T2Pair: Secure and Usable Pairing for Heterogeneous IoT Devices

Xiaopeng Li
University of South Carolina
xl4@email.sc.edu

Qiang Zeng*
University of South Carolina
zeng1@cse.sc.edu

Lannan Luo
University of South Carolina
lluo@cse.sc.edu

Tongbo Luo
JD.com
tongbo.luo@jd.com

ABSTRACT

Secure pairing is key to trustworthy deployment and application of Internet of Things (IoT) devices. However, IoT devices lack conventional user interfaces, such as keyboards and displays, which makes many traditional pairing approaches inapplicable. Proximity-based pairing approaches are very usable, but can be exploited by co-located malicious devices. Approaches based on a user's physical operations on IoT devices are more secure, but typically require inertial sensors, while many devices do not satisfy this requirement. A secure and usable pairing approach that can be applied to heterogeneous IoT devices still does not exist. We develop a technique, *Universal Operation Sensing*, which allows an IoT device to sense the user's physical operations on it without requiring inertial sensors. With this technique, a user holding a smartphone or wearing a wristband can finish pairing in seconds through some very simple operations, e.g., pressing a button or twisting a knob. Moreover, we reveal an inaccuracy issue in original fuzzy commitment and propose *faithful fuzzy commitment* to resolve it. We design a pairing protocol using faithful fuzzy commitment, and build a prototype system named Touch-to-Pair (T2Pair, for short). The comprehensive evaluation shows that it is secure and usable.

CCS CONCEPTS

• **Security and privacy** → **Security services**; **Network security**;
• **Computer systems organization** → *Sensor networks.*

KEYWORDS

Pairing; Internet of Things; fuzzy commitment; PAKE

ACM Reference Format:
Xiaopeng Li, Qiang Zeng, Lannan Luo, and Tongbo Luo. 2020. T2Pair: Secure and Usable Pairing for Heterogeneous IoT Devices. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3372297.3417286

*Corresponding author.

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417286

1 INTRODUCTION

Internet-of-Things (IoT) devices are widely deployed, generating great impacts on various industries and our daily lives. A study estimates that the number of installed IoT devices will reach 75 billion by 2025 [44]. As pairing is to establish a communication channel, a convenient and secure pairing approach is critical to wide deployment and trustworthy application of IoT devices.

To pair a desktop or smartphone to an existing network, the user simply inputs the network password to the device. However, most IoT devices do not have user interfaces (UIs) for inputting passwords, and thus cannot apply this approach.

Many IoT device vendors have the user use her personal mobile device (e.g., a smartphone) to connect the IoT device's hotspot and input the home WiFi password. If the network connection is secure (how to ensure it is a challenge [4, 12]), the IoT device can obtain the password from the mobile device securely. This way, the problem of pairing an IoT device is reduced to mutual authentication between the IoT device and user's mobile device [48]. We also leverage a user's mobile device for pairing IoT devices.

The literature has proposed many IoT pairing approaches, which can be divided into at least two categories. The first category establishes pairing on proximity between devices to be paired [4, 12, 14, 32, 42, 48, 49]. It can be further divided into two sub-categories. (a) As all IoT devices have certain wireless communication capabilities, some approaches (such as Move2Auth [48]) prove proximity by exploiting characteristics of wireless signals [12, 48, 49]. (b) Other approaches (such as Perceptio [14]) make use of the ambient context, like audio and light, to prove proximity [4, 32, 42]. Approaches in this category usually feature usability; however, they can be exploited by co-located malicious devices.

Approaches in the second category require the user to physically contact or operate the IoT device [15, 30, 45]. For example, ShaVe/ShaCK [30] has a user hold her smartphone and the IoT device together in one hand and shake them, and then the knowledge of the shared movement sequence is used for pairing. They are generally more secure, as physical operations are involved in the pairing process. But they require IoT devices to have inertial (or touch) sensors that sense the user's operations, while many IoT devices do not have such sensors.

We consider IoT devices that (1) do not have sophisticated UIs like keyboards, (2) may be located close to untrusted or malicious devices (for example, a hospital may contain a mix of devices that belong to the doctors, patients, or attackers), (3) do not necessarily have inertial sensors, and (4) may be mobile or mounted, installed indoors or outdoors. A secure and usable pairing approach that is

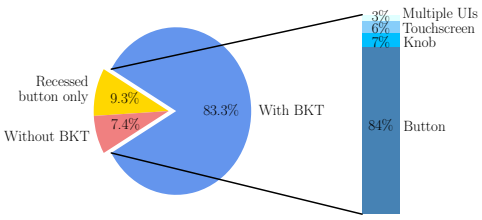


Figure 1: Distribution of physical UIs on 270 popular IoT devices. "*With BKT*" means the device has a normal button, knob or touchscreen; "*Recessed button*" refers to a small hole that can be pressed using, e.g., a ball-point pen.

applicable to such heterogeneous IoT devices still does not exist. For example, Perceptio [14] can only be applied to IoT devices installed indoors without co-located malicious devices. We propose a system named Touch-to-Pair (T2Pair, for short) that is not only secure but applicable to heterogeneous IoT devices. Moreover, it is very easy to use—a user wearing a wristband (or holding a smartphone) only needs to *touch*, in the form of some very simple operations, the IoT device for a few seconds to finish pairing.

T2Pair is built on a technique, *Universal Operation Sensing (UOS)*, which allows an IoT device to sense user operations on an IoT device without requiring any inertial sensors. When a user wearing a wristband or holding a smartphone *touches* an IoT device, such as *pressing* its button a few times, *twisting* its knob back and forth, or *swiping* its touchscreen[1] in a zig-zag way, *salient points* arise when the button is pressed/released or the twisting/swiping changes its direction. We share an insight with P2Auth [23] and Perceptio [14] that every IoT device has a clock. To make the technique widely applicable, we use *timestamps* to describe salient points. On the user's wristband (or smartphone) side, the same set of salient points can be identified by analyzing the motion data captured by the built-in Inertial Measurement Unit (IMU). Subsequently, the wristband and the IoT device can make use of the knowledge of the salient points to authenticate each other.

T2Pair can be widely applied to most IoT devices on the market. As shown in Figure 1, our survey of 270 most popular IoT devices on Amazon (ranked by the number of reviews) indicates that 92.6% of them have a normal button, knob, touchscreen, or recessed button. For example, an Amazon smart plug, which does not need much interaction, has a button for pairing and turning it on/off.

We assume the adversary has full control over all communication channels. Thus, given that the wristband and the IoT device do not have prior security association, how to perform secure authentication in the presence of attacks, such as man-in-the-middle (MITM) attacks, is a challenge. This is a critical difference between IoT *pairing* and *authentication* [23], as the latter usually assumes the IoT device is already securely associated with the user's token or device used for authentication. Another challenge is that the user's wristband and the IoT device may have small differences with regard to the observations of salient points. To overcome the two challenges, we first tried fuzzy commitment [20], which incorporates cryptography and error-correcting code, such that the two sides (wristband and IoT device) can securely authenticate each other without leaking the knowledge to the MITM adversary and, meanwhile, tolerate small differences aforementioned.

However, this attempt failed. Our experiment shows that the original fuzzy commitment leads to a high pairing-failure rate, and our investigation reveals that sometimes small differences between observations lead to very different encodings, while large differences result in similar encodings. We thus propose *faithful* fuzzy commitment, which makes sure distances between encodings faithfully reflect differences between observations.

Furthermore, we uncover a security weakness under trained mimicry attacks (i.e., an attacker who is familiar with the victim user mimics her pairing operations) and show how to enhance T2Pair without harming usability. A prominent advantage of T2Pair is that it does not need clock synchronization, as it uses intervals between salient points for encoding, which makes the pairing resilient to attacks that interfere with clock synchronization.

We implement T2Pair and evaluate it on prototypical IoT devices with buttons, knobs or touchscreens. The evaluation results show that T2Pair has very low false rejection/acceptance rates. The pairing takes only 7 seconds. A user study is performed, confirming high usability of T2Pair. We make the following contributions.

- We develop *Universal Operation Sensing* (UOS), which allows IoT devices to sense user operations and uses timestamps to describe them, without requiring inertial sensors. We reveal the weakness of pairing based on UOS under trained mimicry attacks and enhance it to attain both usability and security.
- We propose *faithful fuzzy commitment*, such that small distances between encodings faithfully reflect small differences between values being encoded, and vice versa. A pairing protocol based on faithful fuzzy commitment and password-authenticated key exchange [5] is proposed, with strong resilience to attacks.
- Built on the two techniques, we propose and implement T2Pair. A user only needs to *touch* the IoT device, in the form of pressing a button, twisting a knob, or swiping a touchscreen, to finish pairing. The paring method can be applied to heterogeneous IoT devices without requiring inertial sensors, and largely eliminates the threat of co-located malicious devices. The comprehensive evaluation shows that T2Pair is secure and usable.

The rest of the paper is organized as follows. Section 2 describes the system overview and threat model. Section 3 presents UOS, Section 4 the protocol, and Section 5 implementation details. Section 6 describes the dataset collection and Section 7 the evaluation. Related work is discussed in Section 8, and limitations in Section 9. The paper is concluded in Section 10.

2 SYSTEM OVERVIEW AND THREAT MODEL

Given an IoT device, our goal is that a user can utilize her personal mobile device, called a *helper*, such as a smartphone, fitness tracker, smartwatch, or smart ring [34], to securely pair an IoT device by quickly performing simple operations on the device.

We take the device with a single button, as an example, to illustrate the overview of our pairing mechanism. Figure 2 shows a block diagram of T2Pair, where a user wearing a helper presses the button a few times to conduct the pairing. In the process, the device makes use of its clock to describe the button-pressing events

[1]Touchscreens allow users to input passwords directly; however, the usability of inputting a WiFi password of eight characters or longer on a small touchscreen is poor. We thus extend T2Pair to touchscreens.

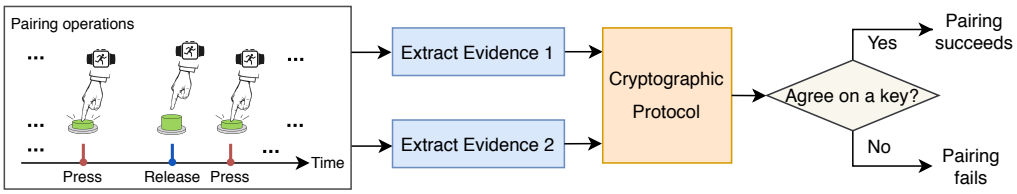


Figure 2: Architecture of T2Pair (a wristband as the helper and an IoT device with a button as an example).

and derives a piece of *evidence* of the pairing operations, while the helper collects readings from its accelerometer/gyroscope, and independently generates another piece of *evidence*. We further design a protocol that allows the two parties to compare the evidence without leaking it to the MITM attacker. If the difference is small enough, they will be able to agree on a session key.

Threat Model. The attacker $\mathcal{A}$ has one or multiple of the following goals. (G1) The victim $\mathcal{V}$'s helper H_v pairs $\mathcal{A}$'s malicious device D_a, so H_v is fooled to exchange data (such as the WiFi password, photos, sensor data) with D_a. (G2) The IoT device D_v pairs a malicious helper H_a of $\mathcal{A}$, so D_v is fooled to exchange data with H_a. (G3) $\mathcal{A}$ cracks the symmetric key and uses it to eavesdrop and/or manipulate the communication between H_v and D_v.

We assume $\mathcal{A}$ has full knowledge of our pairing protocol. Like [14, 32], our work follows a standard Dolev-Yao adversary model [11]; i.e., the adversary has full control over all communication channels. Based on this, $\mathcal{A}$ may launch MITM attacks, e.g., by intercepting D_v's (H_v's, resp.) message sent to H_v (D_v, resp.) and sending faked messages instead. We further consider the attacks below.

Mimicry Attacks. If $\mathcal{A}$ has a visual observation of $\mathcal{V}$, $\mathcal{A}$ may launch a mimicry attack by mimicking $\mathcal{V}$'s pairing operations, in order to achieve G1 and/or G2. We examine the following threat scenarios where $\mathcal{A}$ has increasing capabilities. **MA-obstructed:** $\mathcal{A}$ can see $\mathcal{V}$, but cannot directly see $\mathcal{V}$'s hand motions due to certain obstructions. **MA-clear:** $\mathcal{A}$ can clearly see $\mathcal{V}$'s hand motions by selecting an optimal viewing angle. **MA-trained:** $\mathcal{A}$ is familiar with $\mathcal{V}$ and trained by learning the pairing operations of $\mathcal{V}$ before launching a mimicry attack described in MA-clear.

Brute-Force Attacks. **BF-online:** During the pairing process, $\mathcal{A}$ tries every possible piece of evidence until it hits a correct one, so H_v and/or D_v are fooled to pair the attacker. **BF-offline:** $\mathcal{A}$ may collect all the pairing traffic and perform offline analysis in order to crack the established key after pairing.

Attacks beyond Scope. $\mathcal{A}$ may be equipped with a camera and computer-vision techniques to analyze $\mathcal{V}$'s hand movements. Like other pairing approaches that require physical operations, such as ShaVe/ShaCK [30], T2Pair is also vulnerable to such attacks. At a user's home or office, however, the attack is not easy to launch, as it requires an attacker-controlled camera that points at the user.

$\mathcal{A}$ may launch Denial-of-Service (DoS) attacks to manipulate the communication channel and disrupt the pairing. But if failed pairings occur repetitively, the helper can alert the user, who can take actions to investigate or report the attacks.

3 PAIRING OPERATIONS AND EVIDENCE

We introduce pairing operations in Section 3.1, study operation sensing in Section 3.2, and present evidence extraction in Section 3.3.

3.1 Pairing Operations

To devise usable and effective pairing operations, the UI properties of IoT devices should be taken into consideration. According to our survey, the most common UIs of resource-constrained IoT devices include *buttons* (e.g., AWS IoT Button [2]), *knobs* (e.g., Nest Thermostats [13]), and *touchscreens* (usually small, e.g., Honeywell T9 Smart Thermostats [16]). Thus, our design of Universal Operation Sensing (UOS) considers the three types of UIs: buttons, knobs, and touchscreens, and includes the following pairing operations.

- *Pressing the button a few times with one or more random pauses added.* A *"pause"* here means that after the button is pressed down, the user holds, intentionally, for a random short time before releasing it. Note that it does *not* refer to the *natural* pause when a user presses down a button and naturally holds shortly before releasing it. Our experiments reveal that *UOS without pauses is weak under trained mimicry attacks (Section 7.2), while UOS with pauses is much more resilient (Section 7.2).*
- *Twisting the knob back and forth with one or more random pauses added.* When the knob is twisted, the micro-controller on the IoT device can detect the direction and amount of current twisting. To add a pause, the user intentionally holds for a random short time right prior to changing the twisting direction.
- *Zig-zag swiping on the touchscreen with one or more random pauses added.* Rather than asking the user to draw a specific shape or pattern on a small screen, which harms usability, the user simply swipes the screen using a finger from left to right and back again for a few times. Similarly, for better security the user can hold for a short time right before changing the swiping direction.

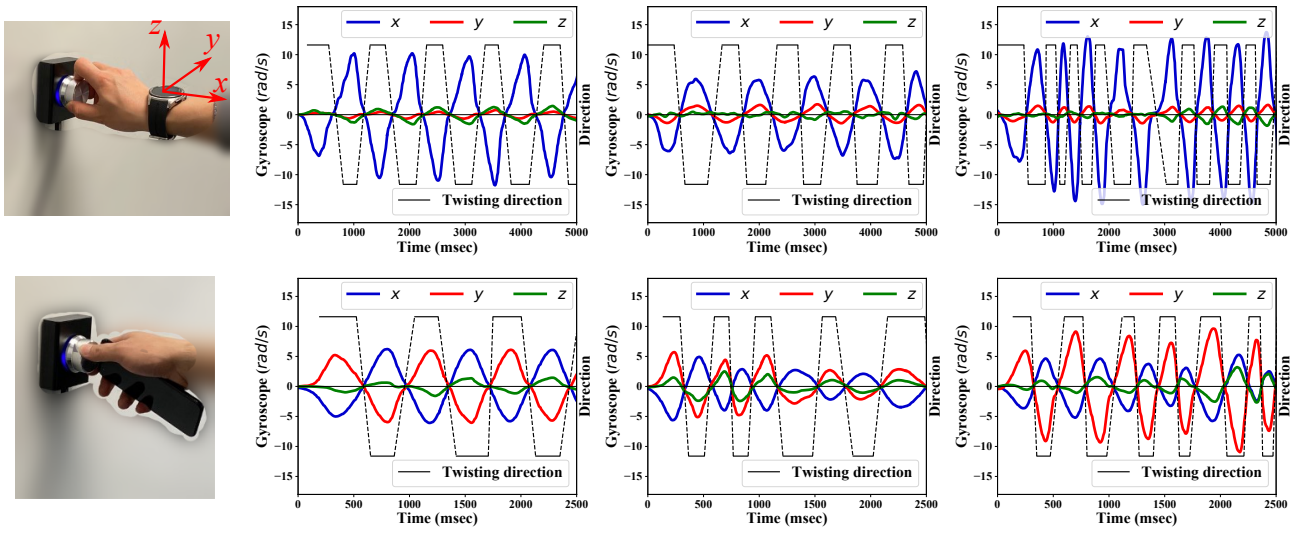
All the operations are simple and easy to perform. More importantly, each involves *"crispy"* speed/direction changes, which can be sensed by both IoT device and the helper (Section 3.2). Similar operations, without pauses, were used in our prior work P2Auth for authentication [23], but it was unclear how they could be used for pairing and whether they were resilient to trained mimicry attacks.

3.2 Study of Sensing Pairing Operations

It is reliable (and trivial) to use the controller or sensor of an IoT device to sense the button-down/button-up, knob twisting or screen swiping. We collect the data readings from the IoT device, along with the corresponding time, and regard them as ***ground truth***.

On the side of the helper, it uses the embedded IMU to collect motion data during pairing operations. It is thus critical to explore the following questions. (1) Does the IMU data show certain correlations with the ground truth? (2) Are the correlations reliable across different devices, users and pairing instances?

To this end, we ask users to perform each of the three types of pairing operations (no pauses for simplicity of discussion). The user can decide the posture of her hand and wrist, and use different



Demonstration User 1 User 2 User 3

Figure 3: Gyroscope data captured when three (3) users twist knobs. The black lines show the ground truth of twisting direction.

helpers (a smartwatch or smartphone). For example, the photo at the upper left of Figure 3 shows a user wearing a smartwatch, while that at the lower left shows a user holding a smartphone.

We use knob twisting as an example, as shown in Figure 3. We observe a strong correlation between the twisting operations and the gyroscope data, which measures the angular velocity of its rotation: regardless of the user and her posture of hand and wrist, the gyroscope data changes from positive (resp. negative) values to negative (resp. positive) values, as the rotation direction changes according to the ground truth. (On the other hand, the acceleration does not show such a strong correlation, as twisting affects the angular velocity significantly, rather than the linear speed.)

This strong correlation can be observed in *at least* one axis of the gyroscope data. E.g., when the knob is rotated roughly around the x-axis of the smartwatch (the upper row of Figure 3), the gyroscope signal along the x-axis (blue line) changes significantly as the rotation direction (black line) changes. In the lower row of Figure 3, there exist significant signal changes in both the x-axis (blue line) and y-axis (red line) of the gyroscope data. We thus conclude that the gyroscope and knob twisting have a strong correlation in at least one axis of the gyroscope data, which features significant value changes. It is straightforward to detect the axis of data that shows the most significant value changes, and we call it the *dominant axis*.

Strong correlations are observed for the other two types of pairing operations (see Figures 11 and 12 in Appendix B). For each button pressing, the acceleration data along at least one axis has a sharp peak or valley. The gyroscope data does not have significant changes—when the user's finger presses a button, the acceleration reaches a peak quickly because the finger's moving speed suddenly decreases to zero, while the gyroscope data is not affected much. Like twisting knobs, in the case of zig-zag swiping, the gyroscope data changes significantly as the swiping direction changes.

3.3 Extracting Evidence

The strong correlations provide basis for comparison, but it is not easy to directly compare the two sequences of heterogeneous data: the IoT device receives a sequence of input events, while the helper's IMU generates a sequence of motion data. To address it, we propose to extract *salient points* from the data, and use the *occurrence time* of each point to represent it, making it easier to compare. Below, we use d_1 to denote the IoT device, and d_2 the helper.

3.3.1 *Salient Points on the IoT Device Side.*

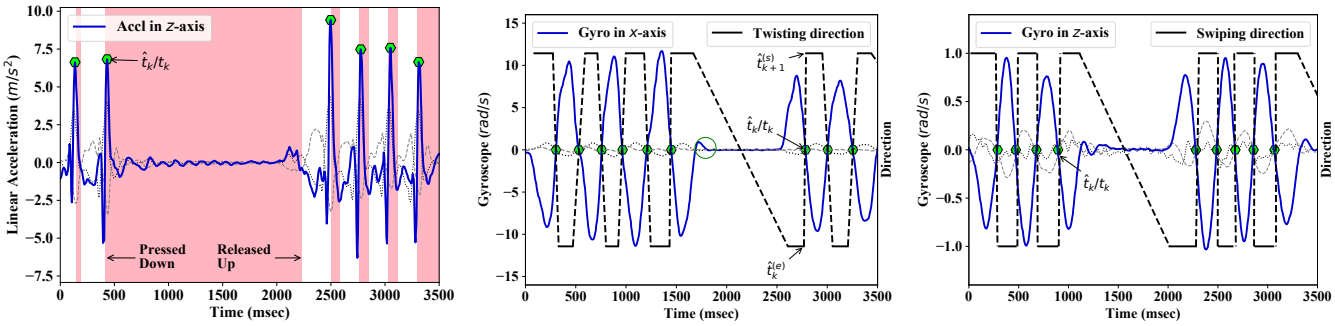
Pressing buttons. Pressing a button once generates two events: `PressedDown` and `ReleasedUp`, as shown in Figure 4(a) (the pink area shows the duration between two consecutive `PressedDown` and `ReleasedUp` events). We adopt the `PressedDown` events during pairing as the *salient points*, as they can be sensed on both sides (see Section 3.2). We thus obtain the timestamp sequence $S_{d_1} = \{\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_n\}$, where $\hat{t}_k$ is the occurrence time of the kth `PressedDown`. It is worth noting that a random pause just introduces a relatively longer time span between two consecutive salient points. We thus do not explicitly identify and represent pauses.

Twisting knobs. Each rotation-direction change is handled as a salient point. As shown in Figure 4(b), the kth salient point is represented using $\hat{t}_k \approx \frac{1}{2}(\hat{t}_k^{(e)} + \hat{t}_{k+1}^{(s)})$, where $\hat{t}_k^{(e)}$ denotes the end time of the kth rotation and $\hat{t}_{k+1}^{(s)}$ the start time of the $(k+1)$th rotation. The timestamps $\hat{t}_k^{(e)}$ and $\hat{t}_{k+1}^{(s)}$ should be close for identifying a salient point. We thus obtain $S_{d_1} = \{\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_{n-1}\}$, where $\hat{t}_k$ is the occurrence time of the kth salient point.

Swiping touchscreens. Each swiping-direction change is handled as a salient point, as shown in Figure 4(c). We extract a timestamp sequence $S_{d_1} = \{\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_{n-1}\}$, where $\hat{t}_k$ is the kth salient point.

3.3.2 *Salient Points on the Helper Side.*

Pressing buttons. Figure 4(a) shows an example of pairing via pressing a button. In this case the z-axis of acceleration is the dominant axis (see Section 3.2); the signal along the other two axes are in dashed grey lines. At each salient point of the ground truth, i.e., `PressedDown` event, a sharp peak is observed. We retrieve the occurrence time of each sharp peak, and derive the sequence $S_{d_2} = \{t_1, t_2, \ldots, t_m\}$, where t_k is the time of the kth sharp peak.



(a) Pairing via pressing a button (b) Pairing via twisting a knob (c) Pairing via swiping a touchscreen

Figure 4: Salient points for the three types of pairing operations.

Twisting knobs. According to our study of motion data (see Section 3.2), we use the gyroscope data for detecting salient points, which correspond to rotation-direction changes. In the example shown in Figure 4(b), the x-axis is the dominant axis. As the rotation direction of the IoT device changes, the signal sign of the gyroscope changes as well. Therefore, we detect salient points by searching for the points of signal sign switches during large-amplitude changes. We extract a sequence of timestamps of all the salient points. The timestamp sequence is denoted as: $S_{d_2} = \{t_1, t_2, \ldots, t_m\}$, where t_k refers to the occurrence time of the kth salient point.

During a pause, the gyroscope readings are near zero. But there may exist small fluctuations, especially at the start (denoted by the green circle in Figure 4(b)). To avoid detection of false salient points, such fluctuations are filtered via simple thresholding.

Swiping touchscreens. Each direction change of the swiping produces a salient point in the gyroscope data trace. As shown in Figure 4(c), each salient point corresponds to a sharp sign change due to a swiping-direction change. We obtain a sequence of timestamps: $S_{d_2} = \{t_1, t_2, \ldots, t_m\}$, where t_k is the time of the kth salient point.

Big silence. It is critical to identify the first salient point. After the pairing is initiated (e.g., by long pressing a button), as the user's hand approaches the IoT device's button/knob/screen, there may exist some noisy motion data that looks like salient points. To address this, we simply ask the user to touch the button/knob/screen and hold shortly, around 2–3 seconds, before performing the pairing operations. This way, the "*big silence*" in the motion data works as an indication that pairing operations follow and the detection of salient points from the motion data can start.

3.3.3 No Clock Synchronization. To eliminate the need of clock synchronization, we convert each timestamp sequence into a series of time intervals using the equations $\hat{i}_k = \hat{t}_{k+1} - \hat{t}_k$ and $i_k = t_{k+1} - t_k$ for S_{d_1} and S_{d_2}, respectively. We then concatenate the time intervals and call them *evidence*: $E_{d_1} = \{\hat{i}_1 || \hat{i}_2 || \cdots || \hat{i}_{q-1}\}$ and $E_{d_2} = \{i_1 || i_2 || \cdots || i_{p-1}\}$, where E_{d_1} represents the evidence collected by the IoT device, and E_{d_2} by the helper.[2]

[2]Note that clock drift during pairing does not cause an issue here, as the pairing operations take only around three seconds (Section 7.5), leading to 3ms of drift in the worst case (see Section 3.2 of [26]); such small differences between evidence are tolerated by our protocol based on fuzzy commitment (Section 4.1).

4 PROTOCOL FOR KEY AGREEMENT

Once two pieces of evidence are extracted, the two sides use the evidence to mutually authenticate each other and establish a key.

4.1 Challenges and Solution

How to perform secure mutual evidence verification, when there are powerful attacks such as man-in-the-middle (MITM) attacks, is a challenge. Another challenge is that the wristband and the IoT device may have small differences, e.g., due to sensor readings and clock drift, with regard to the observations of salient points.

Failed attempt. To address the two challenges, we first adopt a fuzzy commitment scheme (FCS) building on error correcting codes [20]. Fuzzy commitment schemes have been utilized for proximity based pairing [14, 32, 42]. It allows mutual evidence verification without disclosing the evidence to MITM attackers and handles small differences between two pieces of evidence. The sender converts its evidence to an encrypted message, which can be successfully opened only if the receiver owns the evidence that is *similar* to the sender's in the metric of *Hamming distance* [20]. We call the original fuzzy commitment as *vanilla fuzzy commitment*.

To conduct the vanilla fuzzy commitment, the evidence needs to be firstly encoded into a bit-representation. Previous studies convert a value directly to its binary representation [14]. But the encoding method may incorrectly consider two dissimilar (resp. similar) evidence sequences as similar (resp. dissimilar).

For example, given the interval values {121} and {57}, which are encoded as "0111 1001" and "0011 1001", respectively, based on the definition of Hamming distance, which is the number of different digits in the two bit strings, we obtain $\mathrm{Ham}(121, 57) = 1$. Since their Hamming distance is very small, the two intervals are considered similar, while in fact their difference is large. As another example, the interval values, {128} and {127}, can be represented as "1000 0000" and "0111 1111", respectively. We have $\mathrm{Ham}(128, 127) = 8$. Thus, the vanilla fuzzy commitment incorrectly considers the two similar interval values very *different*.

In short, while the vanilla fuzzy commitment works fine in certain applications, e.g., when the values fall in large ranges, it does not work well in our case as the difference of intervals is not huge.

Solution. To address the problem, we propose *faithful fuzzy commitment*, which encodes each time interval by first dividing the interval value by a base value to tolerate small differences and

Table 1: The pairing protocol.

Device d_1		Device d_2
Phase 1: Initialization		
Initiates the pairing		
Phase 2: Extracting Evidence		
$E_{d_1} = Time\_Int\_Seq(d_1)$		$E_{d_2} = Time\_Int\_Seq(d_2)$
if self-checking fails, aborts		if self-checking fails, aborts and reminds the user
Phase 3: Fuzzy Commitment		
① picks a random value $P \in \mathbb{F}_{2^k}^m$		
② $\lambda \in \mathbb{F}_{2^k}^n \xleftarrow{\text{encode}} \text{RS}(2^k, m, n, P)$		
③ commits: $\delta = e(E_{d_1}) \oplus \lambda$	$\xrightarrow{\delta}$	④ decommits: $\lambda' = e(E_{d_2}) \oplus \delta$
		⑤ $P' \xleftarrow{\text{decode}} \overline{\text{RS}}(2^k, m, n, \lambda')$
Phase 4: PAKE		
⑥ picks a; $A = g^a \bmod p$; $w = \text{h}(P)$	$\xrightarrow{\text{E}(w, A)}$	⑦ picks b; $B = g^b \bmod p$; $w' = \text{h}(P')$
⑨ $K = B^a \bmod p$	$\xleftarrow{\text{E}(w', B\|\|C_1)}$	⑧ $K' = A^b \bmod p$; picks a challenge C_1
⑩ picks a challenge C_2	$\xrightarrow{\text{E}(K, C_1\|\|C_2)}$	⑪ if C_1 is not received, aborts
⑫ if C_2 is not received, aborts	$\xleftarrow{\text{E}(K', C_2)}$	

reduce the encoding length and then representing the result as a sequence of *consecutive* "1" and "0" bits. The distance then can be computed as the Hamming distance between their encodings.

Given the base value B and an interval value i, we derive $n = \lfloor i/B \rfloor$. We make sure all intervals have the same length L of encodings. Then, the interval is represented as n *consecutive* "1" bits, with another $L - n$ "0" bits appended to the end. So the interval with the value i is encoded as:

$$e(i) = \underbrace{\overbrace{1, 1, \cdots, 1}^{n}, 0, 0, \cdots, 0}_{L} \tag{1}$$

A large base value B leads to more efficient key agreement but less precise evidence comparison, and vice versa. We discuss how to select the base value B in Section 7.4. Assume $B = 4$ and consider the two examples above. {121} can be encoded as $\lfloor 121/4 \rfloor = 30$ consecutive "1" bits follwed by $L - 30$ "0" bits. {57} can be encoded as 14 consecutive "1" bits followed by $L - 14$ "0" bits. Thus, we have $\text{Ham}(e(121),\ e(57)) = 16$, which is much larger than $\text{Ham}(e(128),\ e(127)) = 1$. Therefore, our faithful fuzzy commitment overcomes the limitation of the vanilla fuzzy commitment and makes correct decisions.

The encoding can only represent a value between 0 and $L * B + (B - 1)$. It works well in our case as intervals do not fall in a huge range. We do not claim it as a general encoding solution.

4.2 Protocol Details

Table 1 shows our protocol, which consists of four phases. (1) *Initialization.* Almost all commercial off-the-shelf devices have some built-in method to initiate the pairing process (e.g., long pressing a button). (2) *Extracting Evidence.* As the user wearing/holding the helper device preforms pairing operations on the IoT device, each side extracts evidence independently. Here, ***self-checking*** is enforced: if there are no pauses detected, the pairing aborts and the helper reminds the user of adding one or more pauses. As illustrated in Section 7.2, pauses are critical to defeat trained mimicry attacks. (3) *Fuzzy Commitment.* The two devices use the evidence to communicate a "password". (4) *Password-Authenticated Key Agreement* (PAKE). The devices make use of the "password" to agree on a session key. Below we interpret the details of Phases (3) and (4).

Fuzzy Commitment. This phase is accomplished using faithful fuzzy commitment and the Reed-Solomon (RS) error correcting code [38]. Given a set of possible words $\mathbb{P}$ each with m bits, a set of possible codewords $\mathbb{Q}$ each with n bits, and $n > m$, RS codes are initialized as $\mathbb{P} = \mathbb{F}_{2^k}^m$, and $\mathbb{Q} = \mathbb{F}_{2^k}^n$, where k is a natural number and 2^k denotes the number of words (codewords) in $\mathbb{P}$ ($\mathbb{Q}$).

The device d_1 first randomly selects a "password" $P \in \mathbb{P}$ using a key generation algorithm (①). Then, P is uniquely mapped to a codeword $\lambda \in \mathbb{Q}$ using the RS encoding function (②). This step adds redundancy to the original words with $n > m$, based on polynomials over Galois fields [38], to support error correcting. After that, the commitment process produces an encryption of the codeword λ by hiding it using the evidence E_{d_1}. It performs an exclusive-OR ($\oplus$) between $e(E_{d_1})$ and λ, and obtains the commitment $\delta = e(E_{d_1}) \oplus \lambda$ [20, 32], where $e()$ is the encoding described in Section 4.1 (③). d_2 then conducts decommitment. It uses the received δ and $e(E_{d_2})$ to obtain a codeword $\lambda' = e(E_{d_2}) \oplus \delta$ (④). Finally, λ' is decoded to P' using the RS decoding function (⑤). Readers are referred to [20] for more detailed interpretation of fuzzy commitment.

The effects of this phase are as follows. (1) If E_{d_1} and E_{d_2} are close enough, d_2 is able to derive a value $P' = P$; otherwise, $P' \neq P$. (2) However, at this moment, d_2, no matter it is a benign device (because of false rejections) or an attacker (who has derived P' using a guess of the evidence), is not sure whether $P' = P$.

The reason we call P, which is actually a random key value, a "password" is to take offline brute-force attacks (BF-offline) into consideration. If P is directly used as the session key, an offline attacker who has collected the traffic can try every possible evidence and repeat ④ and ⑤ until he finds a key that can decrypt the traffic; thus, the entropy of evidence (see Section 7.3) disqualifies P to work

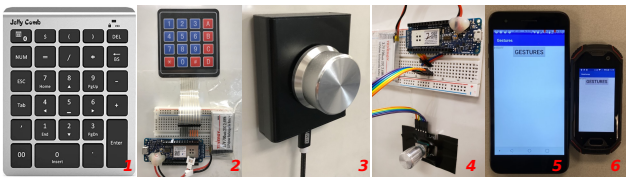


Figure 5: Six devices are used in our experiments, including two keypads (a plastic keypad labeled as 1, and a rubber one as 2; in either case, we only use *one button* for pairing); two knobs (a large knob labeled as 3, and a small one as 4); two touchscreens (a 5.2" Google Nexus 5X labeled as 5, and a 2.45" Unihertz Atom labeled as 6).

as a secure shared key. We thus use PAKE, which securely generates a high-entropy shared key from a low-entropy password [8].

PAKE. We use Diffie-Hellman Encrypted Key Exchange (DH-EKE) [5], which has led to the PAKE family of methods in IEEE P1363.2 [18], but many other PAKE methods should also work. DH-EKE is a DH-based key exchange method that makes use of a password to defeat MITM attacks, as both A and B are transmitted in encrypted messages (⑥ and ⑧). Note the base g and the modulus p are public knowledge, h( ) is a cryptographic hash function, and E( ) is a symmetric encryption function. If $P' \neq P$, d_2 will receive a value different from the challenge C_1 it has picked (⑪); otherwise, after ⑪ and ⑫, d_1 and d_2 establish a key $K = K'$.

Parameter Consideration. The security of λ is primarily governed by the size (i. e., 2^k) of the set of codewords [20]. To provide strong security, k should be larger than 80, which is comparable to RSA-1024. By applying RS, a word of length m is uniquely mapped to a codeword of length n. The maximum number of bits between two codewords that can be corrected is $Thr = \lfloor \frac{n-m}{2} \rfloor$. Thus, if and only if the Hamming distance between two pieces of evidence satisfies $\text{Ham}(e(E_{d_1}), e(E_{d_2})) \leq Thr$, the symmetric key $P' = P$ can be established. The value selection for Thr is studied in Section 7.1.

Resilience to Brute-Force Attacks. The forward secrecy of DH ensures that even if P is cracked offline (e.g., recording a video of the user to assist offline analysis of P, or enumerating every possible evidence to reveal P), it cannot be used to reconstruct the session key. Thus, offline brute-force (***BF-offline***) attacks will fail. ***BF-online*** will not work either. As PAKE attains zero-knowledge password proof [18], an active (man-in-the-middle) attacker can perform exactly one guess (unless he gets it right, he learns no information), and a passive eavesdropper learns no information about the password or the generated key.

5 PROTOTYPE IMPLEMENTATION

Helper. A user can either wear a wristband or hold a smartphone to perform pairing. We implement the prototypes on two helpers: (1) an LG W200 smartwatch, and (2) a Google Nexus 5X smartphone. We develop an application for the smartwatch running Android Wear 2.0, and an application for the smartphone running Android 7 to collect the motion data. Both the smartwatch and smartphone are equipped with a Bosch BMI160 inertial measurement unit containing a triple-axis accelerometer and a triple-axis gyroscope.

IoT device. A variety of IoT devices are used to build the prototypes, as shown in Fig. 5. (1) *Buttons made of two different materials* are used: a plastic keypad labeled as 1, and a rubber one labeled as 2. An Arduino board MKR1000 is adopted to interface with the rubber keypad, and the communication is via the Wi-Fi module of MKR1000. The plastic one has a Bluetooth module to communicate with the helper. (2) *Knobs with two different sizes* are used: a large knob labeled as 3, and a small one labeled as 4. The large knob is a volume controller for desktop; we write an interface function to read its data. For the small one, we use an Arduino board MKR1000 to build its interface. (3) *Touchscreens with two different sizes* are used: Nexus 5X labeled as 5 has a screen size of 5.2", and Unihertz Atom labeled as 6 has a screen size of 2.45". We implement an application to collect the touch trajectory on the screen and record the coordinates of each touch point in the xy-plane of the screen.

6 DATA COLLECTION

We build two datasets: (1) *Dataset I* is used to measure the accuracy of our system, and (2) *Dataset II* is used to evaluate the resistance of our system to mimicry attacks.

We recruit 20 participants: 14 males and 6 females with ages ranging from 18 to 36. We use three devices, including the large knob, the plastic keypad, and the Nexus 5X smartphone, to collect data (the other three devices are used to evaluate the stability of the system, presented in Section 7.4).

6.1 Dataset I for Evaluating Accuracy

To build *Dataset I*, we ask each participant to wear a smartwatch and perform the pairing operations on each of the three devices for 30 times. In addition, to measure the impact of pauses, each participant is asked to perform two types of pairing each time: one without pauses, and another with random pauses (the user can choose to add one or two pauses during the pairing operations).

Positive pairs. When a participant performs the pairing operations on a device, we collect one positive data pair from the smartwatch and device. Thus, for the pairing operations without pauses, our dataset contains 1,800 (= 20 × 30 × 3) positive pairs, each with a label $s = 1$; for the pairing operations with random pauses, we also collect 1,800 (= 20 × 30 × 3) positive pairs, each with a label $s = 1$.

Negative pairs. Assuming two users, μ_1 and μ_2, perform the same pairing operations on two devices, the evidence E_{d_1} from μ_1's IoT device and the evidence E_{h_2} from μ_2's helper constitute a negative pair; similarly, the evidence E_{h_1} from μ_1's helper and the evidence E_{d_2} from μ_2's device constitute another negative pair.

By randomly selecting two users performing the same pairing operations, we generate 1,800 negative pairs (the same amount as the positive pairs) for the pairing operations without pauses, and 1,800 negative pairs for the pairing operations with pauses, each with a label $s = -1$.

6.2 Dataset II for Evaluating Resilience to Mimicry Attacks

To build Dataset II, we have 10 participants act as victims and the other 10 as attackers. We consider the three attack settings of mimicry attacks as discussed in **Threat Model** in Section 2.

For **MA-trained**, we first ask each victim to perform pairing on each type of device for five times, and record a video of each pairing. Each attacker is trained by watching the corresponding

video as many times as needed to train herself. The attacker only needs to learn one victim's actions and launches attacks against that victim. During the training, we provide the attackers with immediate feedback on the differences between their evidence and the victims', so that they can adapt their operations to mimic better.

For each attack setting, each pair of attacker and victim performs the pairing operations with/without pauses on each device for 15 times. Given 4 pieces of evidence: E_{d_V} from $\mathcal{V}$'s device, E_{h_V} from $\mathcal{V}$'s helper, E_{d_A} from $\mathcal{A}$'s device, and E_{h_A} from $\mathcal{A}$'s helper, two kinds of evidence pairs are constructed based on the attackers' goal.

(*G1*) The first pair consists of E_{h_V} and E_{d_A}, implying that $\mathcal{A}$ attempts to have $\mathcal{V}$'s helper accept a pairing with $\mathcal{A}$'s device.

(*G2*) The second pair consists of E_{d_V} and E_{h_A}, implying that $\mathcal{A}$ attempts to fool $\mathcal{V}$'s device into pairing with $\mathcal{A}$'s helper.

For each attack setting, we collect 900 evidence pairs for the pairing operations without pauses, containing 450 (= $10 \times 15 \times 3$) *G1* pairs and 450 *G2* pairs. We collect the same number of pairs for the pairing operations with pauses.

7 EVALUATION

We conduct four in-lab studies to evaluate T2Pair in terms of pairing accuracy, security, stability, and efficiency. The first study (Section 7.1) examines its pairing accuracy. The second (Section 7.2) evaluates the resilience of our system to mimicry attacks. The third (Section 7.3) evaluates the randomness and entropy of evidence. The fourth (Section 7.4) tests the stability of T2Pair under different parameters and experimental settings. The time efficiency is evaluated in Section 7.5. The user study that evaluates the usability of our pairing operations is presented in Appendix A.

7.1 Pairing Accuracy

We use *False Rejection Rate* (FRR) and *False Acceptance Rate* (FAR) to measure the pairing accuracy. 1) FRR is the rate that our system fails to pair the legitimate user's IoT device with the helper. A low FRR is important for usability. 2) FAR is the rate that our system pairs the legitimate user's IoT device (resp. helper) with the attacker's helper (resp. IoT device). So a low FAR is critical for security.

Given a pairing operation, T2Pair accepts the pairing if a shared key can be successfully derived from a pair of evidence that has a Hamming distance smaller than the threshold (see Section 4.1). The threshold (Thr) indicates the allowed evidence difference for T2Pair to accept a pairing. A false rejection occurs if T2Pair obtains $\text{Ham}(E_{d_1}, E_{d_2}) > Thr$ for a legal pairing of d_1 and d_2, and a false acceptance if $\text{Ham}(E_{d_1}, E_{d_3}) < Thr$ for an illegal pairing of d_1 and d_3. The *evidence length* is defined as the number of time intervals it contains. For pairings with pauses, we set the evidence length to 7 for knobs, and 6 for both touchscreens and buttons (see ***Evidence Length*** in Section 7.4). For pairings without pauses, we set the evidence length to 8 for all devices.

We use Dataset I to evaluate the accuracy of T2Pair, and compare the performance between the pairing operations with and without pauses. Figure 6 and Figure 7 show the performance in terms of FAR and FRR by varying the threshold of Hamming distance. We choose the base value as 10ms (***Base Value*** is studied in Section 7.4).

As expected, the larger the threshold, the lower the FRR (better usability), but the higher the FAR (worse security). Figure 6 presents the results for pairings ***without pauses***. By choosing the threshold that yields an FRR 0.10 (we consider an error below 0.10 is reasonably good for usability), we can achieve an FAR 0.02, 0.03, and 0.09 for buttons, knobs, and screens, respectively (see the *vertical dashed lines*). An FRR of 0.10 means that on average 10 out of 100 pairing attempts fail, and thus a user is expected to perform 100/90=1.1 pairing attempts for pairing one device.

Figure 7 shows the performance when ***random pauses*** are introduced during pairing. We find that the FAR can be significantly improved—the FAR grows very slowly as the threshold value increases. The results indicate that random pauses can enhance the discriminability of each pairing. If security is particularly important for certain applications, we can set the FAR as 0.00 and T2Pair achieves (FAR, FRR)=(0.00, 0.03) for buttons, (0.00, 0.09) for knobs, and (0.00, 0.07) for screens (see the *vertical dashed lines*). Thus, security is much improved with usability keeping good. But if vanilla fuzzy commitment (Section 4.1) is used, we can only achieve (FAR, FRR) (0.00, 0.81) for buttons, (0.00, 0.48) for knobs, and (0.00, 0.73) for screens, showing heavily degraded accuracies.

7.2 Resilience to Mimicry Attacks

This section evaluates the resilience of T2Pair (based on the thresholds selected in Section 7.1) to mimicry attacks for two types of pairing operations: one without pauses (*Type-I*) and the other with pauses (*Type-II*). We use FAR to measure the success rate of attacks. We evaluate the resilience using Dataset II (see Section 6.2).

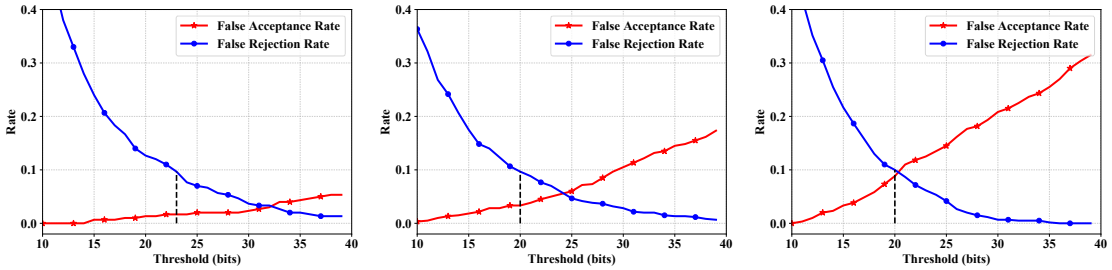
Resilience to MA-obstructed. The attacker ($\mathcal{A}$) stands behind the victim ($\mathcal{V}$) with a distance of 2–3 meters and does not have a clear view of $\mathcal{V}$'s hand movements. As shown in Table 2, for the pairing operations of *Type-I*, T2Pair can successfully identify 96.0%, 95.3% and 90.7% of attacks on buttons, knobs, and screens, respectively. The performance can be greatly improved if the random pauses are considered—specifically, for the *Type-II* operations, T2Pair can successfully defend against all the attacks on knobs, and 99.3% of attacks on screens and buttons.

Resilience to MA-clear. $\mathcal{A}$ stands next to $\mathcal{V}$ and has a clear view of $\mathcal{V}$'s hand movements. As shown in Table 2, for the *Type-I* operations, the attackers' success rate increases, especially for the screen-based device. However, for the *Type-II* operations, the attackers' success rate is still very low. The results demonstrate that the random pauses during each pairing can increase the difficulty for attackers to mimic the victims' hand movements. Thus, the pairing operations with random pauses are more secure.

Resilience to MA-trained. How to train the attacker is described in Section 6.2). Compared to the *Type-II* operations, FARs for the *Type-I* operations increase sharply (up to 27.4%), which reveals a noticeable ***weakness*** of pairing without pauses. The pauses make the intervals more unpredictable and difficult to mimic. To eliminate the weakness, our protocol performs self-checking at Phase 2 in Table 1, which aborts pairing if there are no pauses.

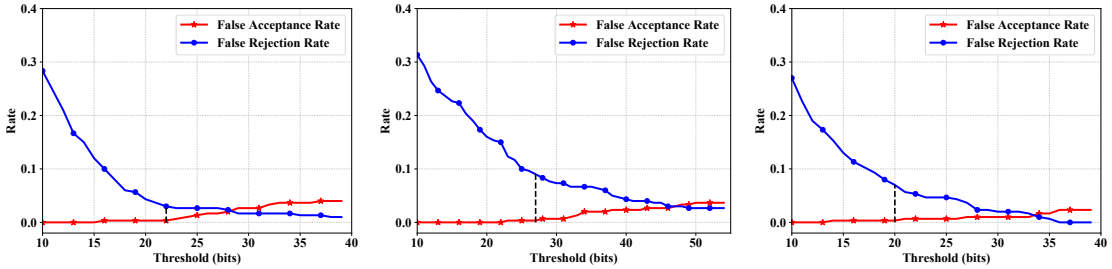
7.3 Randomness and Entropy

Randomness. The randomness level of the time interval between two consecutive events directly affects the entropy of evidence. We notice that it ranges from large values when the user pauses to small ones when she presses/twists/swipes quickly. It is challenging



(a) Button-based device pairing. (b) Knob-based device pairing. (c) Screen-based device pairing.

Figure 6: FARs and FRRs with different threshold values for pairing operations *without* random pauses.



(a) Button-based device pairing. (b) Knob-based device pairing. (c) Screen-based device pairing.

Figure 7: FARs and FRRs with different threshold values for pairing operations *with* random pauses.

Table 2: FARs under mimicry attacks. (Legend: Ai stands for the ith attacker.)

Attacks	Pauses?	Device	$A1$	$A2$	$A3$	$A4$	$A5$	$A6$	$A7$	$A8$	$A9$	$A10$	Avg.
MA-obstructed No clear view Untrained attackers	No	Button	0.07	0.0	0.13	0.0	0.07	0.07	0.0	0.07	0.0	0.0	0.040
		Knob	0.07	0.07	0.0	0.07	0.0	0.13	0.0	0.07	0.07	0.0	0.047
		Screen	0.07	0.13	0.13	0.07	0.13	0.07	0.07	0.13	0.07	0.07	0.093
	Yes	Button	0.0	0.0	0.07	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.007
		Knob	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000
		Screen	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.07	0.0	0.007
MA-clear Clear view Untrained attackers	No	Button	0.07	0.13	0.20	0.07	0.07	0.07	0.07	0.0	0.13	0.13	0.093
		Knob	0.13	0.07	0.13	0.13	0.07	0.20	0.0	0.07	0.13	0.07	0.100
		Screen	0.07	0.13	0.33	0.27	0.20	0.33	0.07	0.13	0.20	0.07	0.180
	Yes	Button	0.0	0.0	0.07	0.0	0.0	0.07	0.0	0.0	0.07	0.0	0.020
		Knob	0.07	0.0	0.07	0.0	0.0	0.13	0.07	0.0	0.0	0.07	0.040
		Screen	0.0	0.07	0.0	0.07	0.07	0.0	0.0	0.0	0.0	0.0	0.020
MA-trained Clear view Trained attackers	No	Button	0.20	0.27	0.27	0.40	0.20	0.20	0.33	0.27	0.33	0.27	0.274
		Knob	0.27	0.20	0.27	0.33	0.20	0.13	0.27	0.20	0.40	0.13	0.240
		Screen	0.20	0.07	0.13	0.27	0.33	0.20	0.13	0.20	0.20	0.07	0.180
	Yes	Button	0.0	0.07	0.0	0.07	0.07	0.07	0.07	0.0	0.07	0.0	0.040
		Knob	0.0	0.0	0.07	0.07	0.0	0.07	0.07	0.0	0.13	0.0	0.040
		Screen	0.0	0.0	0.0	0.0	0.07	0.07	0.0	0.0	0.13	0.0	0.027

to examine their randomness as plenty of samples are required. The prior work [14, 30] also confirms this challenge and directly assumes the human generated events are random.

We instead examine the randomness of the collected intervals over a limited range. Similar to H2H [39], we study whether the six least significant bits of the time intervals are randomly distributed. We verify it by applying NIST statistical test suite [40] on the distribution of our time interval bits. It is a widely used randomness test suite [39, 45, 46]. Our dataset, which is subsampled from Dataset I and II based on users, has a size of 19.2 Kbits consisting of 3200 intervals for each type of pairing operations.

The outputs of the NIST tests are p-values. A p-value represents the probability that the input bit sequence is generated by a random bit generator [40]. If a p-value is less than a chosen critical value (usually 0.01), the null hypothesis for randomness is rejected. Table 3 shows that all the p-values are larger than 0.01 for the three types of devices. The results confirm the randomness.

Entropy analysis. We use I_1 to denote the set of intervals, each of which is generated without pauses, and I_2 to denote the set of intervals, each with a pause. The possible range of I_1 is related to the specifications of a given device (e. g., size, rotation/swiping range) and the device users' behavior habit, while the range of I_2 is mainly determined by device users.

As many human characteristics show normal distributions [7], we assume I_1 and I_2 among all users follow a normal distribution each. The entropy (in bits) of a time interval (with mean denoted

Table 3: NIST statistical test results. A p-value greater than 0.01 indicates a randomness test is passed.

Test	p-value		
	Button	Knob	Screen
Frequency	0.327	0.581	0.300
Block Frequency	0.854	0.118	0.807
Runs	0.190	0.697	0.046
Longest Run	0.249	0.624	0.164
Approximate Entropy	0.051	0.369	0.095
FFT	0.567	0.567	0.829
Cumulative Sums (Fwd)	0.537	0.318	0.505
Cumulative Sums (Rev)	0.476	0.681	0.343
Serial	0.387	0.251	0.360
	0.601	0.074	0.796

Table 4: Average entropy and estimated bit rate.

	Button	Knob	Screen
σ of I_1 (ms)	67	72	53
σ of I_2 (ms)	501	362	424
Entropy (bits)	34.3–38.5	34.3–37.9	32.3–36.6
Bit Rate (bit/sec)	10.3–13.2	10.6–13.6	11.6–14.8

as μ and standard deviation as σ) can be computed as follows [35].

$$E_i = \frac{1}{2}\log_2(2\pi e\sigma^2) \tag{2}$$

Assuming each piece of evidence contains n_1 intervals from I_1 and n_2 intervals from I_2, the evidence entropy can be computed as:

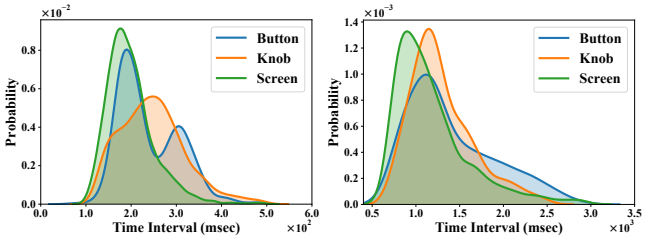
$$l_E = n_1 * E_1 + n_2 * E_2 + \log_2 \binom{n_1+n_2}{n_2} \tag{3}$$

The term $\binom{n_1+n_2}{n_2}$ is introduced to account for the random occurrence positions of the n_2 pauses in the evidence.

The total time of generating a piece of evidence is denoted as T. Then, the bit rate is l_E/T.

Entropy evaluation using a real-world dataset. Figure 8 shows the distributions of the time intervals of I_1 and I_2 among all the users. We test the normality of the distributions with one-sample Kolmogorov-Smirnov testing [24]. For each device, more than 86% of the time intervals follow the normality assumption. Thus, most of the data for each device could be abstracted into a normal distribution. The prior studies [10, 21] of keystrokes and/or screen touches are consistent with our finding.

We then use the pairing operations on buttons as an example to compute the entropy. As summarized in Table 4, the intervals of I_1 mostly fall in [100ms, 500ms] with the standard deviation σ_1 67ms, while those of I_2 in [800ms, 3000ms] with the standard deviation σ_2 501ms. With the base value = 10ms (see Section 7.4), σ_1 and σ_2 become 6.7 and 50.1, respectively. According to our entropy definition in Equation 2, the entropy for one interval in I_1 is around 4.8 bits, and that in I_2 around 7.7 bits. As each piece of evidence consists of 4 (or 5) intervals of I_1 and 2 (or 1) intervals of I_2, the total entropy is around 38.5 (or 34.3) bits. The mean values for the intervals of I_1 and I_2 are 238ms and 1402ms, respectively, so the total time for generating a piece of evidence is 3756ms (or 2592ms). The bit rate is around 10.3 bit/s (or 13.2 bit/s).



(a) Intervals of I_1. (b) Intervals of I_2.

Figure 8: Time interval distributions.

7.4 Study of Parameters and Stability

For the following experiments, we focus on pairing with pauses.

Evidence length. The evidence length is represented as the number of time intervals, which is related to the number of salient points. Longer evidence provides better security, but also requires longer time to finish the pairing, which sacrifices usability. Thus, the evidence length is a trade-off between security and usability.

To study its impact, we set the FRR to a fixed value 0.05, and examine the changes of the FAR as the evidence length varies. Figure 9(a) shows the FARs with different evidence lengths for the three types of devices. As expected, if the evidence length is longer, the FAR is lower—the security is better. For knob-based devices, an evidence length 7 is appropriate as a longer length can only improve the FAR a little bit. For both button-based and screen-based devices, the FAR is below 0.01 if the evidence length is longer than 6. Hence, 6 is an appropriate length for them.

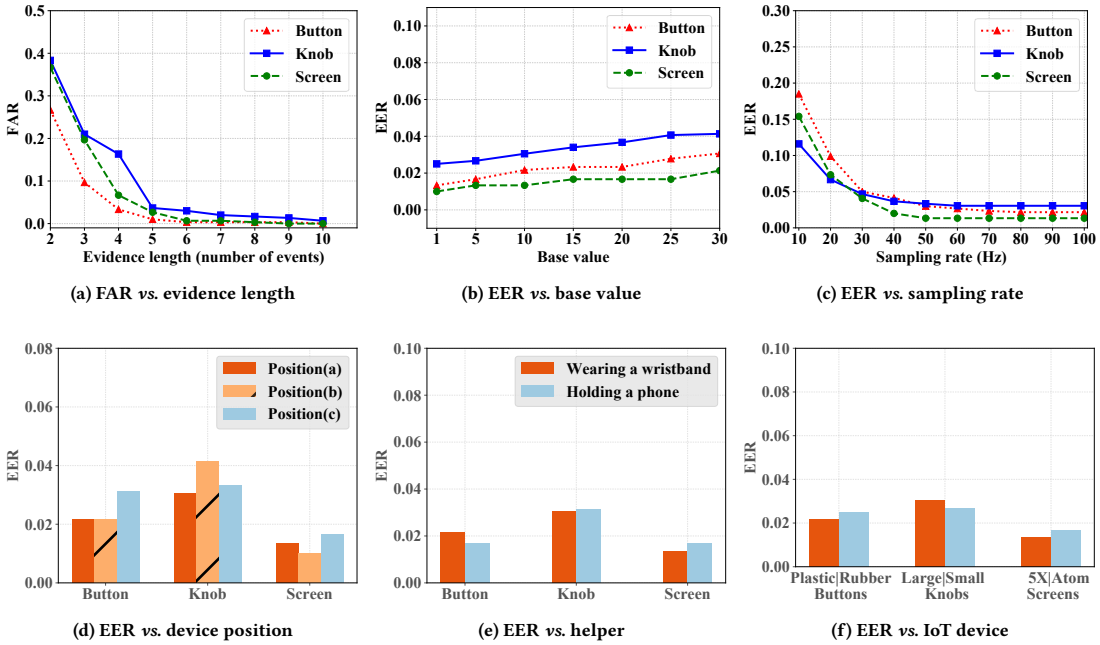
Base value. The base value is used to encode the time intervals. In general, a larger base may generate a less accurate encoding of the time interval because of more coarse approximations, but it can create a shorter encoding of the evidence that is more efficient. Thus, selecting an appropriate base value is a trade-off between accuracy and efficiency. For simplicity, we use EER to study the impact of the base by weighting the FAR and FRR equally.

Figure 9(b) shows the EERs for the three types of pairing by varying the base from 1 to 30ms. We find that the EERs grow slowly as the base increases. Although a base smaller than 10 can slightly improve EERs, it also yields long evidence. Considering both accuracy and efficiency, we choose the base value as 10ms.

Sampling rate. The sensor data from the wristband ("helper") is used to extract salient points and generate the evidence. A low sampling rate of the sensor data may result in inaccuracy in detecting salient points. While a high sampling rate can help capture more subtle motions, it also introduces a higher burden on data collection. An optimal sampling rate needs to be determined by considering both accuracy and efficiency.

Figure 9(c) presents the performance of T2Pair by changing the sampling rate from 10Hz to 100Hz at a step of 10Hz. We observe that button clicking requires a sampling rate higher than 80Hz to achieve the best performance, and knob rotation and screen swiping only require a sampling rate higher than 50Hz. We thus select a sampling rate of 80Hz, 50Hz, and 50Hz for button clicking, knob rotation, and screen swiping, respectively.

IoT device position. IoT devices may be installed/placed at different positions based on the demand (e.g., whether needing to be



(a) FAR *vs.* evidence length (b) EER *vs.* base value (c) EER *vs.* sampling rate

(d) EER *vs.* device position (e) EER *vs.* helper (f) EER *vs.* IoT device

Figure 9: Impacts of different parameters and experimental settings.

connected to a power source) or the user's preference. We examine three common positions that a device may be installed/placed: (a) plugged into a wall outlet; (b) placed on a table; (c) held in a hand.

Figure 9(d) shows the EERs for the three types of pairings in terms of different device positions. For buttons and screens, T2Pair performs slightly better when the devices are in the positions (a) and (b), while for knobs, the positions (a) and (c) achieve slightly better performance. Overall, the results indicate that different device positions have little impact on the pairing performance.

Different kinds of helpers. Besides wristbands, we also test the feasibility of holding a smartphone to perform pairings. We present the EERs in Figure 9(e) for the three types of pairings using the two different helpers: wristband and smartphone. When the smartphone is used, T2Pair achieves an EER of 0.017, 0.031, and 0.017 for buttons, knobs, and screens, respectively. No obvious difference is observed in the pairing performance between the two helpers. We thus conclude that holding a smartphone for pairings is feasible. Nevertheless, we find the usability is not satisfactory when the user holding a smartphone twists a small knob.

Different sizes and materials of IoT devices. We further study whether T2Pair can work well on IoT devices with different sizes and materials. We have two knob-based devices (a large knob and a small knob), two button-based devices (a rubber keypad and a plastic keypad), and two touchscreens (a smartphones Nexus 5X and a Unihertz Atom that have different screen sizes). Dataset I is collected using the large knob, the plastic keypad, and the Nexus 5X with a relatively large screen. We then recruit *another* 5 participants to perform the pairing operations on the other three devices.

Figure 9(f) shows the EERs for the six devices. For any two devices with the same type of UI, we do not observe any significant difference between their performance. Thus, the device size and material have little impact on the pairing performance of T2Pair.

Table 5: Comparison with other works.

Method	(FAR, FRR)	Time(s)
ShaVe/ ShaCK [30]	(0.0, 0.10–0.12)	3
SFIRE [12]	(0.0, -)	6
Tap-to-Pair [49]	(-, 0.117)	15–20
Checksum [1]	(-, 0.10)	5.7
T2Pair	(0.0, 0.03–0.09)	3.2–4.1

7.5 Efficiency

We next evaluate the efficiency of the pairing operations; here we only consider the pairing operations with random pauses. Specifically, we measure *the time used for performing the pairing operations* with an evidence length of 7 for knobs, 6 for screens and 6 for buttons (see ***Evidence Length*** in Section 7.4).

For knobs, screens, and buttons, the mean time for pairing is 2.8s (SD=0.85), 2.3s (SD=0.66), and 3.2s (SD=0.93), respectively. The pairing operations require very short time to finish and are efficient.

We also measure the time used for running fuzzy commitment and PAKE to establish a shard key between two parties. The average execution time on the smartwatch and the Arduino controller is 0.9s (SD=0.37) and 0.7s (SD=0.25), respectively.

Note that the "*big silence*" (≤ 3s) before each pairing is not included here; it is considered in the Usability Study in Appendix A.

7.6 Comparison with Other Approaches

Table 5 shows the comparison of T2Pair with some prior works. Our work achieves better accuracies than these works [1, 30, 49]. Moreover, T2Pair is more efficient than Tap-to-Pair [49], SFIRE [12], and Checksum [1] in terms of the pairing time. E.g., Tap-to-Pair needs at least 15 seconds, while our system only needs up to 4.1

seconds (the maximum time observed for performing pairing operations 3.2s plus the time running our fuzzy commitment 0.9s). Note that each pairing approach requires some initialization phase, and the statistics about the initialization time are not available in many of the works; we thus exclude the initialization time for fair comparison. But even the initialization time ("big silence") is considered, the maximum time of 7.1s (= 4.1 + 3) still shows our pairing is fast. In contrast, Perceptio [14] takes hours or even days for pairing.

8 RELATED WORK

Proximity-based pairing. Some approaches [19, 29, 37] transform the Received Signal Strength (RSS) values into a key, while others exploit Channel State Information (CSI) [25, 46]. As a user moves her smartphone near an IoT device, Move2Auth [48] and SFIRE [12] authenticate the device by checking whether the RSS changes correlate the motion trace of the smartphone. Tap-to-Pair [49] has a user tap, to create RSS changes, the wireless transmitter on an IoT device following the instructions displayed by another device (e.g., a smartphone) to authenticate the IoT device. Many only authenticate the IoT device to the user's mobile device [12, 48, 49], but not the other way around; hence, IoT devices may get paired with the attacker's device, while T2Pair provides mutual authentication.

The changes in ambient context, such as audio [42] and luminosity [32], can also be used to prove co-presence. Perceptio [14] clusters contextual information detected by devices equipped with different sensors to derive a key. Like our work, it also aims at a pairing approach applicable to heterogeneous IoT devices. But it assumes there exists a physical security boundary (e.g., the house wall) and no malicious devices within the boundary. In contrast, T2Pair largely eliminates the threat of co-located malicious devices. Perceptio has the advantage of pairing multiple devices without human intervention, but it may take a very long time for pairing some devices (e.g., a laundry washer that is used once per week and a glass-break sensor that is triggered only once during multiple years), while T2Pair takes seconds for pairing a device. Furthermore, Perceptio has no guarantee whether a device can be paired correctly, especially for devices (e.g., in different floors) that perceive different contextual information.

Both wireless signal changes and the ambient context can be sensed and thus exploited by co-located malicious devices.

Physical contact-based pairing. Some approaches require users to have physical contact with IoT devices for pairing purposes. By shaking [30] or bumping [15] two devices simultaneously, the motion data on both devices becomes correlated and can be used for pairing. Touch-And-Guard [45] has the user wearing a wristband touch the target IoT device, and the wristband's vibration motor creates resonance, which is measured by the accelerometers of both sides and used for pairing. Sethi *et al.* [43] require users to perform synchronized drawings on two touchscreens; the resulting drawings can be used for pairing. By shaking [36] or moving [1] an IoT device according to the public key shown on the display, the key is authenticated. But all these approaches require inertial or touch sensors embedded in the IoT device or a metal pin on its surface [47], which are not available on many IoT devices. While many authentication approaches [27, 28, 33] based on physical contact (including our prior work P2Auth [23]) have been proposed, they all assume a secure communication channel between devices, while the pairing task cannot assume it.

There exist other approaches that do not fall in either of the two categories above. SiB [31] authenticates other device's public key by taking a picture of a 2D bar code encoding the hash of the public key of the other device. VIC [41] improves it by presenting the key with a binary display. Many vendors embed a hard-coded password into the firmware of an IoT device, and print the password on the user manual, so the vendor has to carefully make sure the device and the unique manual are packaged together correctly, which is a burden to vendors [12, 48]. Some vendors simply use an identical password for all devices, which is a critical security flaw. Moreover, given an IoT device (such as a smart blood-pressure meter in Walmart) that needs to get paired with many users' personal mobile devices, a single password for all users is insecure, while T2Pair provides a secure and usable solution.

9 LIMITATIONS

T2Pair largely eliminates the threat of co-located malicious devices, but not completely. If a nearby malicious device (or an attacker) has a camera that points at the user performing authentication, T2Pair is vulnerable to man-in-the-middle attacks assisted by computer vision techniques. Similarly, if the authentication operations generate noises, for example, in the case of pressing a button, a nearby malicious device which has a microphone can also be used to facilitate MITM attacks. How to mitigate side-channel attacks that infer information from noises has been studied [3, 4]. It is worth pointing out that, as analyzed in Section 4.2, offline attacks based on recorded videos or audios do not work.

It is not very usable to hold a large smartphone and twist a small knob. But given a large knob (e.g., a Nest Thermostat), a button or a touchscreen, it is not an issue. As wearable devices, such as smartwatches and fitness trackers, become increasingly popular, the usability of T2Pair can benefit from the trend.

10 CONCLUSION

IoT devices lack traditional user interfaces and are diverse in nature. A secure pairing approach that is applicable to heterogeneous IoT devices is urgently needed. We have presented T2Pair, which is secure and applicable to a large variety of IoT devices. It can be applied to IoT devices without requiring any hardware modifications, sensor calibration, or clock synchronization. We designed very simple physical operations that allow users to finish a pairing process conveniently in a few seconds. We proposed *faithful fuzzy commitment*, which ensures small distances between encodings faithfully indicate small differences between the encoded values, leading to high pairing accuracy. Pauses and self-checking were proposed to enhance the resilience of T2Pair to powerful attacks. A comprehensive evaluation along with a user study was performed, showing its high security, usability, stability, and efficiency.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive suggestions that have helped improve the paper. This project was supported by NSF CNS-1850278, CNS-1815144, and CNS-1856380.

REFERENCES

[1] Imtiaj Ahmed, Yina Ye, Sourav Bhattacharya, N. Asokan, Giulio Jacucci, Petteri Nurmi, and Sasu Tarkoma. 2015. Checksum Gestures: Continuous Gestures As an Out-of-band Channel for Secure Pairing. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*.

[2] Amazon. 2019. AWS IoT Button. https://aws.amazon.com/iotbutton/. Accessed: 2019-06-04.

[3] S Abhishek Anand and Nitesh Saxena. 2016. A sound for a sound: Mitigating acoustic side channel attacks on password keystrokes with active sounds. In *International Conference on Financial Cryptography and Data Security*. Springer, 346–364.

[4] S. A. Anand and N. Saxena. 2019. Noisy Vibrational Pairing of IoT Devices. *IEEE Transactions on Dependable and Secure Computing* 16, 3 (May 2019).

[5] S. M. Bellovin and M. Merritt. 1992. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*.

[6] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *2012 IEEE Symposium on Security and Privacy*.

[7] John H Borghi. 1965. Distribution of Human Reaction Time. *Perceptual and motor skills* 21, 1 (1965), 212–214.

[8] Victor Boyko, Philip MacKenzie, and Sarvar Patel. 2000. Provably secure password-authenticated key exchange using Diffie-Hellman. In *International Conference on the Theory and Applications of Cryptographic Techniques*.

[9] John Brooke. 1996. SUS: A Quick and Dirty Usability Scale. In *Usability Evaluation in Industry*. Taylor & Francis, Chapter 21.

[10] Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. 2018. Observations on Typing from 136 Million Keystrokes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*.

[11] D. Dolev and A. Yao. 1983. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory* 29, 2 (1983).

[12] Nirnimesh Ghose, Loukas Lazos, and Ming Li. 2018. SFIRE: Secret-Free-in-band Trust Establishment for COTS Wireless Devices. In *IEEE INFOCOM - IEEE Conference on Computer Communications*.

[13] Google Nest. 2019. What makes a Nest thermostat a Nest thermostat? https://store.google.com/us/magazine/compare_thermostats?hl=en-US.

[14] J. Han, A. J. Chung, M. K. Sinha, M. Harishankar, S. Pan, H. Y. Noh, P. Zhang, and P. Tague. 2018. Do You Feel What I Hear? Enabling Autonomous IoT Device Pairing Using Different Sensor Types. In *2018 IEEE Symposium on Security and Privacy*.

[15] Ken Hinckley. 2003. Synchronous Gestures for Multiple Persons and Computers. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*.

[16] Honeywell Home. 2019. T9 Wi-Fi Smart Thermostat. https://www.honeywellstore.com/store/products/honeywell-home-t9-wifi-smart-thermostat-rcht9510wfw2001w.htm.

[17] Michael Horowitz. 2019. WiFi Encryption. https://routersecurity.org/wepwpawpa2.php.

[18] IEEE. 2009. IEEE Standard Specification for Password-Based Public-Key Cryptographic Techniques. *IEEE Std 1363.2-2008* (2009), 1–140.

[19] Suman Jana, Sriram Nandha Premnath, Mike Clark, Sneha K. Kasera, Neal Patwari, and Srikanth V. Krishnamurthy. 2009. On the Effectiveness of Secret Key Extraction from Wireless Signal Strength in Real Environments. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*.

[20] Ari Juels and Martin Wattenberg. 1999. A Fuzzy Commitment Scheme. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*.

[21] Sunjun Kim, Byungjoo Lee, and Antti Oulasvirta. 2018. Impact Activation Improves Rapid Button Pressing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*.

[22] James R. Lewis. 1995. IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *Int. J. Hum.-Comput. Interact.* 7, 1 (1995).

[23] Xiaopeng Li, Fengyao Yan, Fei Zuo, Qiang Zeng, and Lannan Luo. 2019. Touch Well Before Use: Intuitive and Secure Authentication for IoT Devices. In *The 25th Annual International Conference on Mobile Computing and Networking*.

[24] Hubert W. Lilliefors. 1967. On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown. *J. Amer. Statist. Assoc.* 62, 318 (1967).

[25] H. Liu, Y. Wang, J. Yang, and Y. Chen. 2013. Fast and Practical Secret Key Extraction by Exploiting Channel Response. In *IEEE INFOCOM - IEEE Conference on Computer Communications*.

[26] Sathiya Kumaran Mani, Ramakrishnan Durairajan, Paul Barford, and Joel Sommers. 2018. A System for Clock Synchronization in an Internet of Things. *arXiv preprint arXiv:1806.02474* (2018).

[27] Shrirang Mare, Andrés Molina Markham, Cory Cornelius, Ronald Peterson, and David Kotz. 2014. ZEBRA: Zero-Effort Bilateral Recurring Authentication. In *2014 IEEE Symposium on Security and Privacy*.

[28] Shrirang Mare, Reza Rawassizadeh, Ronald Peterson, and David Kotz. 2018. SAW: Wristband-based Authentication for Desktop Computers. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 3 (2018), 1–29.

[29] Suhas Mathur, Wade Trappe, Narayan Mandayam, Chunxuan Ye, and Alex Reznik. 2008. Radio-telepathy: Extracting a Secret Key from an Unauthenticated Wireless Channel. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*.

[30] Rene Mayrhofer and Hans Gellersen. 2009. Shake Well Before Use: Intuitive and Secure Pairing of Mobile Devices. *IEEE Transactions on Mobile Computing* 8, 6 (2009).

[31] J. M. McCune, A. Perrig, and M. K. Reiter. 2005. Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication. In *2005 IEEE Symposium on Security and Privacy*.

[32] Markus Miettinen, N. Asokan, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and Majid Sobhani. 2014. Context-Based Zero-Interaction Pairing and Key Evolution for Advanced Personal Devices. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.

[33] Fabian Monrose and Aviel Rubin. 1997. Authentication via Keystroke Dynamics. In *Proceedings of the 4th ACM SIGSAC Conference on Computer and Communications Security*.

[34] Motiv Inc. 2019. Motiv Ring. https://mymotiv.com/.

[35] Kenneth H. Norwich. 1993. *Information, Sensation and Perception*. Academic Press.

[36] Shwetak N. Patel, Jeffrey S. Pierce, and Gregory D. Abowd. 2004. A Gesture-based Authentication Scheme for Untrusted Public Terminals. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*.

[37] N. Patwari, J. Croft, S. Jana, and S. K. Kasera. 2010. High-Rate Uncorrelated Bit Extraction for Shared Secret Key Generation from Channel Measurements. *IEEE Transactions on Mobile Computing* 9, 1 (2010).

[38] I. Reed and G. Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *J. Soc. Indust. Appl. Math.* 8, 2 (1960).

[39] Masoud Rostami, Ari Juels, and Farinaz Koushanfar. 2013. Heart-to-Heart (H2H): Authentication for Implanted Medical Devices. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*.

[40] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. 2001. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Technical Report.

[41] Nitesh Saxena, Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. 2006. Secure Device Pairing Based on a Visual Channel. In *2006 IEEE Symposium on Security and Privacy*.

[42] D. Schürmann and S. Sigg. 2013. Secure Communication Based on Ambient Audio. *IEEE Transactions on Mobile Computing* 12, 2 (Feb 2013).

[43] M. Sethi, M. Antikainen, and T. Aura. 2014. Commitment-Based Device Pairing with Synchronized Drawing. In *2014 IEEE International Conference on Pervasive Computing and Communications*.

[44] Statista. 2016. Internet of Things (IoT) Connected Devices Installed Base Worldwide from 2015 to 2025 (in Billions). https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[45] Wei Wang, Lin Yang, and Qian Zhang. 2018. Resonance-Based Secure Pairing for Wearables. *IEEE Transactions on Mobile Computing* 17, 11 (2018).

[46] Wei Xi, Chen Qian, Jinsong Han, Kun Zhao, Sheng Zhong, Xiang-Yang Li, and Jizhong Zhao. 2016. Instant and Robust Authentication and Key Agreement Among Mobile Devices. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.

[47] Zhenyu Yan, Qun Song, Rui Tan, Yang Li, and Adams Wai Kin Kong. 2019. Towards Touch-to-Access Device Authentication Using Induced Body Electric Potentials. In *The 25th Annual International Conference on Mobile Computing and Networking*.

[48] Jiansong Zhang, Zeyu Wang, Zhice Yang, and Qian Zhang. 2017. Proximity Based IoT Device Authentication. In *IEEE INFOCOM - IEEE Conference on Computer Communications*.

[49] Tengxiang Zhang, Xin Yi, Ruolin Wang, Yuntao Wang, Chun Yu, Yiqin Lu, and Yuanchun Shi. 2018. Tap-to-Pair: Associating Wireless Devices with Synchronous Tapping. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4 (Dec. 2018).

A USABILITY STUDY

This study investigates the usability of T2Pair and compares it with the password-based pairing mechanism as the baseline, which is currently one of the most widely used pairing mechanisms.

A.1 Recruitment and Design

We recruit 20 participants (9 females) by posting the recruitment flyers on the university campus. The study is advertised as "*evaluating*

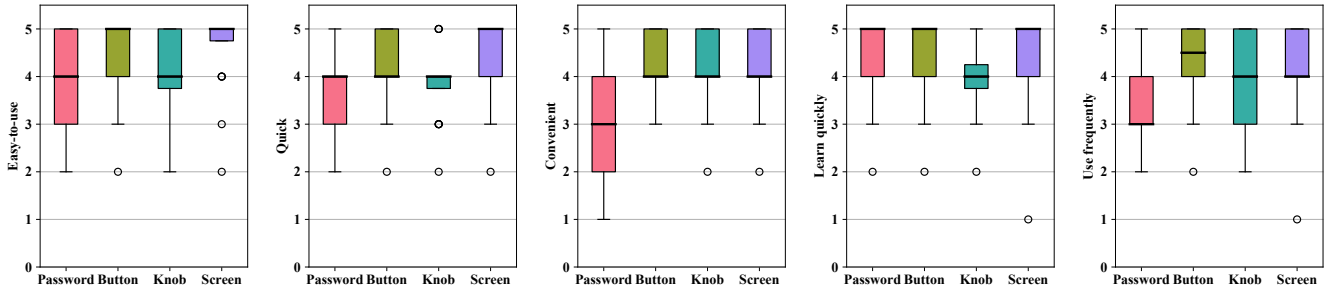


Figure 10: Usability surveyed using questions adapted from SUS [9].

the usability of different pairing mechanisms for IoT devices". Most participants are not from the CS department and none of them have computer security background. Specifically, 3 participants are local residents near the campus, 15 are students, and 2 are staff/faculty members. Their ages range from 20 to 70.

Considering the social desirability bias, we do ***not*** make the participants aware that T2Pair is a mechanism that we are working on. Instead, we inform them that *we are investigating the usability of different pairing methods*. For the password-based mechanism, as a Wi-Fi password usually requires a minimum of 8 alphanumeric characters [17], we randomly create a 8-char alphanumeric password, and show the password to the participants before pairing.

The experiment is conducted in a lab environment. We first ask each participant to sign a consent form and fill out an initial survey to collect the demographic information. We then introduce the two pairing mechanisms (i.e., T2Pair and the password-based mechanism) to them in a random order to avoid the learning bias. Specifically, for T2Pair, we introduce the three pairing operations with respect to the three types of IoT devices, while a smartphone is used for inputting a password. Next, each participant is instructed to perform two pairing attempts on each of the three IoT devices as well as the smartphone to get familiar with T2Pair and the password-based mechanism. These attempts are excluded from further analysis. After that, each one performs another three pairing attempts on each IoT device and the smartphone, respectively.

Finally, the participants are asked to rate the following five statements to examine user preferences and usability (the rating score is from 1 to 5, where 1 stands for strongly disagree, and 5 for strongly agree): (a) I thought the pairing method was easy to use; (b) I am satisfied with the amount of time it took to complete the pairing; (c) I thought the pairing method was convenient; (d) I would imagine that most people would learn to perform the pairing very quickly; and (e) I would be happy to use this pairing method frequently. The questions are inspired from the metrics used in previous studies [6, 22] and adapted based on SUS [9]. We do not use all the 10 questions in SUS as some do not fit our scenario. At the end, we conduct a brief interview with the participants to gain insights into what they like and dislike about each mechanism.

A.2 Usability Results

Perceived usability. We investigate the usability from five aspects based on the five statements above: easy to use, quick, convenient, learn quickly, and use frequently. Figure 10 shows the results. The overall scores for button clicking, knob twisting, and screen swiping are (21.70 ± 3.29), (19.80 ± 3.76), and (21.65 ± 3.54), respectively. For password-based pairing, the overall score is (18.45 ± 3.37).

To analyze the statistical significance of these results, we first hypothesize that T2Pair shows similar usability as password. We use the one-way ANOVA test to examine the hypothesize. The result of the one-way ANOVA test shows that (i) there are significant differences between button clicking and inputting an 8-char password ($F(1, 19) = 9.057$, $p = 0.005 < 0.05$) and between screen swiping and inputting password ($F(1, 19) = 8.149$, $p = 0.007 < 0.05$), and thus our hypothesis can be rejected; and (ii) there is no significant difference between knob twisting and inputting password ($F(1, 19) = 1.358$, $p = 0.251$). We thus conclude that users perceive better usability with button clicking and screen swiping than using an 8-char password, and similar usability for knob twisting and using an 8-char password.

Pairing time. We do not consider the time used for running the pairing protocol as we only focus on the time used by the user. For T2Pair, the mean time for performing a pairing on the button, knob, and screen is 5.2 ± 0.57s, 6.0 ± 0.83s, and 5.6 ± 0.73s, respectively. With respect to password, the mean time for reading and inputting an 8-char alphanumeric password is 9.5 ± 0.78s. Thus, our mechanism is more efficient.

Failure rate. For T2Pair (based on the thresholds selected in Section 7.1), each participant performs three attempts on each device—there are 60 pairings for each device. We see 3 failures out of 60 attempts for button, 2 failures for knob, and 4 failures for touchscreen. Then, each one reads and inputs a given password three times on the smartphone—there are 60 pairings and 5 failures. Thus, T2Pair has a slightly lower failure rate than the password-based mechanism.

Feedback. We also collect their comments about the advantages and disadvantages of our three pairing operations from different perspectives. We here report some representative comments: comments from seven subjects indicate that they like the button clicking pairing operations as they require little effort and/or burden; some also mention that twisting the knob for too many rounds can lead to fatigue, but 7 twistings used by T2Pair are acceptable.

B SENSING PAIRING OPERATIONS (BUTTONS AND SCREENS)

The correlation of IMU data and pairing operations (see Section 3.2).

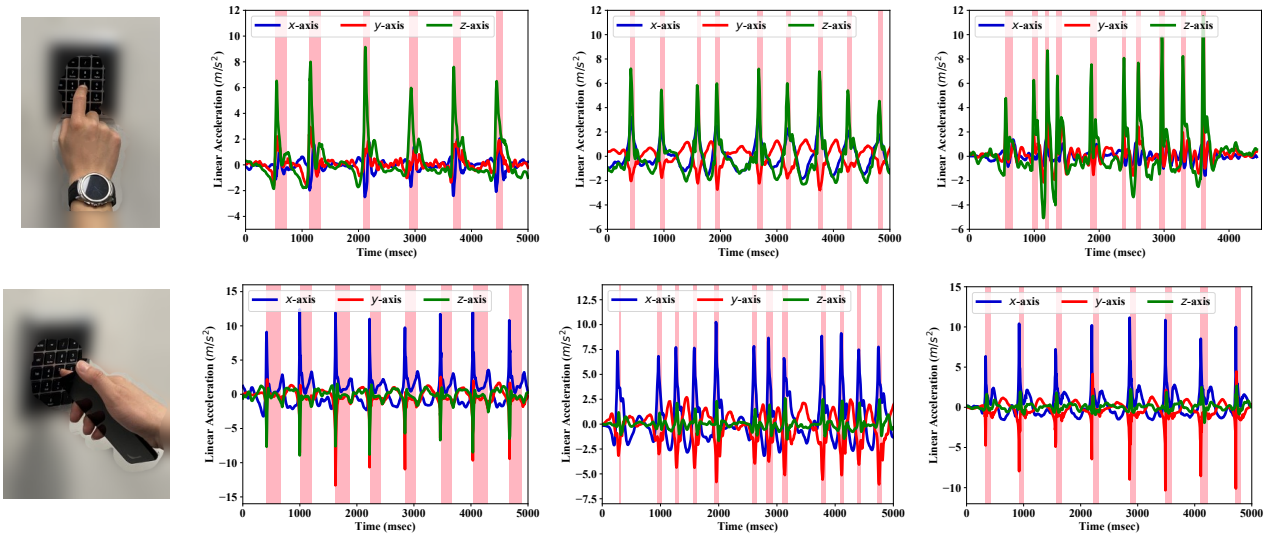


Figure 11: The acceleration data captured when users press buttons, and their correlation with button-pressing operations.

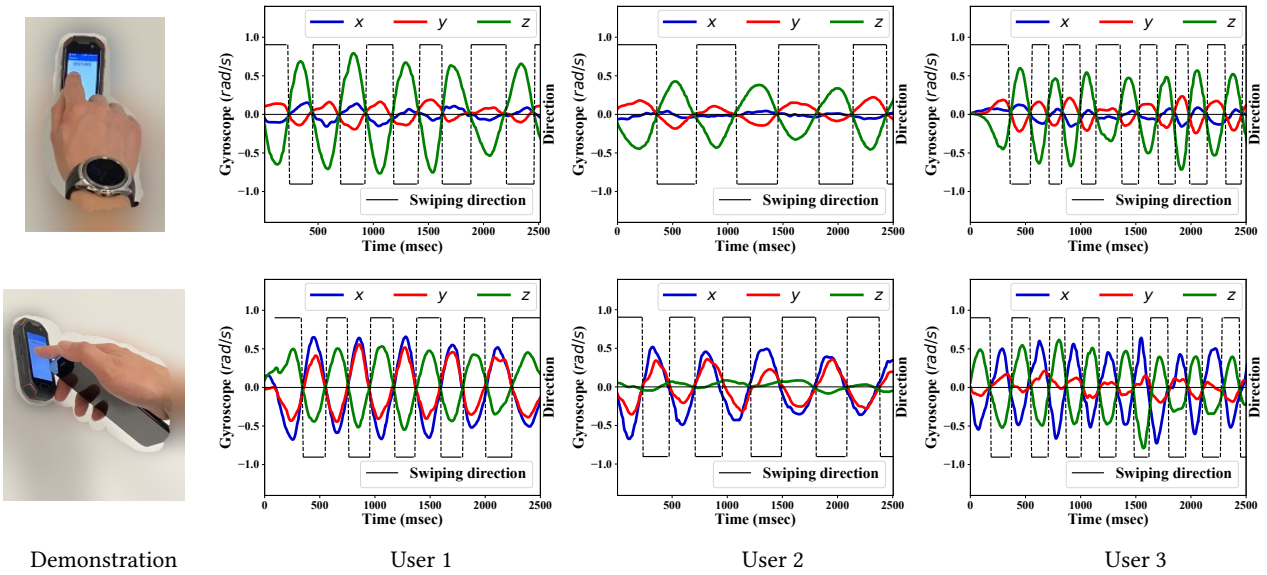


Figure 12: The gyroscope data captured when users swipe touchscreens, and their correlation with swiping operations.