches are limited to pairing devices only equipped with homogeneous sensors that sense identical sensing modalities [5], [15], [16], [17]. For instance, to pair a power meter with a microphone, another microphone must be embedded into the power meter so that both devices capture a common audio context.

Recent work has proposed capturing event timings among heterogeneous devices as evidence for device co-presence to derive secure keys [6]. While this approach supports heterogeneous sensing types without human intervention, it suffers from four fundamental limitations: (1) takes several hours or a few days for pairing, (2) is limited to sensors that only sense instant physical quantities (e.g., cannot pair widely deployed temperature and humidity sensors), (3) entire

*. Habiba Farrukh and Muslum Ozgur Ozmen contributed equally.

pairing process is impaired by concurrent events (e.g., when the sound from door lock and coffee machine overlaps), and (4) can pair solely two sensors at a time. These limit its usability, make it infeasible for pairing diverse device types and ultimately fail in offering promise for adoption to a wide-variety of IoT deployments in practice.

In this paper, we design and develop IoTCupid, a secure group pairing system for IoT deployments with heterogeneous sensing modalities. IoTCupid complements trusted gateways in IoT deployments when they experience operation failures or are compromised and enables secure communication among devices. It operates both on instantly and continuously influenced sensors, supports concurrent events for context extraction, and establishes a secure shared group key among devices that sense the same events.

IoTCupid first obtains the sensor measurements corresponding to events sensed by each device. It implements a feature processing technique through a window-based derivation algorithm to support sensors that measure instant (e.g., sound) and continuous physical quantities (e.g., temperature and humidity). It then derives temporal sensor features from detected events, extends a fuzzy clustering algorithm to group concurrent and independent events into different types, and obtains inter-event timings (time interval between consecutive events) for each event type. Lastly, it uses the inter-event timings as evidence to authenticate the devices and establishes a shared group key among all devices that sense the same events. IoTCupid's group key establishment protocol enables dynamic group generation and is resilient to MitM, offline brute-force and denial of key exchange attacks. In contrast to previous approaches, IoTCupid provides a fast, practical, and secure group pairing scheme that automates pairing diverse device types with no active user involvement and a minimal computational and storage cost.

We present two studies evaluating IoTCupid's effectiveness. In a first study, we conducted experiments in a smart home with 4 sensors and 4 event sources while two residents were conducting their routine activities. We also deployed devices outside the home to evaluate IoTCupid's resilience against attacker devices that aim to pair with legitimate devices. To demonstrate IoTCupid's practicality in different environmental conditions, in a second study, we evaluated IoTCupid on a dataset [18], [19] collected in an office environment with multiple devices and event sources while four people were conducting their everyday work. IoTCupid correctly detected events with an average precision and recall rate of 95.8% and 83%, and successfully paired all devices with only four equivalent inter-event timings. We show that the attacker devices cannot pair with legitimate devices using IoTCupid. These studies demonstrate IoTCupid establishes group keys without a significant overhead. It takes 39 milliseconds on average to derive group keys among 20 devices with 4 event sources, and the overhead increases linearly with an increasing number of event sources and devices. In summary, we make the following contributions:

- We introduce IoTCupid, a secure and practical group pairing system for heterogeneous sensors. IoTCupid leverages inter-event timings of commonly sensed events to pair devices with diverse sensing modalities in a short duration with minimal computation and storage cost.

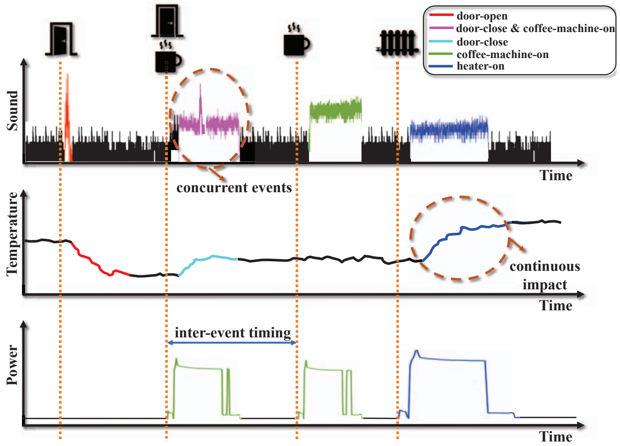


Figure 1: An illustration of how common events sensed by different sensors can be used for device pairing.

- We design a dynamic group key establishment protocol that extends a partitioned group password-authenticated key exchange scheme with provable security.
- We perform two studies in a smart home and smart office to show IoTCupid's effectiveness and performance in pairing multiple devices with diverse modalities.

2. Problem Statement

Trusted gateways may not always be available in IoT deployments. In such cases, secure communication channels are constructed through decentralized pairing systems that authenticate the devices and establish cryptographic keys for device-to-device communication. Most of these systems require human involvement (e.g., simultaneously shaking two devices) to pair the devices. However, such solutions have limited usability and are infeasible for IoT deployments with the increasing number of devices. Therefore, recent efforts have explored context-based pairing to provide secure communication channels [5], [15], [16], [17]. These approaches, however, can only pair devices equipped with homogeneous sensors that sense identical sensing modalities.

Since IoT devices are equipped with sensors with different capabilities and sensing modalities, our goal is to design a secure, and practical pairing system for these heterogeneous devices based on their shared context, without user involvement. To illustrate, we consider an IoT deployment with three devices. Each device is equipped with one of the following sensors: a microphone, a power meter, and a temperature sensor. In the morning, `user-A` opens the door (`door-open`) to go out. Meanwhile, `user-B` turns on the coffee machine (`coffee-machine-on`). While the coffee machine is on, `user-A` returns and closes the door (`door-close`). `User-A` prepares a cup of coffee for herself (`coffee-machine-on`) and turns on the heater (`heater-on`).

Figure 1 presents the physical influences of the described event sequence on the sensors, where all devices perceive

TABLE 1: Commonly occurring events in IoT environments and the sensors impacted by these events.

Event	Sensors Impacted
door-open/close	air pressure, humidity, illuminance, microphone, motion, temperature
coffee-machine-on/off	microphone, power
window-open/close	air pressure, humidity, illuminance, motion, temperature
oven-on/off	humidity, power, temperature
light-on/off	illuminance, power
AC-on/off	air pressure, humidity, microphone, power, temperature
heater-on/off	humidity, microphone, power, temperature
TV-on/off	illuminance, microphone, power
dryer-on/off	humidity, microphone, power, temperature

common events. For instance, the door-open/close events influence the microphone and temperature sensor while the microphone and power meter sense the coffee-machine-on event. Similarly, heater-on influences all three sensors.

In general, commonly occurring events in typical IoT environments are sensed by multiple devices equipped with heterogeneous sensors [6], [18], [20], [21], [22]. Table 1 summarizes commonly occurring events and the group of sensors impacted by these events in a typical smart home setting. Although the devices' measurements are not directly comparable due to different signal characteristics and their times may not be synchronized, these devices can leverage the inter-event timings to verify they observed the same event. Particularly, two or more devices can use the time interval between the subsequent occurrences of a commonly observed event type (e.g., coffee-machine-on events sensed by the microphone and power meter) as a proof of co-presence and use them as an evidence to establish a symmetric key.

Definitions. In this paper, we use the term *events* to refer to instances of changes in the device states (e.g., door-open, coffee-machine-on and heater-on). Events influence a set of physical channels that are measured by sensors. We use the term *signals* to refer to the processed sensor readings that represent the influence of an event separated from the background noise. Common devices in IoT environments are influenced by multiple *event types*. For instance, a device equipped by a temperature sensor may be influenced by events of types door-open, heater-on and AC-on.

2.1. Design Requirements and Challenges

Several schemes leveraged shared homogeneous context to securely pair IoT devices [5], [15], [16], [17], and only a single prior work [6] explored using inter-event timings for the secure pairing of heterogeneous sensing devices. However, these schemes suffer from four primary problems, which makes them impractical in many IoT environments. We detail them below and address each with IoTCupid.

Short Pairing Time. Existing works use cryptographic primitives that are vulnerable to offline brute-force attacks. In these attacks, the attacker enumerates all possible inter-event timings to derive the cryptographic keys. Thus, they require many inter-event timings to provide sufficient entropy for the shared keys to be cryptographically secure. The time needed to derive secure keys is further increased due to concurrent events, which cannot be used for pairing as they create inter-event timing mismatches (Detailed below).

Pairing Continuously Influenced Sensors. Prior pairing systems only consider sensors that are instantly influenced by an event. Yet, in real environments, many sensors measure continuous physical quantities (e.g., temperature and humidity). We refer to such sensors as *continuously influenced* sensors. For instance, in Figure 1, the heater-on event's sound instantly influences the microphone. However, the temperature continuously increases over time. Unless event detection considers gradual changes in the temperature, the heater-on event cannot be detected and used to pair the microphone and temperature sensor. Without considering continuously influenced sensors, the number and types of devices which can be paired are very limited.

Concurrent Events. Existing pairing schemes mainly evaluate scenarios where a device only senses a single event per time period. However, in practical scenarios, multiple events occur simultaneously and produce an overlapping influence on the sensors. In Figure 1, coffee-machine-on and door-close happen concurrently, and the microphone measures their aggregated influence. Prior works group concurrent events into a new separate event type. This leads to longer pairing times since the inter-event timings for a device sensing concurrent events will not match with another device sensing only one of those event types. To address this, a method to separate independent and concurrent events into groups is needed for scalable and practical pairing.

Deriving Group Keys. In centralized IoT deployments, a trusted hub monitors and controls the devices. However, in decentralized IoT protocols, devices need to broadcast their states to invoke their event-triggered automations. For instance, a smoke-detector broadcasts the smoke-detected event; upon receiving, the lights are turned on and the door is unlocked for user safety. Previous decentralized pairing schemes establish individual keys among pairs of devices. When a device needs to broadcast a message, it individually encrypts it with all shared keys and then sends the ciphertexts, causing linear computation, communication and energy consumption overhead. Since IoT devices typically have limited resources and energy constraints, this overhead negatively impacts their performance and battery.

2.2. Threat Model

The attacker, $\mathcal{A}$, aims to eavesdrop on the communication between IoT devices and learn private information about users. We assume that the devices are deployed within an indoor closed physical space (e.g., smart home, smart office, industrial control room, etc.), and controlled by a common trusted entity (e.g., smart home owner, office occupants, etc.). We assume that the attacker is not present within the physical boundary of the indoor IoT environment and cannot access, add devices or control the devices inside. We also assume that the attacker has complete knowledge of the pairing protocol and has access to the communication channels.

We consider $\mathcal{A}$ can launch the following attacks: (1) Eavesdropping attack, $\mathcal{A}$ places malicious devices $D_{\mathcal{A}}$

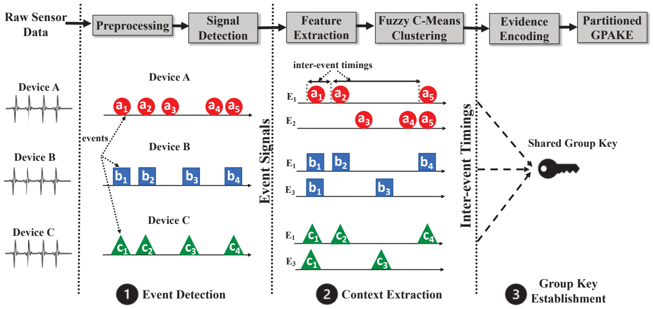


Figure 2: Overview of IoTCupid's architecture.

outside the IoT environment's physical boundary to pair them with legitimate devices. $D_{\mathcal{A}}$ may be $(\mathtt{a})$ embedded with off-the-shelf sensors with similar capabilities as the legitimate devices, or $(\mathtt{b})$ equipped with higher-end sensors that are more powerful and expensive compared to the legitimate devices. (2) Man-in-the-middle attack, $\mathcal{A}$ intercepts the messages between legitimate devices and attempts to establish keys with them. (3) Brute-force attacks, $\mathcal{A}$ tries every possible evidence to derive the cryptographic keys used by legitimate devices. $\mathcal{A}$ can conduct this attack in two ways. For online attacks, $\mathcal{A}$ joins the key establishment process and guesses the evidence. For offline attacks, $\mathcal{A}$ eavesdrops on the communication between the legitimate devices and attempts to crack the established key after pairing. (4) Denial of key exchange, $\mathcal{A}$ participates in group key establishment with random evidences to prevent legitimate devices from establishing a key.

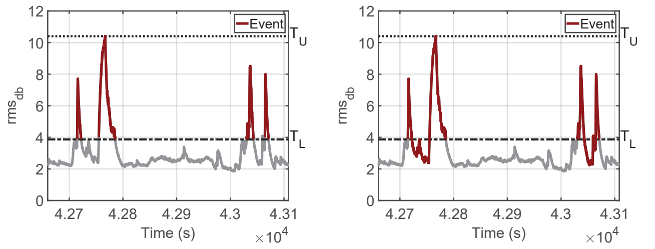
3. IoTCupid

3.1. System Overview

Figure 2 illustrates the three stages of IoTCupid. IoTCupid first processes the raw time-series data collected in real-time by both instant and continuously influenced sensors and performs a threshold-based signal detection to separate the sensor data corresponding to events (signals) from background noise (❶). For instance, it determines that Device A detects five signals ($\mathtt{a}_1$-$\mathtt{a}_5$), and Device B and Device C detect four signals ($\mathtt{b}_1$-$\mathtt{b}_4$, $\mathtt{c}_1$-$\mathtt{c}_4$).

Second, IoTCupid extracts distinctive time-series features from the signals each sensor has detected. It then extends a fuzzy clustering algorithm to group independent and concurrent signals into different events (❷). For example, Device A creates two event clusters, $\mathtt{E}_1$ and $\mathtt{E}_2$. It then groups each detected signal into one or both of these clusters ($\mathtt{a}_5$ is grouped into both clusters as $\mathtt{E}_1$ and $\mathtt{E}_2$ occur concurrently). The clustered events are used to obtain the sequence of time intervals between consecutive events of a given type, which serve as evidence of the devices' shared context.

Lastly, IoT devices use the inter-event timings to authenticate each other and establish a shared group key (❸). IoTCupid encodes the inter-event timings into passwords and extends a partitioned group password-based authenticated key exchange scheme for a group key establishment protocol.



(a)

(b)

Figure 3: Signal detection using lower ($\mathtt{T}_\mathtt{L}$) and upper ($\mathtt{T}_\mathtt{U}$) thresholds. (a) shows short discontinuities between detected signals aggregated by our approach in (b).

We consider the devices that sense the same event as a *group*. Through this, each subset of devices that have the same inter-event timings establishes a group key. For instance, Devices A, B, and C derive a shared group key since they extract matching inter-event timings through $\mathtt{E}_1$.

Deployment. IoTCupid presents a secure system for ad-hoc connectivity among heterogeneous IoT devices without a central gateway or IoT hub. It does not require specific user actions to initiate the protocol or trigger events in the IoT environment. It solely depends on the entropy extracted from events resulting from users' routine activities. In IoTCupid's key establishment protocol initiation, all devices broadcast their public keys encrypted with the extracted inter-event timings for establishing group keys. Given the ad-hoc nature of the network, any device broadcasting its encrypted public key can initiate and participate in the protocol. Thus, IoTCupid operates without knowing the number of devices present in the IoT deployment.

3.2. Event Detection

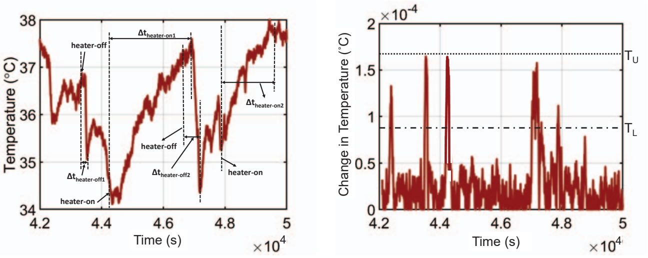
IoTCupid's event detection operates on each sensor's raw time-series data. We first pre-process the sensor data for signal smoothing and noise reduction. We then perform threshold-based signal detection to separate the events' impact on sensor data from noise. We adapt our approach for both instantly and continuously influenced sensors.

3.2.1. Sensor Data Extraction and Pre-processing.

To extract signals corresponding to events, we first segment the sensor data into multiple samples with window size, $\mathtt{w}_\mathtt{s}$. Many sensor values heavily fluctuate throughout the day (e.g., temperature sensor readings depend on ambient temperature) [6], [23]. To address this, we first normalize the sensor readings to eliminate these fluctuations' impact and capture the transient changes caused by events. We then apply a smoothing filter by computing sensor data's exponentially weighted moving average (EWMA) to reduce noise. We compute the sensor data's EWMA as $\mathtt{S}_\mathtt{w} = \alpha * \mathtt{Y}_\mathtt{w} + (1 - \alpha) * \mathtt{S}_{\mathtt{w}-1}$, where α is the weight, $\mathtt{Y}_\mathtt{w}$ is the sensor data in window $\mathtt{w}$, and $\mathtt{S}_{\mathtt{w}-1}$ is the EWMA of the preceding window. Appendix A shows an example of the sensor data before and after pre-processing.

3.2.2. Event Signal Detection.

We design a threshold-based approach to distinguish events' influence on sensor readings



(a) (b)

Figure 4: (a) Raw sensor data collected from a temperature sensor. (b) Absolute of the first derivative of the temperature data with upper and lower thresholds.

from background noise. We leverage a lower threshold, T_L, to identify peaks in the sensor readings that distinguish events' impact from background noise, and an upper threshold, T_U, to remove high amplitude noise signals. We consider the consecutive timestamps at which the sensor values exceed T_L but are below T_U as a signal representing a single event.

Figure 3 (highlighted region) shows how the thresholds determine the microphone signal corresponding to an event. We disregard short discontinuities between the signals (Figure 3a) to ensure small fluctuations do not segment a signal representing a single event into multiple signals. To detail, we aggregate consecutive events into a single signal if the duration between these signals is less than the aggregation threshold, t_A, (Figure 3b).

Detecting Continuous Physical Quantities. The event detection approach described above is suitable for sensors instantly impacted by an event since the raw sensor readings can be directly compared to the thresholds. Yet, many sensors measure continuous physical quantities, such as temperature, and humidity, that may not change instantly in response to an event. For instance, a `heater-on` event occurring at time t causes a gradual increase in temperature sensor values after a delay (Δt), which may vary depending on environmental factors such as the sensor's distance from the heater.

To illustrate, consider the raw sensor data from a temperature sensor shown in Figure 4a. The sensor values gradually increase or decrease in response to `heater-on/off` events. However, the maximum (or minimum) temperature values are sensed after a delay from the original event timestamp. The length of this delay is different even for two events of the same type. For example, for the two `heater-on` events, $\Delta t_{\mathtt{heater\text{-}on2}}$ is larger than $\Delta t_{\mathtt{heater\text{-}on1}}$. Additionally, the maximum (or minimum) values recorded in response to the two events also vary. Due to these variations, using thresholds determined from the raw sensor data results in inaccurate event detection and incorrect inter-event timings.

To account for the gradual changes and the varying delay in the impact on sensor readings, IoTCupid leverages the rate of change in the sensor readings to detect signals corresponding to events for continuously influenced sensors. IoTCupid first computes the derivative of the pre-processed sensor values in each window (w) as $S'_w = (S_{w_{w_s}} - S_{w_0})/w_s$, where w_s is the window size and w_0 and w_{w_s} are the first and last sensor values in the window. IoTCupid then applies the lower and upper thresholds (T_L and T_U) determined based on the average derivative values for each sensor. We extract the timestamps where the absolute value of the sensor readings' derivatives lie within the predetermined T_L and T_U for the sensor. Compared to the raw sensor data, the sensor data's derivative has clear peaks that align well with the events' timestamps, as shown in Figure 4b.

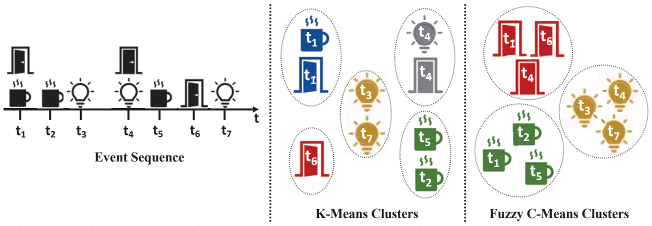


Figure 5: Comparison of event clustering through K-Means and Fuzzy C-Means algorithms.

3.3. Context Extraction

IoTCupid leverages inter-event timings as evidence of a shared context among devices. To compute inter-event timings, it first derives the temporal features of the detected signals, performs dimensionality reduction, and clusters them into events. It then computes the sequence of time intervals between the start times of events of a given type.

3.3.1. Event Clustering. We cluster the detected signals into different events to extract their inter-event timings. This is because each device can sense multiple event types, and the devices do not know the type of detected signals. We use temporal features extracted from the detected signals to cluster similar events via a fuzzy clustering algorithm, without any prior information about the event types.

Deriving Temporal Sensor Features. IoTCupid implements a feature extraction and selection algorithm to extract features representing different events. It extracts time-domain features (F) such as min, max, mean, and median from the signals corresponding to each event. We perform feature selection to identify the features that enable the correct characterization of different events. We normalize the selected features and perform dimensionality reduction through principal component analysis (PCA) [24] to select a subset of features, $F_{\mathtt{min}}$, which is used to differentiate event types.

Fuzzy C-Means Event Clustering. In real IoT deployments, multiple events may occur simultaneously and produce an overlapping signal on the devices. For instance, consider the event sequence shown in Figure 5, where `door-open` and `coffee-machine-on` events occur concurrently. The signal features corresponding to such simultaneous events may significantly differ from the same event's occurrence in isolation. Thus, traditional hard clustering methods such as K-Means may cluster concurrent events into a separate type instead of the existing types (as shown by the K-Means output in Figure 5). This approach results in a higher mismatch in the inter-event timings generated by two devices for a given event type, leading to longer pairing times.

To address this, we extend fuzzy C-Means clustering [25] to assign the detected signals into one or more appropriate event clusters based on the extracted features. Fuzzy C-Means partitions signal into $\mathtt{c}$ fuzzy event clusters, where $\mathtt{c}$ is an input. Each signal corresponding to an event is assigned a degree of membership to each of the $\mathtt{c}$ clusters. For a given signal ($\mathtt{e}$), the degree of its membership ($\mathtt{u_{ej}}$) in cluster $\mathtt{j}$ is determined by $\mathtt{u_{ej}} = 1/\sum_{\mathtt{k}=1}^{\mathtt{c}}(\mathtt{d_{ej}}/\mathtt{d_{ek}})^{2/\mathtt{m}-1}$, where $\mathtt{d_{ej}}$ represents the Euclidean distance between the cluster $\mathtt{j}$'s centroid and signal $\mathtt{e}$'s feature vector. Since a device does not know the possible event types that may occur, IoTCupid employs the elbow method to identify the optimal value of $\mathtt{c}$ [26]. The fuzziness index, $\mathtt{m}$, is the hyper-parameter controlling the tendency of an element to belong to multiple clusters. We detail how $\mathtt{c}$ and $\mathtt{m}$ are determined in Section 4. This process allows two events occurring simultaneously to belong to their appropriate clusters (Figure 5 (Right)).

3.3.2. Context Evidence Generation. IoTCupid generates inter-event timings, $\mathtt{I}$, for each device after clustering the detected signals into different event types. Each device $\mathtt{d}$ extracts and concatenates the time intervals between the start times of event occurrences $\langle \mathtt{e_1}, \ldots, \mathtt{e_n} \rangle$ in cluster $\mathtt{k}$ to generate inter-event timings, $\mathtt{i_k}$. IoTCupid uses the inter-event timings for all events sensed by a device ($\mathtt{I}$) as evidence in our group key establishment protocol (Section 3.4).

Using inter-event timings as evidence to bootstrap key establishment has significant advantages for pairing heterogeneous devices. First, two devices detecting the same event roughly record the same inter-event timings even if their raw signals have different characteristics (e.g., `door-close` events sensed by a microphone and gyroscope). Second, inter-event timings eliminate the need for a global clock or time synchronization. Even if an event's timestamps observed by two devices are not synchronized (e.g., `heater-on` is instantly sensed by a sound sensor but gradually sensed by a temperature sensor with a delay), the time intervals between consecutive events of the same type are still similar. Third, inter-event timings eliminate the impact of changes in the duration of detected events, e.g., a `coffee-machine-on` event's duration depends on the number of cups.

3.4. Establishing Group Keys from Evidences

After devices extract inter-event timings, they use the timings as evidence to authenticate each other and establish group keys. A group key is shared between the devices that sense the same event type, and all devices in the group can securely communicate using a single key. Group keys have unique advantages over deriving individual keys among each device pair. First, devices must store an individual key for each device they pair with. Second, when a device communicates with multiple devices (e.g., broadcasts a message), it must encrypt and authenticate the message with all individual keys. Due to the storage, computation, and energy constraints of IoT devices, linear storage, communication, and computation overhead from individual keys significantly deteriorates the devices' performance and battery.

3.4.1. Design Space Exploration. Deriving group keys using inter-event timings from multiple events in a dynamic IoT deployment introduces several challenges. First, the groups must be generated dynamically based on the devices that sense the same event. Second, the group key establishment protocol must support device addition and removal. When a device is added, it must pair with the existing devices for secure communication, and when a device is removed, its keys must be revoked since an adversary can capture it (e.g., through reselling or returning [4], [27]) and physically extract the keys. Unfortunately, prior works cannot be easily extended to address these challenges.

Group Diffie-Hellman. Group keys can be generated using the secure communication channels from the individual keys derived through a standard pairing protocol. This means the devices run an additional Group Diffie-Hellman (GDH) [28] protocol to derive group keys, increasing the pairing time. Here, the group key establishment protocol must be run over the secure communication channels to prevent MitM attacks. Since group key establishment protocols require multiple broadcasting rounds, using the secure channels between each pair of devices further increases the communication and computation overhead of the group key establishment. Additionally, when a device is added to the IoT environment, it must first individually pair with other devices to be authenticated and then participate in the group key establishment.

Fuzzy Commitment. Several approaches use fuzzy commitment schemes [29], [30] to generate individual keys. These schemes are built on error-correcting codes and enable verifying two evidences even when they have small differences (e.g., Hamming distance less than a threshold). These schemes can be extended to derive group keys where each device broadcasts its commitment, and the ones with similar evidences derive the same keys. Yet, approaches only built on fuzzy commitment are vulnerable to offline brute-force key guessing attacks [12]. In this attack, the adversary collects the network traffic and tries all evidences until they find the one that can decrypt the network traffic.

There are two approaches to protect against these attacks. First, a large number of evidences (i.e., inter-event timings) can be used to derive the keys. Yet, this approach sacrifices efficiency since it may take a long time to derive a large number of evidences. Second, Password-Authenticated Key Exchange (PAKE) schemes have been proposed to prevent offline brute-force attacks. However, extending PAKE into group settings is non-trivial, as discussed below.

Group PAKE (GPAKE). GPAKE enables multiple devices sharing the same evidence to derive group keys [31], [32]. However, the passwords of all devices that participate in the key agreement must be the same because these schemes abort without establishing a shared key even if a single password is different. The adversary can leverage this limitation by joining the key agreement protocol with arbitrary evidences to deny legitimate devices from deriving shared keys. We refer to this attack as *Denial of Key Exchange*.

TABLE 2: Our group key establishment protocol.

	Device 1 (d_1)	**Device 2** (d_2)	...	**Device N** (d_N)
	Step 1: Evidence Extraction			
❶	$\{i_{1,d_1} \dots i_{c_1,d_1}\} = \text{CONTEXT\_EXTRACTION}(d_1)$	$\{i_{1,d_2} \dots i_{c_2,d_2}\} = \text{CONTEXT\_EXTRACTION}(d_2)$		$\{i_{1,d_N} \dots i_{c_N,d_N}\} = \text{CONTEXT\_EXTRACTION}(d_N)$
	Step 2: Encoding			
❷	$\{pw_{1,d_1} \dots pw_{c_1,d_1}\} = \lfloor \{i_{1,d_1} \dots i_{c_1,d_1}\}/W \rfloor$	$\{pw_{1,d_2} \dots pw_{c_2,d_2}\} = \lfloor \{i_{1,d_2} \dots i_{c_2,d_2}\}/W \rfloor$		$\{pw_{1,d_N} \dots pw_{c_N,d_N}\} = \lfloor \{i_{1,d_N} \dots i_{c_N,d_N}\}/W \rfloor$
	Step 3: Partitioned GPAKE			
❸	Determine the public parameters, two primes p and q, a finite field $\mathbb{F}_q$ and a group $\mathbb{Z}_p$. $E(\mathbb{F}_q)$ is an elliptic curve, and $P \in E(\mathbb{F}_q)$ is its generator.			
❹	Choose random $\{x_{1,d_1} \dots x_{c_1,d_1}\} \overset{\$}{\leftarrow} \mathbb{Z}_p$	Choose random $\{x_{1,d_2} \dots x_{c_2,d_2}\} \overset{\$}{\leftarrow} \mathbb{Z}_p$		Choose random $\{x_{1,d_N} \dots x_{c_N,d_N}\} \overset{\$}{\leftarrow} \mathbb{Z}_p$
❺	$X_{i,d_1} \leftarrow x_{i,d_1} \cdot P \bmod q$, $Y_{i,d_1} \leftarrow Enc_{pw_{i,d_1}}(X_{i,d_1})$	$X_{i,d_2} \leftarrow x_{i,d_2} \cdot P \bmod q$, $Y_{i,d_2} \leftarrow Enc_{pw_{i,d_2}}(X_{i,d_2})$		$X_{i,d_N} \leftarrow x_{i,d_N} \cdot P \bmod q$, $Y_{i,d_N} \leftarrow Enc_{pw_{i,d_N}}(X_{i,d_N})$
❻	Broadcast (d_1, Y_{i,d_1}), where $i \in \{1, \dots, c_1\}$	Broadcast (d_2, Y_{i,d_2}), where $i \in \{1, \dots, c_2\}$		Broadcast (d_N, Y_{i,d_N}), where $i \in \{1, \dots, c_N\}$
❼	For every received message (d_j, Y_{j,d_j}), where $j \in \{1, \dots, N\}$:			
❽	$X_{j,d_j} \leftarrow Dec_{pw_{i,d_1}}(Y_{j,d_j})$, $if(X_{j,d_j} \in E(\mathbb{F}_q))$:	$X_{j,d_j} \leftarrow Dec_{pw_{i,d_1}}(Y_{j,d_j})$, $if(X_{j,d_j} \in E(\mathbb{F}_q))$:		$X_{j,d_j} \leftarrow Dec_{pw_{i,d_1}}(Y_{j,d_j})$, $if(X_{j,d_j} \in E(\mathbb{F}_q))$:
❾	$sid_{i,j,d_1} = \{d_1, Y_{i,d_1}, d_j, Y_{j,d_j}\}$	$sid_{i,j,d_2} = \{d_2, Y_{i,d_2}, d_j, Y_{j,d_j}\}$		$sid_{i,j,d_N} = \{d_N, Y_{i,d_N}, d_j, Y_{j,d_j}\}$
❿	$sk_{i,j,d_1} \leftarrow H(d_1, d_j, X_{i,d_1}, X_{j,d_j}, x_{i,d_1} \cdot X_{j,d_j} \bmod q)$	$sk_{i,j,d_2} \leftarrow H(d_2, d_j, X_{i,d_2}, X_{j,d_j}, x_{i,d_2} \cdot X_{j,d_j} \bmod q)$		$sk_{i,j,d_N} \leftarrow H(d_N, d_j, X_{i,d_N}, X_{j,d_j}, x_{i,d_N} \cdot X_{j,d_j} \bmod q)$
⓫	$r_{i,d_1} \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\alpha_{i,j,d_1} \leftarrow Enc_{sk_{i,j,d_1}}(r_{i,d_1})$	$r_{i,d_2} \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\alpha_{i,j,d_2} \leftarrow Enc_{sk_{i,j,d_2}}(r_{i,d_2})$		$r_{i,d_N} \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\alpha_{i,j,d_N} \leftarrow Enc_{sk_{i,j,d_N}}(r_{i,d_N})$
⓬	Broadcast $(d_1, sid_{i,j,d_1}, \alpha_{i,j,d_1})$	Broadcast $(d_2, sid_{i,j,d_2}, \alpha_{i,j,d_2})$		Broadcast $(d_N, sid_{i,j,d_N}, \alpha_{i,j,d_N})$
⓭	For every received message $(d_j, sid_{i,j,d_j}, \alpha_{i,j,d_j})$, where $i \in \{1, \dots, c\}$ and $j \in \{1, \dots, N\}$:			
⓮	$if(Y_{i,d_1} \in sid_{i,j,d_j}) : r_{i,j,d_j} \leftarrow Dec_{sk_{i,j,d_1}}(\alpha_{i,j,d_j})$	$if(Y_{i,d_2} \in sid_{i,j,d_j}) : r_{i,j,d_j} \leftarrow Dec_{sk_{i,j,d_2}}(\alpha_{i,j,d_j})$		$if(Y_{i,d_N} \in sid_{i,j,d_j}) : r_{i,j,d_j} \leftarrow Dec_{sk_{i,j,d_N}}(\alpha_{i,j,d_j})$
⓯	$key_i \leftarrow \sum_{j=1}^{t}(r_{i,j,d_j})$	$key_i \leftarrow \sum_{j=1}^{t}(r_{i,j,d_j})$		$key_i \leftarrow \sum_{j=1}^{t}(r_{i,j,d_j})$

TABLE 3: Comparison of group key exchange approaches.

Group Key Exchange Protocol	Dynamic Group Generation	Efficiency	Resilience to Offline Attacks	Resilience to Denial of Key Exchange
Group DH	✗	✗	✓	✓
Fuzzy Commitment-based	✓	✗	✗	✓
Group PAKE	✗	✓	✓	✗
Our Protocol	✓	✓	✓	✓

3.4.2. Our Group Key Establishment Protocol. Table 2 shows our group key establishment protocol, which offers dynamic group generation with computational efficiency, device addition/removal, and resilience to offline brute-force and denial of key exchange attacks. We extend a partitioned GPAKE scheme [33] to build our protocol. Particularly, we include evidence extraction and encoding steps to first derive inter-event timings and then encode them into passwords to address their deviations. The devices then use the passwords to run the partitioned GPAKE scheme such that each subset of devices sensing the same events derives a group key. Here, we implement the partitioned GPAKE scheme over an elliptic curve to offer compact key sizes and fast computations. Lastly, we introduce a re-initiation-based key management scheme to support device additions and removals. Table 3 shows our protocol's advantages over the alternatives.

Evidence Extraction. Each device first extracts inter-event timings as evidence of co-location (❶). We represent the evidences as $\{i_1, \dots, i_c\}$, where c is the number of event types the device senses. For each event type, the devices can concatenate multiple inter-event timings to increase the entropy of the evidence. Our protocol requires 32-bit entropy in its evidences, whereas the protocols based only on fuzzy commitment (e.g., Perceptio's key establishment [6]) need 128-bit entropy that requires a larger number of inter-event timings. This is because our protocol is resilient to offline brute-force attacks, where an adversary who guesses the password correctly after the group keys are established cannot extract the keys. We further elaborate on each inter-event timing's entropy and required number of timings in Section 5.

Encoding. Slight deviations in the inter-event timings may occur since devices may have different sampling rates for their measurements. For instance, a device with a 10 Hz rate collects a sensor measurement every 0.1 seconds, whereas a device with 1 Hz rate collects every second. This leads the first device to compute an inter-event timing of 24.2 whereas the second device computes it as 24. Yet, the passwords used in the partitioned GPAKE must be identical for the devices to pair. To address this, we use a quantization window (W) to round down the inter-event timings while deriving the passwords (❷). Here, the quantization window introduces a trade-off between efficiency and the password's entropy. With a larger W, more devices have matching passwords, and with a smaller W, the passwords have higher entropy.

Partitioned GPAKE. We extend a partitioned GPAKE scheme with provable key secrecy and password privacy [33]. In this, a probabilistic polynomial time adversary who does not know the passwords cannot extract keys or passwords by eavesdropping or MitM attacks (See Appendix B for proof).

We implement the partitioned GPAKE scheme on an elliptic curve (EC). The devices first determine the public EC parameters (❸). Each device then generates a unique private and public key pair for each event type it senses and encrypts the public keys with its passwords (❹-❺). The devices broadcast the encrypted public keys with their device identifiers (❻). Only the devices with the same password can decrypt the messages, preventing an adversary from conducting a MitM attack.

After a device receives the encrypted public keys (❼), it tries to decrypt them using all of its passwords. If one matches with the password the public key was encrypted with, the device derives a valid public key (❽). The device then generates session IDs using the received IDs and public keys (❾), and derives intermediate two-party elliptic curve Diffie-Hellman (ECDH) keys (❿). It generates random values for each event type, encrypts them using the intermediate keys, and broadcasts along with its ID and the session ID (⓫-⓬).

Upon receiving a message, the device checks whether its ID is in the session ID of the received message. If it is, the device decrypts the message with its intermediate key to derive the random value of the other device (⓭-⓮). After collecting all such random values, the device adds them to derive the group keys (⓯). Since random values are uniformly sampled from $\mathbb{Z}_\mathtt{p}$, the group keys are random and secure. Yet, if the devices do not have a reliable source of randomness, they can use a key derivation function instead of addition to generate group keys. At the end of the protocol, each device derives a separate shared group key with the other devices that can sense the same event type.

Key Management. In IoT environments, device additions and removals are common. Yet, an added device cannot securely communicate with other devices without establishing keys with them, and an adversary may leverage the removed devices to physically extract keys. Therefore, a key management scheme is required to support added and removed devices. Prior schemes for sensor networks, however, require a trusted entity to assign keys to devices [34], [35], [36], which is infeasible for IoT devices from different vendors.

To address this problem, we integrate a re-initiation-based key management strategy. First, the added devices re-initiate IoTCupid and extract inter-event timings to prove their legitimacy to the existing devices. Thus, when IoTCupid is re-initiated, the existing devices also start extracting inter-event timings to pair with the newly added devices. Yet, an adversary can abuse this and attempt a denial of service attack by initiating the protocol unnecessarily. Such attacks are easy to detect since the adversary cannot prove its legitimacy and pair with the existing devices. When an adversarial pairing attempt is detected, the devices notify the user and we provide a waiting period to re-initiate the protocol. This period offers a trade-off between time to re-initiate the protocol for benign devices and the security against denial of service attacks.

Second, the removed devices can no longer derive correct inter-event timings. We integrate a periodic liveness check to detect if a device is removed and initiate IoTCupid to derive new keys after a device fails the liveness check.

Group-to-Group Communication. Devices may need to communicate with other devices in the same environment that they do not share a common event type. In such cases, they can leverage intermediary devices from their groups for communication. For instance, we consider a case where devices $\mathtt{d_a}, \mathtt{d_b}, \mathtt{d_c}$ share a key, and devices $\mathtt{d_b}, \mathtt{d_c}, \mathtt{d_d}$ share a key. When $\mathtt{d_a}$ needs to communicate with $\mathtt{d_d}$, it can leverage $\mathtt{d_b}$ or $\mathtt{d_c}$ as an intermediary device for secure communication.

Attack Resiliency. Our protocol offers resiliency to MitM attacks, offline brute-force password enumeration, and denial of key exchange. First, the adversary can try to conduct a MitM attack by intercepting a legitimate device's messages and establishing keys with other devices. Yet, as legitimate devices encrypt their public keys with passwords (❺), the adversary cannot decrypt them to join the protocol. Thus, the passwords (i.e., inter-event timings) provide the authentication necessary to protect against MitM attacks.

Second, the adversary can enumerate all possible inter-event timings to find the password used. However, the adversary still cannot extract the group keys due to the online ECDH session in the partitioned GPAKE (❿). Particularly, since the probability that the adversary can guess the password is very low (e.g., $1/2^{32}$), the adversary's attack can be successful only after the key agreement is completed. If an adversary recovers the password after keys are established, the adversary can decrypt the devices' broadcasted public keys. Yet, the adversary cannot use them to derive the private keys or group keys due to the hardness of the computational elliptic curve Diffie-Hellmann problem [37].

Lastly, an adversary can enter the protocol with random passwords to disrupt pairing. Here, the legitimate devices cannot decrypt the adversary's public keys and thus would ignore the adversary's keys while deriving their group keys.

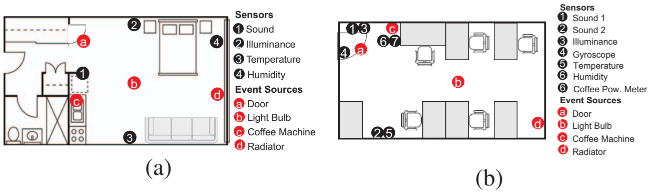
4. Implementation

Event Detection. We implement IoTCupid's event detection in Python 3.9.12. IoTCupid's event detection module uses lower and upper thresholds for sensors to detect events. Prior pairing schemes relying on event detection for pairing assume that these thresholds are built-in to the sensors by device manufacturers or manually configured [6]. However, in the case that such thresholds are not available (as for the devices in our experiments), we design an approach for threshold identification, which requires minimal sensor data without any information about the event types.

Our approach begins by separating the ambient noise from the event signals from the sensor data. We first extract the signal data in the interval $[\mathtt{t_s} - \Delta\mathtt{t_n}, \mathtt{t_e} + \Delta\mathtt{t_n}]$ for all events, where $\mathtt{t_s}$ and $\mathtt{t_e}$ are event start and end timestamps, and consider the rest of sensor data as noise. From the sensor signal values corresponding to an event, we consider the values between intervals $[\mathtt{t_s}, \mathtt{t_s} + \Delta\mathtt{t_v}]$ and $[\mathtt{t_e} - \Delta\mathtt{t_v}, \mathtt{t_e}]$ as event start and end signals. Here, we empirically determine $\Delta\mathtt{t_n}$ and $\Delta\mathtt{t_v}$ from a specific event type's average duration. We determine the frequency distribution of the signal values for noise and event samples separately by assigning them to equally sized bins. We then select the bin with the highest and lowest frequency for the event and noise samples, respectively. We set the selected bin's maximum and minimum values as the upper ($\mathtt{T_U}$) and lower ($\mathtt{T_L}$) thresholds. This approach allows IoTCupid to determine event detection thresholds for both instantly and continuously influenced sensors.

To determine the optimal value for the event detection window size ($\mathtt{w_s}$) and aggregation threshold ($\mathtt{t_a}$), we perform a grid search between 1 sec to 5 mins and choose the ones that give the highest event detection accuracy.

Context Extraction. We use the `MinimalFCParameters` class of `tsfresh` [38] package in Python to extract common time-domain features. We extend the Scikit-Fuzzy [39] package to implement the fuzzy C-Means clustering. To determine the optimal number of clusters ($\mathtt{c}$), we employ elbow method [26], average Silhouette coefficient [40], and gap statistic [41]. We observe the optimal number of clusters is similar using these methods and select the elbow method. Since IoTCupid does not assume any prior information about the type of events occurring in the environment, we need to identify the fuzziness index ($\mathtt{m}$) to accurately separate independent and concurrent events. For this, we perform a grid search to select the optimal value for the fuzziness index



(a)

(b)

Figure 6: IoT deployments in (a) a smart home and (b) office.

(m) based on the variance in the distances between cluster centroids and event feature vectors.

Group Key Establishment. We implement partitioned GPAKE on the FourQ elliptic curve [42], which offers 128-bit security with fast computations. We use blake2 [43] as the hash function and ChaCha-Poly [44] as the authenticated encryption due to their efficiency. We leverage the portable libraries of FourQ [45], blake2 [46] and ChaCha-Poly [47] to implement our protocol in C. We implement the communication rounds with the zeromq library [48].

5. Evaluation

We perform two studies to evaluate IoTCupid in two different IoT environments, a smart home and smart office.

In the first study, we conduct experiments with 4 sensors and 4 event sources to evaluate IoTCupid's effectiveness in pairing devices inside a home and its security against attacks launched from outside the environment. In the second study, we evaluate IoTCupid on a dataset [18], [19] collected in an office with multiple sensors and event sources to demonstrate IoTCupid's practicality in different environmental conditions. Our studies show IoTCupid effectively pairs all devices in the smart home and office via shared group keys using only four equivalent inter-event timings extracted from 13 or fewer events detected by each device. We present our IoTCupid analysis results by focusing on several research questions:

- **RQ1** What is the accuracy of IoTCupid in event detection?
- **RQ2** What is the impact of sensors' locations on IoTCupid's event detection accuracy?
- **RQ3** What is the time required to achieve sufficient entropy for our group key establishment protocol?
- **RQ4** How effective is IoTCupid in establishing group keys?
- **RQ5** How resilient is IoTCupid against adversarial sensors?
- **RQ6** What is IoTCupid's performance overhead?
- **RQ7** How does IoTCupid perform against other schemes?

Evaluation Setup. Figure 6a shows the placement of the 4 event sources and 4 sensors in the smart home. The deployed sensor types are similar to commonly found sensors in smart homes. The event sources include the smart home's interior door, a ceiling light, an electric drip coffee machine, and a portable heater. We collect sensor traces over three days during which events are sporadically triggered by the two occupants (authors) while conducting their daily activities (e.g., walking, cooking, cleaning, etc.). We contacted our university's IRB office and got advised that IRB approval is not required as we do not collect any sensitive information.

TABLE 4: Events detected by sensors in IoT environments.

Events†	Sound	Illum.	Gyroscope	Temp.	Humidity	Pow. meter
`Door-open/close`	✓	✓	✓	✓	✓	✗
`Coffee-machine-on`	✓	✗	✗	✗	✗	✓
`Light-on/off`	✗	✓	✗	✗	✗	✗
`Radiator-on/off`	✗	✗	✗	✓	✓	✗

† The smart home includes a BMP180 temperature sensor, DHT11 humidity sensor, photoresistive illuminance sensor and an iPhone XR microphone. The smart office includes two USB microphones, a TSL2560 illuminance sensor, an ST Micro LSM9DS1 gyroscope, a BMP280 temperature sensor and a BME680 humidity sensor. There are two microphones in the office and they detect the same events.

TABLE 5: Smart home event detection results.

Event Sources	Sensors	Precision	Recall
ⓐ `door-open/close`	❶ Sound	1.0	1.0
	❷ Illuminance	1.0	1.0
	❸ Temperature	1.0	0.97
	❹ Humidity	0.9	1.0
ⓑ `light-on/off`	❷ Illuminance	1.0	1.0
ⓒ `coffee-machine-on`	❶ Sound	1.0	1.0
ⓓ `radiator-on/off`	❸ Temperature	1.0	1.0
	❹ Humidity	1.0	1.0

For the second environment, we use 4 event sources and 7 sensors, as shown in Figure 6b. The sensor data is measured over three days while four people are using the office for their everyday work. The door, light, and coffee events are triggered during the office occupants' routine activities, while the radiator events are triggered automatically.

We run IoTCupid on a Raspberry Pi 4 with ARM Cortex-A72 processor and 2 GB RAM.

5.1. Event Detection Performance

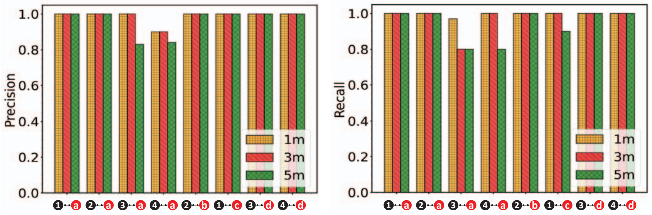
We evaluate each sensor's performance in distinguishing the events' physical influences from background noise. With a higher event detection accuracy, more sensors can derive similar inter-event timings in a shorter duration and establish group keys. Table 4 presents the events detected in the two environments. For instance, `door-open/close` events are detected by all devices except the power meter, and `light-on/off` events are detected by the illuminance sensor.

We present the event detection accuracy for each sensor type in terms of precision and recall. The precision represents the ratio of the number of detected events whose start times accurately match the ground truth event start times. The recall is the ratio of the number of correctly detected events to the number of events occurred. We only use the events' ground truth timestamps in the first hour of data collection from the smart office for identifying event thresholds and exclude these events from our evaluation. We use the same thresholds for event detection in both smart home and office. IoTCupid correctly detects events with an average precision and recall of 95.8% and 83%, and uses these detected events to extract matching inter-event timings for group key establishment. We note that IoTCupid's event detection may not capture some events with very small influence on the sensor data, resulting in a lower recall rate. Yet, the high precision detection of high impact events ensures IoTCupid extracts a sufficient number of inter-event timings for group key establishment.

Smart Home Deployment Results. Table 5 presents the event detection results of the 4 sensors in the smart home. All sensors detect events with a high precision and recall.

TABLE 6: Smart office event detection results.

Event Sources	Sensors	Precision	Recall
ⓐ `door-open/close`	❶ Sound 1	1.0	0.97
	❷ Sound 2	0.91	0.86
	❸ Illuminance	0.98	0.64
	❹ Gyroscope	1.0	1.0
	❺ Temperature	0.84	0.42
	❻ Humidity	0.98	0.44
ⓑ `light-on/off`	❸ Illuminance	1.0	1.0
ⓒ `coffee-machine-on`	❶ Sound 1	1.0	0.71
	❷ Sound 2	1.0	0.71
	❼ Power Meter	1.0	1.0
ⓓ `radiator-on/off`	❺ Temperature	0.8	0.55
	❻ Humidity	0.75	0.33



(a)

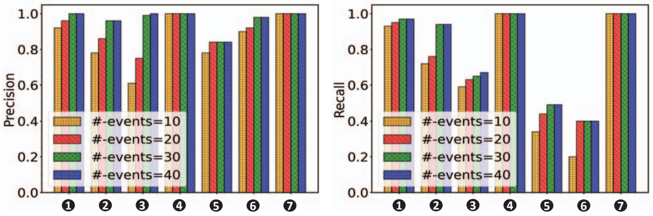
(b)

Figure 7: (a) Precision and (b) recall with varying distance between sensors and the event sources.

For instance, all sensors detect the `door-open/close` events with a precision and recall greater than 0.9. Thus, these sensors can use the door events to successfully establish a group key. Additionally, we found that the events' influence on the sensor data is relatively large compared to the ambient noise such as disturbances from the home occupant's activities, resulting in a high event detection accuracy.

Smart Office Deployment Results. Table 6 shows the precision and recall rate for the 7 sensors in the smart office. All sensors detect events with high precision. The recall rate is also high for gyroscope, coffee power meter, and sound sensors for all events. Yet, the temperature and humidity sensors yield a lower recall rate for the `radiator-on/off` and `door-open/close` events. This low recall is attributed to the relatively small influence of these events on sensor measurements, compared to the ambient changes in the temperature and humidity levels, particularly due to the uncontrolled disturbances from four office occupants' activities. Similarly, although the illuminance sensor accurately detects `light-on/off`, its recall is lower for the `door-open/close` events. This is because the change in illuminance caused by some events occurring during the day is insignificant compared to the ambient lighting in the office. Despite the low recall rate, these sensors' high precision event detection ensures that they can still correctly extract sufficient inter-event timings for participating in group pairing.

Impact of Distance on Event Detection. We conduct additional experiments in the smart home to demonstrate varying sensors' locations impact on event detection. We vary the devices' distance from the event sources between 1 to 5 m. Figure 7 shows the precision and recall for the detected events at different distances. IoTCupid's threshold-based signal detection lets most sensors detect all events correctly even when their distance from the event source increases. We find that the influence of door events on temperature and humidity sensors slightly deteriorates with an increased distance. This is because the short duration of the door events results in a small change in temperature and humidity, making it difficult to detect the events at a longer distance.



(a)

(b)

Figure 8: (a) Precision and (b) recall with varying number of events used for sensor calibration.

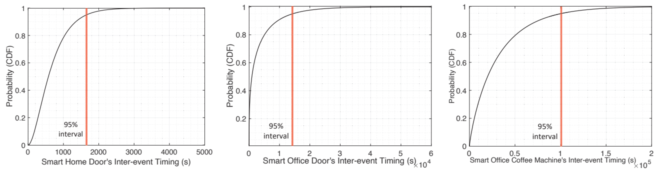
Impact of Number of Events on Sensor Calibration. We analyze how the number of events used for sensor calibration (i.e., determining sensor thresholds and parameters) impacts IoTCupid's event detection accuracy. We perform this analysis on the smart office dataset since it is collected in a noisy environment with a higher variance in the events' influence on the sensor data. Figure 8 shows the precision and recall rates for the detected events with an increasing number of events used for calibration. Both the precision and recall increase with more event calibration data as it improves the generalization of determined signal thresholds in detecting different event types. All sensors detect events with a precision higher than 0.8. Sensors that only detect a single event type require fewer events to achieve accurate detection. For instance, the coffee power meter and gyroscope need less than 10 events for high precision and recall.

5.2. Context Extraction and Key Agreement

We evaluate IoTCupid's efficacy in context extraction and key establishment by analyzing events' entropy and then present the groups of securely paired sensors. IoTCupid's fuzzy clustering enables deriving four matching inter-event timings among six sensors when only 13 or fewer events are detected by each device.

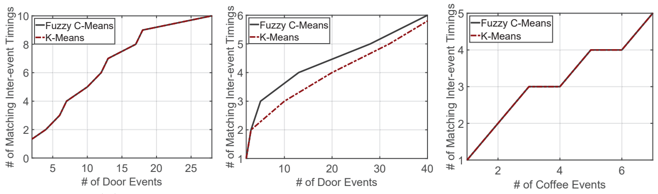
Entropy Analysis. We analyze the events' entropy to determine how many bits of security each inter-event timing provides with different quantization windows (`W`). This window is used in IoTCupid's encoding to address slight deviations in inter-event timings extracted by different devices. This analysis is required to determine the number of inter-event timings sufficient to securely pair the devices. We found inter-event timings provide enough entropy, even with larger windows, that can be used in IoTCupid's key establishment.

Previous works model event arrivals as a Poisson process, and thus, inter-event timings' probability density function follows a gamma distribution [6], [49], [50]. Hence, we fit a gamma distribution on the inter-event timings to compute their entropies. Figure 9 shows the cumulative distribution function (CDF) of the inter-event timings of the events sensed by multiple devices (door events in the smart home and door and coffee events in the smart office). From this, an adversary



(a) (b) (c)

Figure 9: CDF of inter-event timings for (a) smart home's door events, (b) smart office's door, and (c) coffee events.



(a) (b) (c)

Figure 10: # of matching inter-event timings for (a) smart home's door events, (b) smart office's door, and (c) coffee machine events with K-Means and fuzzy C-Means clustering.

TABLE 7: Event detection results for malicious devices.

Event Sources	Sensors	Precision	Recall
ⓐ door-open/close	Sound Illuminance Temperature Humidity	0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0
ⓑ light-on/off	Illuminance	1.0	0.2
ⓒ coffee-machine-on	Sound	0.0	0.0
ⓓ radiator-on/off	Temperature Humidity	0.0 0.0	0.0 0.0

Advanced attacker devices include a Snowball Black Ice USB microphone, a TSL2560 illuminance sensor and a BME 680 temperature and humidity sensor.

can uniformly sample inter-event timings from the intervals ($[0, T]$) that contain 95% of the timings to establish keys with legitimate devices. In this case, the probability that the adversary successfully guesses the inter-event timing is W/T. Thus, an inter-event timing's bit security is $\log_2(T/W)$. For instance, the coffee machine events in the smart office provide $\log_2(100500/30) = 11.71$ bits of security when the quantization window is $W = 30$ secs, and 95% of the inter-event timings fall into an interval of $[0, T = 100500]$.

Inter-event Timings Analysis. We present the similarity of inter-event timings computed by IoTCupid from the detected events after event clustering. Our analysis shows that sensors influenced by the same event types extract a sufficient number of matching inter-event timings. We determine the number of equivalent inter-event timings from the events sensed by multiple devices (door events in the smart home and door and coffee machine events for the smart office) as they can derive group keys with these timings.

Figure 10 shows the number of matching inter-event timings extracted by each sensor with an increasing number of events. We compute the inter-event timings for smart home door events with a quantization window $W = 8$ seconds to account for the differences in sensor sampling rates. Based on our entropy analysis, for the smart home sensors, only four matching inter-event timings are sufficient to achieve 32-bit password security. This enables four sensors influenced by door events to derive a secure group key with matching inter-event timings from less than 10 detected events.

For the smart office events, we use a larger quantization window $W = 60$ seconds due to the larger number of devices and higher ambient noise. Since the door and coffee events in the smart office offer a higher entropy (See Figure 9), four similar inter-event timings are sufficient to securely pair the devices despite the larger quantization window. The six smart office sensors sensing the door events establish a group key with only 13 door events. Even though the recall rate of temperature and humidity sensors for door events is relatively low, the correctly detected events are sufficient for these devices to derive the group key. The number of coffee events required to pair the sound sensors and the coffee power meter is even smaller, as shown in Figure 10c.

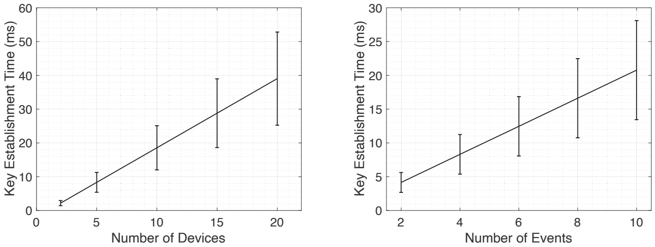
Comparison of K-means with Fuzzy C-Means. Figure 10 shows the effect of using K-Means instead of fuzzy clustering in distinguishing event types. K-Means takes longer to extract the same number of matching inter-event timings for door events in the smart office as it clusters concurrent events into separate event types. For instance, generating four matching inter-event timings among the six sensors needs 13 detected door events with fuzzy C-Means clustering while it takes 20 events with K-Means. The inter-event timings generated for the door events in the home and coffee events in the office are similar for the two methods since their timings do not overlap with the other events in the datasets.

Group-to-group Communication. IoTCupid supports group-to-group communication where devices that do not share a key can securely communicate over a common device. In the smart office, sensors are paired into two groups, i.e., the six sensors influenced by the door events and the sound sensors and power meter influenced by the coffee events. Although the power meter is not directly paired with the illuminance, temperature, humidity, and gyroscope sensors, it can still securely communicate with these devices via the sound sensors. With group-to-group communication, multiple devices that belong to various groups can broker communication between devices in different groups. Thus, unlike centralized IoT systems, IoTCupid does not rely on a single device for secure communication.

5.3. Security Analysis

We evaluate IoTCupid against eavesdropping attacks where an attacker tries to sense the events from outside. We conduct experiments in the smart home by placing sound, illuminance, temperature and humidity sensors outside the front door to simulate attacker devices ($\mathcal{D}_{\mathcal{A}}$). We deploy two types of $\mathcal{D}_{\mathcal{A}}$ equipped with: (a) off-the-shelf sensors with the same capabilities as the smart home devices, and (b) advanced sensors that are more powerful and expensive compared to the inside sensors. Our experiments show that $\mathcal{D}_{\mathcal{A}}$ can only detect events with a precision and recall rate of 0.125 and 0.025 on average. Thus, an attacker is unable to extract sufficient inter-event timings for pairing.

Table 7 demonstrates the average event detection results for the two types of $\mathcal{D}_{\mathcal{A}}$. We found that both the normal and advanced $\mathcal{D}_{\mathcal{A}}$'s sound, temperature and humidity sensors



(a) (b)

Figure 11: Key establishment time overhead with (a) varying number of devices when number of event types is 4, (b) varying number of event types when number of devices is 5.

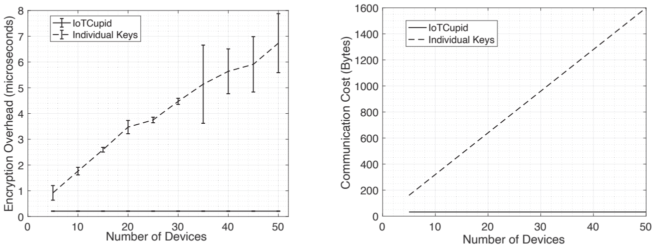
cannot correctly detect any of the events that occur inside the home; hence they cannot participate in the pairing protocol. This is because the home walls induce a significant distortion and attenuation in the event signals. We observe that the illuminance sensor senses a limited `light-on/off` events with a precision of 1.0. The light penetrates through the window when the light inside the home is turned on, especially at night when the outside ambient light is too low. Yet, the recall rate of the illuminance sensor is 0.2, lower than the sensors inside the home. This prevents it from extracting sufficient inter-event timings required for pairing with inside sensors.

We note that IoTCupid, by design, offers resiliency to MitM, offline brute-force, denial of key exchange attacks, and key extraction from removed devices. We show in Section 3.4.2 that IoTCupid's partitioned GPAKE scheme offers provable password and group key security against a probabilistic polynomial-time adversary conducting a MitM or brute-force attacks [33]. To launch a denial of key exchange attack, the adversary uses a malicious device to enter the key establishment with random passwords. Yet, legitimate devices still derive their group keys despite this attack because they discard the public keys of malicious devices as they do not share a password. Lastly, a removed device's keys cannot be used to communicate with legitimate devices since the legitimate devices derive new group keys with new inter-event timings in each key update.

5.4. Performance Evaluation

Event Detection and Context Extraction Overhead. We evaluate IoTCupid's event detection overhead by measuring the average time for processing and detecting events in each sensor data window. IoTCupid takes, on average, 20.08 secs to pre-process and extract event signals when the window size ($\mathtt{w_s}$) is 2 mins. From the extracted event signals, IoTCupid performs feature extraction in 5.66 ms and inter-event timing extraction in 73.9 ms on average. Since the computation overhead for signal detection and feature extraction is negligible compared to the sensor data window size, it does not impact the overall performance of IoTCupid.

Group Key Establishment Overhead. We evaluate IoTCupid's key establishment overhead, the time between context extraction to deriving group keys, with an increasing number of devices and event types. Figure 11 shows the computation time where we run our protocol 100 times and compute the average and standard deviation of the timings. IoTCupid can efficiently pair a large number of devices with inter-event timings extracted from multiple event types. For instance, with 4 event types and 20 devices, the computation overhead is 39.04 ± 13.78 ms ($\approx$ 39M CPU cycles), and with 10 event types and 5 devices, the overhead is 20.77 ± 7.32 ms ($\approx$ 20.8M CPU cycles). The overhead increases linearly with the number of devices (See Figure 11a) and event types (See Figure 11b). This is because the communication rounds and elliptic curve scalar multiplications dominate the time overhead, and their number increases linearly.



(a) (b)

Figure 12: (a) Encryption and (b) communication cost with IoTCupid's group keys and Perceptio's individual keys.

Memory Usage. IoTCupid requires each device to temporarily store the data received from other devices during the group key establishment protocol's communication rounds. This corresponds to $96 * X * Y$ Bytes, where X is the number of event types and Y is the number of devices. For instance, when there are 100 devices that sense 10 event types, the memory usage of each device is 93.75 KB, which is acceptable even for low-end IoT devices.

Encryption and Communication Cost. We evaluate IoTCupid's encryption and communication cost with a varying number of devices (See Section 5.5 for comparison with individual keys). We run the encryption scheme for a 32-byte message 100 times. We observe IoTCupid's secure channels incur 0.21 ± 0.005 ms ($\approx$ 0.2M CPU cycles). Figure 12 shows that IoTCupid's computation overhead and communication cost are constant with an increasing number of devices as the devices communicate with a group key.

We also evaluate IoTCupid's group-to-group communication cost when devices that do not share a key securely communicate over a common device that they share a key with. This requires one additional decryption and encryption operation, which takes 0.42 ± 0.007 ms.

5.5. Comparison with Prior Work

Among existing pairing approaches, only two works, T2Pair [12] and Perceptio [6], can pair heterogeneous IoT devices. T2Pair requires users to swipe screens or press buttons on devices and uses the timings of these actions as evidence for pairing. Given the need for active user involvement, quantitatively comparing T2Pair with IoTCupid is infeasible. Therefore, we discuss its strengths and weaknesses in comparison to IoTCupid in Section 7.

Perceptio [6], similar to IoTCupid, leverages inter-event timings from various sensing modalities as evidence of co-presence to pair devices. It detects events for only instantly influenced sensors and leverages a fuzzy commitment scheme to establish individual keys among devices with similar inter-event timings. Below, we quantitatively compare IoTCupid with Perceptio.

Paired Devices. We compare the number of devices paired using IoTCupid and Perceptio in our evaluation setup. Perceptio can only pair 7 out of the 11 devices (63%) in the two IoT environments while IoTCupid pairs all of them. Perceptio can only pair instant sensors but cannot detect events for continuously influenced sensors since it cannot capture the gradual changes in sensor values (as described in Section 3.2). For instance, Perceptio cannot pair the temperature and humidity sensors with the other devices even though they commonly sense the door events. In contrast, IoTCupid detects the events for both instantaneously and continuously influenced sensors and pairs all six devices in the smart office influenced by the door events.

Pairing Time. To compare IoTCupid and Perceptio's pairing time, we assume the time required to extract inter-event timings is same for both systems and compare their entropy bits required to defend against offline attacks. Since Perceptio relies on fuzzy commitment, it requires 128-bit entropy whereas IoTCupid requires 32-bit entropy due to its resilience to offline brute-force attacks. Thus, IoTCupid requires 4× less matching inter-event timings, resulting in 4× faster pairing compared to fuzzy commitment-based pairing protocols. Moreover, IoTCupid extracts correct inter-event timings using a fewer number of events. Particularly, for smart office's door events, IoTCupid extracts four matching inter-event timings from 13 occurred events whereas Perceptio requires 20. This translates into 54% faster pairing with IoTCupid.

Secure Communication Cost. We compare IoTCupid's computation and communication cost with Perceptio's when a device aims to broadcast a single 32-Byte message. IoTCupid incurs a constant overhead, whereas Perceptio's overhead linearly increases due to its pairwise individual keys [6] (Figure 12). Particularly, a device needs to encrypt the message one by one with all the individual keys and then send it to the other devices individually. For instance, broadcasting a message to 50 devices takes on average 0.21 ms and requires transmitting 32-Byte with IoTCupid's group key, while it takes on average 6.73 ms and requires transmitting 1600-Byte with Perceptio's individual keys. We note that linear overhead is inherent in any pairing protocol that establishes pairwise individual keys. On the contrary, with IoTCupid's group keys, the device encrypts the message only once with the group key and broadcasts it.

6. Limitations and Discussion

Handling Mismatches in Inter-event Timings. IoTCupid requires concatenating four inter-event timings in a password to provide enough entropy for group key establishment. However, inter-event timings of sensors that sense the same event may not always match (e.g., due to a sensor missing an event). This would create discrepancies in the passwords and prevent them from establishing a key.

To address such mismatches, we initially considered integrating a private set intersection (PSI) protocol [51] into IoTCupid for devices to determine which inter-event timings they should use for their passwords. However, since the universal set of possible inter-event timings is small (e.g., 2^8-2^{12}), an adversary can enter the PSI protocol with many inter-event timings to learn the benign devices' timings. Thus, we let devices enter our group key establishment protocol with all combinations of their inter-event timings, allowing them to use the matching ones to derive keys. We set an upper limit ($\mathtt{n_p}$) on the number of inter-event timings the device should extract before entering key establishment to ensure the number of combinations does not hurt the protocol's scalability. For instance, when $\mathtt{n_p} = \mathtt{10}$, the number of devices is 20, and the number of event types is 4, it takes, on average, ≈ 8 secs to run our key establishment protocol.

Deployment Considerations. IoTCupid uses window-based pre-processing and sensor thresholds for event detection and identifies the optimal parameters for clustering via statistical methods [26]. In practice, IoTCupid's calibration for determining these parameters can be performed in two ways: (a) offline by device manufacturers or (b) online by IoT service provider at the time of device installation. For the offline calibration, device manufacturers may calibrate sensors by (1) generating commonly occurring events in IoT deployments or (2) using publicly available smart home datasets [18], [19], [52], [53], [54] that include different sensors' measurements corresponding to common events.

We show in Section 5.1 that parameters extracted from a publicly available dataset [18] are transferable across IoT deployments. Yet, some IoT environments may include unique event types or may be exposed to environmental disturbances, distinct from typical IoT environments. In such cases, the manufacturer determined parameters may require fine-tuning for the specific deployment. For this, calibration can be initiated by IoT service providers by recording timestamps of various events occurring in the IoT deployment for a given amount of time (a few hours is sufficient, detailed in Section 5.1). Both of these calibration methods do not require any information about the event types. Each device only needs the sensor data and the timestamps at which events occurred to determine its parameters.

Resourceful Attackers. IoTCupid is resilient against normal and advanced eavesdropping attackers (Section 5.3). Yet, an attacker may have access to devices with asymmetric capabilities (e.g., x-ray vision). We do not consider such attackers as they could already visualize and reveal users' private activities, independent of our pairing protocol. Moreover, an outside attacker may attempt to inject signals to the inside sensors to pair with them or disrupt the pairing process. For this, the attacker may use electromagnetic interference (EMI), acoustic injection, and inaudible voice attacks [55], [56], [57]. Yet, such attacks require a high amplitude signal, which is difficult to achieve since outside signals experience a high attenuation from the walls (as shown in Section 5.3). Besides, these attacks can be identified by anomaly detection and prevented by shielding techniques [58], [59], [60], [61].

Pairing in Large Spaces. We demonstrate in Section 5.1 that IoTCupid can successfully pair devices at a distance of up to 5m. In large indoor spaces, some devices may be located far away from event sources and may not be able to sense the same events as other devices and establish the same group keys. However, such devices may share group keys with common devices (e.g., a nearby device paired with far away devices) or there may exist transient devices that have a view of different areas of the room (e.g., an illuminance sensor in a dining area may view both the kitchen and living room) and can sense the events occurring in each. These devices can then use the inter-event timings of the commonly observed events as evidence and act as a bridge between the groups in two distant areas for secure communication.

Rarely/Regularly Occurring Events. Although a few frequent events (e.g., `door-open/close`) commonly sensed by sensors are enough to establish group keys, some sensors may only detect rarely occurring events (e.g., a laundry washer's power meter). Such rarely occurring events would cause longer pairing times. Devices equipped with such sensors would need additional sensors (e.g., a microphone) that measure diverse events to timely pair with other devices.

Contrarily, some events may regularly occur in an IoT environment (e.g., smart home door opening at 9 am every day) and may be predictable by attackers. However, it is extremely difficult to predict successive timestamps of such events at a fine granularity. Thus, given that IoTCupid's group key establishment concatenates multiple inter-event timings of a given event type as evidence of co-presence and event detection accuracy is very low for attacker devices (Section 5.3), an attacker cannot extract evidences matching with the legitimate devices.

Impact of Environmental Noise. We develop a signal threshold-based event detection approach where we subtract the sensor readings' mean value in the preceding window for each window and compute the absolute difference before applying a smoothing filter for noise removal (Detailed in Section 3.2). This allows us to accurately detect events even with varying environmental noise. In rare cases, environmental noise's impact might be significantly higher than an event's physical influence, eliminating the event's impact on the sensor readings. Yet, this limitation is present in all existing systems that rely on sensing physical processes.

7. Related Work

Human-in-the-loop-based Pairing. Initial methods leverage mobile phone cameras and 2D barcodes to establish keys [62]. Mayrhofer et al. propose pairing the devices with a user simultaneously shaking them [13]. Move2Auth requires users to perform hand gestures by holding their smartphones in front of the devices and uses the variations in received signal strength for pairing [10]. Tap2Pair pairs devices through a user synchronously tapping on a device following the patterns displayed on the other device [11]. T2Pair needs users to apply operations such as pressing a button and swiping a touchscreen, and uses timestamps as a source of entropy [12]. SenCS pairs devices with mobile phones carried by users using the entropy from their actions (e.g., walking) [63]. These schemes need human involvement which only allows pairwise device pairing, incurring a huge manual effort from users with an increasing number of devices. Automating these user actions would require specialized equipment (e.g., robotic arms); thus, impractical for typical IoT environments.

TABLE 8: Comparison of IoTCupid with context-aware pairing schemes for IoT devices.

Pairing Scheme	Sensing Modality	Concurrent Events	Group Pairing	Continuous Sensors
Schurmann et al. [17]	Ambient sound	✗	✗	✗
Mathur et al. [64]	Wireless signal	✗	✗	✗
Miettinen et al. [15]	Ambient sound or light	✗	✗	✗
Rostami et al. [16]	Heart beat	✗	✗	✗
FastZIP [5]	Accelerometer, Gyroscope, Barometer	✗	✗	✗
Perceptio [6]	Heterogeneous sensors	✗	✗	✗
IoTCupid	Heterogeneous sensors	✓	✓	✓

Context-aware Pairing. To address the limitations of above approaches, context-aware pairing schemes have been proposed. Table 8 compares IoTCupid with several of them.

Schurrman et al. leverage audio context [17] and Miettinen et al. [15] use fingerprints from sound and luminosity to pair sensors. Mathur et al. [64] use similarities in the temporal variations in wireless channels between two nearby wireless devices as evidence. Rostami et al. [16] use the entropy extracted from heart beat signals of patients to pair implantable medical devices with their controllers. FastZIP [5] uses sensor fusion to construct fingerprints of shared context for intra-car device pairing with a Fuzzy PAKE scheme. Yet, these schemes do not support heterogeneous sensor modalities and therefore, their use cases are limited.

Similar to IoTCupid, Perceptio [6] uses inter-event timings from heterogeneous sensing modalities as evidence for co-presence to pair devices. Unfortunately, it does not support pairing continuously influenced sensors (e.g., temperature and humidity), does not support concurrent events, and only establishes individual keys. This results in longer pairing times to establish the keys, and linear storage, computation, and communication overhead after the keys are established. IoTCupid addresses these limitations, providing a secure and practical group pairing solution.

8. Conclusions

We introduce IoTCupid, a secure group pairing system for heterogeneous devices, without requiring active user involvement. IoTCupid exploits the fact that multiple co-located sensors sense the same events, and the time between subsequent event occurrences sensed by different sensors is similar. IoTCupid pairs both instantly and continuously influenced sensors, supports distinguishing concurrent events for context extraction, and establishes group keys from inter-event timings with partitioned GPAKE. We evaluated IoTCupid on two IoT environments, a smart home and smart office, and showed that it can pair devices with diverse sensing modalities with minimal overhead.

Acknowledgment

We thank our shepherd and the anonymous reviewers for their comments and suggestions. This work has been partially supported by the National Science Foundation (NSF) under grant CNS-2144645 and Office of Naval Research (ONR) under grant N00014-20-1-2128. The views expressed are those of the authors only.

References

[1] M. Fomichev, F. Álvarez, D. Steinmetzer, P. Gardner-Stephen, and M. Hollick, "Survey and systematization of secure device pairing," *IEEE Communications Surveys & Tutorials*, 2017.

[2] A. Kumar, N. Saxena, G. Tsudik, and E. Uzun, "A comparative study of secure device pairing methods," *Pervasive and Mobile Computing*, 2009.

[3] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, "A survey of IoT-enabled cyberattacks: Assessing attack paths to critical infrastructures and services," *IEEE Communications Surveys & Tutorials*, 2018.

[4] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in *USENIX Security*, 2019.

[5] M. Fomichev, J. Hesse, L. Almon, T. Lippert, J. Han, and M. Hollick, "Fastzip: Faster and more secure zero-interaction pairing," in *International Conference on Mobile Systems, Applications, and Services*, 2021.

[6] J. Han, A. J. Chung, M. K. Sinha, M. Harishankar, S. Pan, H. Y. Noh, P. Zhang, and P. Tague, "Do you feel what i hear? enabling autonomous IoT device pairing using different sensor types," in *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[7] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, 2013.

[8] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[9] "Openthread," 'https://openthread.io/', 2022, [Online; accessed 30-Jul-2022].

[10] J. Zhang, Z. Wang, Z. Yang, and Q. Zhang, "Proximity based IoT device authentication," in *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[11] T. Zhang, X. Yi, R. Wang, Y. Wang, C. Yu, Y. Lu, and Y. Shi, "Tap-to-pair: Associating wireless devices with synchronous tapping," *ACM Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2018.

[12] X. Li, Q. Zeng, L. Luo, and T. Luo, "T2pair: Secure and usable pairing for heterogeneous IoT devices," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[13] R. Mayrhofer and H. Gellersen, "Shake well before use: Intuitive and secure pairing of mobile devices," *IEEE Transactions on Mobile Computing*, 2009.

[14] M. K. Chong, R. Mayrhofer, and H. Gellersen, "A survey of user interaction for spontaneous device association," *ACM Computing Surveys (CSUR)*, 2014.

[15] M. Miettinen, N. Asokan, T. D. Nguyen, A.-R. Sadeghi, and M. Sobhani, "Context-based zero-interaction pairing and key evolution for advanced personal devices," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[16] M. Rostami, A. Juels, and F. Koushanfar, "Heart-to-heart (H2H) authentication for implanted medical devices," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[17] D. Schürmann and S. Sigg, "Secure communication based on ambient audio," in *IEEE Transactions on Mobile Computing*, 2011.

[18] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[19] S. Birnbach, S. Eberz, and I. Martinovic, "Haunted house: physical smart home event verification in the presence of compromised sensors," *ACM Transactions on Internet of Things*, 2021.

[20] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT." in *NDSS*, 2019.

[21] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering physical interaction vulnerabilities in IoT deployments," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[22] M. O. Ozmen, R. Song, H. Farrukh, and Z. B. Celik, "Evasion attacks and defenses on smart home physical event verification," in *NDSS*, 2023.

[23] H. Farrukh, T. Yang, H. Xu, Y. Yin, H. Wang, and Z. B. Celik, "S3: Side-channel attack on stylus pencil through sensors," *Interactive, Mobile, Wearable and Ubiquitous Technologies (UbiComp)*, 2021.

[24] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, 2010.

[25] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, 1984.

[26] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[27] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, 2018.

[28] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to group communication," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 1996.

[29] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 2004.

[30] A. Juels and M. Wattenberg, "A fuzzy commitment scheme," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 1999.

[31] M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval, "Password-based group key exchange in a constant number of rounds," in *International Workshop on Public Key Cryptography*, 2006.

[32] M. Abdalla and D. Pointcheval, "A scalable password-based group key exchange protocol in the standard model," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2006.

[33] D. Fiore, M. I. G. Vasco, and C. Soriente, "Partitioned group password-based authenticated key exchange," *The Computer Journal*, 2017.

[34] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *IEEE Symposium on Security and Privacy (S&P)*, 2003.

[35] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2002.

[36] J. Lee and D. R. Stinson, "Deterministic key predistribution schemes for distributed sensor networks," in *International Workshop on Selected Areas in Cryptography*, 2004.

[37] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[38] "tsfresh," https://tsfresh.readthedocs.io/en/latest/, 2022, [Online; accessed 15-Jul-2022].

[39] "Scikit-fuzzy," https://pythonhosted.org/scikit-fuzzy/, 2022, [Online; accessed 15-Jul-2022].

[40] S. Aranganayagi and K. Thangavel, "Clustering categorical data using silhouette coefficient as a relocating measure," in *IEEE International Conference on Computational Intelligence and Multimedia Applications*, 2007.

[41] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society*, 2001.

[42] C. Costello and P. Longa, "Fourq: Four-dimensional decompositions on a q-curve over the mersenne prime," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2015.

[43] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, "Sha-3 proposal blake," *Submission to NIST*, 2008.

[44] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," *Internet Engineering Task Force*, 2015.

[45] "Fourqlib," https://github.com/microsoft/FourQlib, 2022, [Online; accessed 24-Jul-2022].

[46] "Blake2," https://github.com/BLAKE2/libb2, 2022, [Online; accessed 24-Jul-2022].

[47] "Chachapoly," https://github.com/grigorig/chachapoly, 2022, [Online; accessed 24-Jul-2022].

[48] "Zeromq," https://zeromq.org/, 2022, [Online; accessed 24-Jul-2022].

[49] A. Ihler, J. Hutchins, and P. Smyth, "Adaptive event detection with time-varying poisson processes," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[50] T. Mahmud, M. Hasan, A. Chakraborty, and A. K. Roy-Chowdhury, "A poisson process model for activity forecasting," in *IEEE International Conference on Image Processing (ICIP)*, 2016.

[51] M. Rosulek and N. Trieu, "Compact and malicious private set intersection for small sets," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[52] H. Alemdar, H. Ertan, O. D. Incel, and C. Ersoy, "Aras human activity datasets in multiple homes with multiple residents," in *IEEE International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, 2013.

[53] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan, "Casas: A smart home in a box," *Computer*, 2012.

[54] T. Van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, "Accurate activity recognition in a home setting," in *International Conference on Ubiquitous Computing*, 2008.

[55] J. Mao, S. Zhu, and J. Liu, "An inaudible voice attack to context-based device authentication in smart IoT systems," *Journal of Systems Architecture*, 2020.

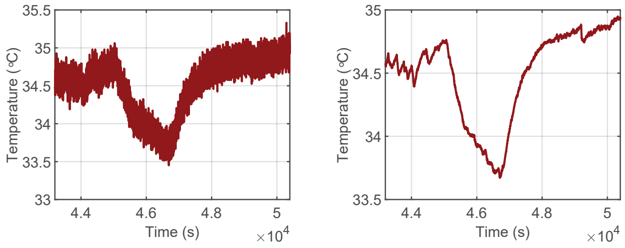
[56] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks," in *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2017.

[57] Y. Tu, S. Rampazzi, B. Hao, A. Rodriguez, K. Fu, and X. Hei, "Trick or heat? manipulating critical temperature-based control systems using rectification attacks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[58] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *USENIX Security*, 2021.

[59] I. Giechaskiel and K. Rasmussen, "Taxonomy and challenges of out-of-band signal injection attacks and defenses," *IEEE Communications Surveys & Tutorials*, 2019.

[60] A. K. Sikder, L. Babun, H. Aksu, and A. S. Uluagac, "Aegis: a context-aware security framework for smart home systems," in *Annual Computer Security Applications Conference (ACSAC)*, 2019.



(a) (b)

Figure 13: Sensor data (a) before and (b) after pre-processing.

[61] Y. Zhang and K. Rasmussen, "Detection of electromagnetic interference attacks on sensor systems," in *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[62] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," in *IEEE Symposium on Security and Privacy (S&P)*, 2005.

[63] C. Li, X. Ji, B. Wang, K. Wang, and W. Xu, "Sencs: Enabling real-time indoor proximity verification via contextual similarity," *ACM Transactions on Sensor Networks (TOSN)*, 2021.

[64] S. Mathur, R. Miller, A. Varshavsky, W. Trappe, and N. Mandayam, "Proximate: proximity-based secure pairing using ambient wireless signals," in *International Conference on Mobile Systems, Applications, and Services*, 2011.

Appendix A.
Sensor Data Pre-processing

Figure 13 shows an example of the temperature sensor signal before and after IoTCupid's pre-processing during event detection. As a result, the signal becomes smoother and amenable for event detection.

Appendix B.
Partitioned GPAKE Security Analysis

Following [33], we provide a formal security analysis of the partitioned GPAKE protocol. We first define two security properties, key secrecy and password-privacy. *Key secrecy* means that assuming the passwords (evidences) are distributed uniformly at random and only a constant number of passwords can be checked by the adversary on each online attack, the probability that the adversary can derive a group key with legitimate devices is negligible. *Password-privacy* ensures that an adversary that conducts an online attack cannot gain any information on the passwords used by legitimate devices, including which devices actually share the same password. We provide a proof sketch below, and refer to [33] for the full proof. We note that we reduce our protocol's security to computational elliptic curve Diffie-Hellmann (ECDH) instead of traditional computational DH in [33], although the reductions remain the same.

Theorem 1. Let the encryption scheme in Table 2 be both unforgeable and chosen plaintext semantically-secure. Then, the protocol in Table 2 is a correct partitioned group password-authenticated key exchange scheme that achieves key secrecy and password-privacy under the elliptic curve computational Diffie-Hellman assumption in the random oracle and ideal cipher model.

Proof Sketch. Correctness. In an honest execution of the partitioned GPAKE protocol where no adversaries are involved, it is straightforward to verify that all devices that share the same password derive the same session ids and a shared group key. This follows from the fact that the devices that share the same password first derive session keys with each other, and then use these session keys to broadcast a random value. The devices with the same session key can decrypt the random values and add them to derive the group key. All the devices that share the same password conduct these steps, and thus, they all add the same random values shared among them, deriving a correct group key.

Key Secrecy. An adversary can target different stages of the protocol to gain information about the group keys.

First, we consider an adversary ($\mathcal{A}$) who has a valid tuple $(d_i,\ d_j,\ X_{i,d_i},\ X_{j,d_j},\ x_{i,d_i} \cdot X_{j,d_j}\ mod\ q)$ in the group key establishment protocol. Such an adversary can derive the same group key with legitimate devices as it has a valid session key. However, we show that if such an adversary exists, we can construct another adversary ($\mathcal{B}$) that can break the computational elliptic curve Diffie-Hellman (ECDH) assumption. Particularly, given the input of $x \cdot P$ and $y \cdot P$, $\mathcal{B}$'s goal is to derive $x \cdot (y \cdot P)$. For this, $\mathcal{B}$ picks two random user indices (i and j) and a random execution number. It next sets the i^{th} device's X_{i,d_i} as $x \cdot P$ and j^{th} device's X_{j,d_j} as $y \cdot P$. $\mathcal{B}$ then uses $\mathcal{A}$ as a subroutine and returns $x_{i,d_i} \cdot X_{j,d_j}\ mod\ q$. If $\mathcal{B}$ guesses the random user indices and the execution number correctly, it correctly breaks the computational ECDH problem and derives $x \cdot (y \cdot P)$. Therefore, the security of the session key of the partitioned GPAKE protocol reduces to the hardness of the computational ECDH problem.

We next consider an adversary who guesses a password correctly. Such an adversary can participate in the protocol and derive the same keys with legitimate devices by following an honest execution of the protocol. However, assuming the password's are uniformly distributed, the probability of such an adversary existing is $q/2^{|pw|}$ where q is the number of times the adversary can guess a password in an online attack and $|pw|$ is the bit-length of the password. Therefore, if the password has a sufficient bit-length, the probability of such an attack is negligible.

Lastly, we consider an adversary who modifies the messages in the broadcast stages of the protocol such that it crafts messages that can decrypt correctly by legitimate devices to derive shared keys with them. Yet, existence of such an adversary reduces to the unforgeability of the encryption scheme used.

Password-privacy. The proof for password-privacy is very similar to the one for key secrecy. This is because after the devices encrypt their public keys using their passwords with an unforgeable and CCA-secure encryption scheme, the protocol messages become independent of the passwords. Therefore, a polynomial-time adversary who has no prior knowledge about the passwords cannot learn any information about them from the protocol under the hardness of the computational ECDH problem and the unforgeability of the encryption scheme.

□

Following this proof, the security of IoTCupid relies on (1) the randomness of the passwords, (2) the hardness of the computational elliptic curve Diffie-Hellman problem, and (3) the security of the encryption scheme used in GPAKE. We show in our evaluation that the inter-event timings provide enough entropy to be used as passwords. The security of the computational ECDH problem and used symmetric encryption scheme are already well-established.