TwistIt: Wristband Authentication with Apple Watch

Pedro Macedo
Faculty of Computer and
Information Science
Ljubljana, Slovenia
Email: pf06708@student.uni-lj.si

***Abstract*—Smart devices allow us to have more data about a person. For example, the person's location in a specific time, perform a person's ECG and a lot other features. Nonetheless, due to this rapidly growing of a person's information, the data security is more compromised, and needs to be saved using different types of authentication. That's why the society invented the current authentication methods. However, this authentication methods have become more tiresome with the increasing of person's personal smart devices. In consequence, people need to pass, for each of their smart devices, the respective security method. They lack the method of logging-in one time on a device, and log on the other person's personal devices without the need to pass throught the authentication process in each smart device. This paper will present an improved authentication method, which will take a smartwatch to work as a token to unlock the other personal devices.**

I. Introduction

Today, we are surrounded by an ever-growing number of smart devices, ranging from smartphones and tablets to wearable and IoT gadgets. This devices have become a part of our daily life, collecting and generating an unprecedented volume of data that must be securely managed and accessed. As the number of interconnected devices increase, so does the complexity of authenticating and securing access to them. Current authentication methods, such as PINs, biometrics (e.g., fingerprint and facial recognition) are design to handle only individual devices. This methods becomes exhausting when user want to authenticate in several smart devices simultaneously. As so, this approach becomes inefficient and introduces a poor user experience. Moreover, many existing authentication methods, such as password authentication possesses high security risks, because passwords can be stolen. Furthermore, biometrics, while convenient to use, have crisis when dealing in different types of environments, such as, poor lightning for facial recognition or dirty hands for fingerprint recognition. Hence, this project aims to address this limitations of current authentication methods by proposing a more efficient solution: a wristband-based authentication system using an *Apple Watch*.

II. Related work

Our approach was deliberated in some studies, such as *WearLock* [2] and *"Flick me once and I know it's you!"* authentication approach [3]. *WearLock* uses "acoustic tones as tokens to automate the unlocking securely" [2], while the other authentication method analysis the behaviour of the user's wrist, and tries to collect the users flicking wrist movement in two different scenarios, sitting and standing [3].

III. System design

In this section, we present the system design implemented in order to build this authentication system. Furthermore, to implement the wristband authentication method, motion data was collected from the accelerometer (acceleration vectors) and gyroscope (quaternions vectors) sensors available on both the Apple Watch and the iPhone. These sensors measure acceleration and angular velocity across the x, y and z axes, providing a detailed representation of the device's movement and orientation. For each sample collected from the sensors, we included the timestamp data using the `timeIntervalSince1970` function in Swift, which provides a precise epoch time for synchronization.

A. Overall system design

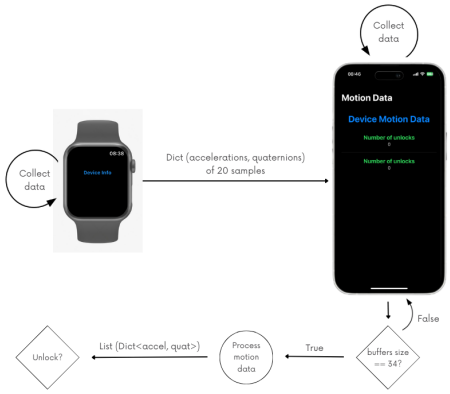

Figure 1: System design

In Fig. 1, we describe how our system is designed. The design consists of the following components and processes:

1) **Data collection**: The Apple Watch collects motion data using its accelerometer and gyroscope sensors. Each data collected is packaged into dictionaries containing 20 samples and sent to the iPhone via the `WatchConnectivity` framework.
2) **Data buffering on the iPhone**: The iPhone receives the data and stores them in a buffer. The buffer size is

monitored to ensure that it reaches the required capacity of 34 samples before proceeding to the next phase (motion data processing).

3) **Motion data processing**: When the buffer has at least 34 samples, the system processes the collected data to analyse synchronized patterns.
4) **Authentication details**: The processed motion data are used to determine whether the movements of the Apple Watch and the iPhone are sufficiently synchronized.

Note: This system operates at a motion frequency rate of 10Hz (which means 10 samples per second), and the unlock movement consists of 15 samples (1.5 seconds). To ensure accurate synchronization and analysis, a window overlap is used during motion data processing (overlap of 14 samples from the next window). This overlap reduces potential data loss between successive windows, as you can see in Fig. 2.

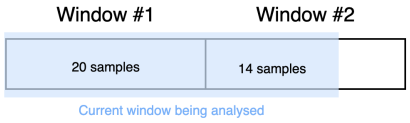

Figure 2: Overlapping window to be processed

B. Data processing

Once the motion data has been collected and synchronized, this system processes these data to analyse patterns and determine whether the Apple Watch and the iPhone movements are sufficiently synchronized to trigger an authentication. First, it checks if the acceleration values are synchronized, and then it checks if the quaternions data are synchronized. In this step, the following process is conducted to analyse the collected acceleration data:

1) **Data Pairing**: Using synchronized timestamps, accelerometer data from the Apple Watch and iPhone are paired to form aligned buffers. Each pair corresponds to a motion event in a specific timestamp of both devices.
2) **Stationary devices**: Using the acceleration data, the system identifies whether at least one of the devices is stationary by comparing the average magnitude of all the accelerations with a predefined threshold (`accelerationStationaryThreshold`).
3) **Sliding window cycle**: It is used a sliding window of size 15 samples (number samples of the synchronized movement), through the entire buffer of 34 samples, with a step of 1 sample.
4) **Sliding window analysis**: For each sliding window, it goes through each pair of accelerations and calculates the absolute difference of both the Apple Watch and the iPhone accelerations magnitude. If at any iteration, this difference is bigger than a predefined threshold (`accelerationDiffThreshold`), the system returns false (which means that there is no synchronized movement in the analysed window).

If there is no detected synchronized movement in terms of acceleration, the system returns false (which means that there is no unlock synchronized movement in the analysed buffer). Otherwise, it continues to the next phase (analyse data quaternions). The following process is conducted:

1) **Data Pairing**: Similar to the pairing process for acceleration data, the same method is applied to the collected quaternion data, ensuring that corresponding samples from the Apple Watch and iPhone are accurately aligned based on their timestamp.
2) **Calculate device rotation**: Each orientation data of a device is actually a 3D rotation from the unknown device's reference frame. Thus, to facilitate the comparison between the quaternion series from the Apple Watch and the iPhone, we compute the change in orientation over time for each device, as it can be seen in Fig. 3. The orientation change can be calculated using Equation 1:

$$\begin{aligned} \Delta q_A &= q_A^t \times (q_A^{t+\alpha})^{-1} \\ \Delta q_B &= q_B^t \times (q_B^t)^{-1}, \end{aligned} \tag{1}$$

where Δq_A is the orientation variation of device A, Δq_B is the orientation variation of device B and α is the time interval between two consecutive quaternion samples.

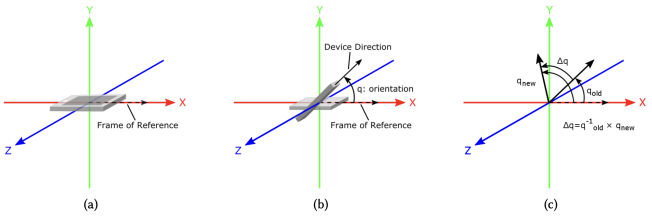

Figure 3: Device's rotation change between two quaternion samples

3) **Estimate basis transformation quaternion**: In order to align both devices' axes, we compute the average of the transformation quaternions derived from each pair of orientation deltas between the devices. Specifically for each time t, we calculate the transformation quaternion as $r_t = \Delta q_A^t \cdot (\Delta q_B^t)^{-1}$. In Equation 2, the final basis transformation is obtained by normalizing the average of these transformation quaternions across all time steps:

$$basisR = \frac{\frac{1}{N}\sum_{t=1}^{N} r_t}{\left\| \frac{1}{N}\sum_{t=1}^{N} r_t \right\|}, \tag{2}$$

where N is the total number of quaternions.
4) **Quaternions alignment**: After calculating the overall basis transformation quaternion, the calculated basis transformation $basisR$ is used to adjust the iPhone quaternions to align them with the watch quaternions, using Equation 3. This alignment ensures that the two sets of quaternion data are expressed in the same coordinate system.

$$\Delta q_A^t = r \Delta q_B^t r^{-1} \quad \forall_{t \in basis} \tag{3}$$

5) **Turning points detection**: Turning points are identified for both the watch quaternions and the aligned iPhone quaternions. These turning points represent significant changes in orientation. A sample is classified as a turning point if its magnitude is either smaller than both of its neighbouring samples (indicating a local minimum) or larger than both of its neighbouring samples (indicating a local maximum).
6) **Turning points overlap ratio**: Finally, we calculate the overlap ratio between the Apple Watch and iPhone turning points, by using Equation 4:

$$\text{Overlap Ratio} = \frac{\text{Nr. of Common Turning Points}}{\text{Total Turning Points in Watch}} \tag{4}$$

7) **Unlock decision**: If the overlap ratio is bigger than a predefined threshold (`turningPointsOverlapThreshold`), the system concludes that there is sufficient synchronization and returns true (which means it triggers an unlock).

IV. Implementation

In this section, we will mention the application developments (tools used), the algorithms implemented for detecting synchronized motion patterns, and the challenges faced during the implementation of this system.

A. Application development

This application was developed using *Swift*, Apple's primary programming language for *iOS* and *watchOS* applications. Additionally, we used `CoreMotion`, `WatchConnectivity` and `SwiftUI` for accessing motion data from the accelerometer and gyroscope sensors on both devices, to enable communication and data transmission between the two devices, and to create an application interface on both devices, respectively.

B. Motion analysis procedure

The motion analysis process is the core process of this application, since it is responsible for synchronizing samples and for detecting synchronized motion patterns between the two devices. In this subsection, we also present the algorithms used in our system to detect if there is a synchronized movement between the devices.

Algorithm 1 Acceleration Analysis Procedure

Require: Time series of motion data (accelerations) from Devices A and B
Ensure: Synchronization State
1: Synchronize accelerations data from Devices A and B using timestamps
2: Check if at least one device is stationary
3: **if** at least one device is stationary **then**
4: **return false** (stationary movement)
5: **end if**
6: **for** each sliding window of size N (with overlap) in synchronized data **do**
7: Extract acceleration data for the current window
8: Compute magnitude differences for each pair:

$$\Delta a_i = |a_i^A - a_i^B|, \quad \forall_{i \in W}$$

9: **if** $\Delta a_i \leq$ accelerationDiffThreshold, $\forall_{i \in W}$ **then**
10: **return true** (motions are synchronized)
11: **end if**
12: **end for**
13: **return false** (no synchronized motion detected)

In Algorithm 1, we can see how the system processes the Apple Watch and the iPhone acceleration values, and how it detects if they have the same acceleration magnitude. If the system detects that the devices have the same acceleration, then it continues the authentication process by dealing with the Apple Watch and iPhone quaternions.

Algorithm 2 Quaternions Analysis Procedure

Require: Time series of motion data (quaternions) from Devices A and B
Ensure: Synchronization State
1: Synchronize quaternions data from Devices A and B using timestamps
2: Calculate delta quaternions for Devices A and B (Δq_A^t and Δq_B^t)
3: Estimate basis transformation matrix r_t between Device A and Device B
4: Align device A quaternions with device B quaternions
5: Detect turning points for both devices
6: Calculate overlap ratio between turning points
7: **if** overlap ratio $\geq$ turningPointsOverlapThreshold **then**
8: **return true** (motion patterns are synchronized)
9: **end if**
10: **return false** (motion patterns are not synchronized)

In Algorithm 2, we can see how the system processes the Apple Watch and iPhone quaternion values, and how it detects if both devices are synchronized. If the system detects that there is at least one sequence of samples (consisting on the number of required samples to form an unlock movement) that are synchronized, the system returns true. Otherwise, discards the buffer.

Algorithm 3 Overall Motion Analysis Procedure

Require: Time series of motion data: $\mathbf{a}_A, \mathbf{q}_A$ (accelerations and quaternions from Device A), $\mathbf{a}_B, \mathbf{q}_B$ (accelerations and quaternions from Device B)
Ensure: Synchronization State
1: Apply **Algorithm 1** with $\mathbf{a}_A$ and $\mathbf{a}_B$ as inputs
2: **if** accelerations are not synchronized **then**
3: **return false** ▷ Acceleration patterns do not match
4: **end if**
5: Apply **Algorithm 2** with $\mathbf{q}_A$ and $\mathbf{q}_B$ as inputs
6: **if** quaternions are synchronized **then**
7: Increment $phoneUnlocks$ variable: $phoneUnlocks \leftarrow phoneUnlocks + 1$
8: **return true** ▷ Both motion patterns are synchronized
9: **end if**
10: **return false** ▷ Motion patterns are not synchronized

In Algorithm 3, we combine both Algorithm 1 and Algorithm 2 to detect if there is a synchronized movement in the received buffer. If there is, increments the overall number of unlocks.

C. Challenges in motion analysis

During the process of collecting accelerations and quaternions of both devices, we encountered some challenges. Such as **clock drifting** (differences in internal clocks between both devices caused minor mismatches between the samples timestamps) and **alignment of the devices axis** (the devices use independent coordinate systems, which makes direct quaternion comparison inaccurate).

V. Implementation details and Experiment results

In this section, we describe the implementation details of our system and the scenarios performed in order to evaluate our method. In the first scenario, we evaluated *TwistIt* with the genuine user trying to unlock the phone, using 3 different unlock movements (one movement per experiment). In the second scenario, we evaluate how robust is our system to the use of the devices in different genuine user's hands, and also an actor/impostor trying to mimic the genuine user's movement.

In order to evaluate our system, we compared the performance using the following evaluation metrics:

- *True Positive Rate (TPR)*: Measures the success rate of considering an unlock of a positive synchronous movement.
- *False Positive Rate (FPR)*: Measures the error rate of accepting an asynchronous movement.
- *False Negative Rate (FNR)*: Measures the error rate of rejecting a synchronous movement.
- *True Negative Rate (TNR)*: Measures the success rate of rejecting an asynchronous movement.
- *Equal Error Rate (EER)*: Measures the accuracy of the system, which is defined as $EER = \frac{FPR+FNR}{2}$.

A. Hardware and Implementation Details

We implemented the prototype of *TwistIt* using *Swift* programming language, in which it collects the motion data and processes the same data in order to identify a synchronized movement. Concerning the hardware, we used an iPhone 14 (iOS 10.0.1) and an Apple Watch Series 9 GPS (watchOS 11.0.1). Both devices contain an accelerometer and a gyroscope, which allowed us to collect the acceleration values and orientation values of the devices. Data was collected at 10Hz using *Apple's Core Motion Framework*. We also used *Apple's WatchConnectivity Framework* in order to be able to stream the Apple Watch motion data to the iPhone.

B. Experiment 1: Determine the number of unlocks for the genuine user scenario

The goal of this experiment is to measure the accuracy of our system when a genuine person (same person holding the iPhone and the *Apple Watch*) moves both devices with the same motion. We decided to test this experiment using three different movements: (i) the user holds the iPhone and the *Apple watch* on the same hand with the wrist turning down while moving the hand in a half circular movement repeatedly for about 2 minutes, (ii) user holds the iPhone and *Apple watch* on the same hand with the wrist up and while moving the hand to the front and back repeatedly for about 2 minutes, and (iii) the user holds both devices similar to the second movement, but now moves both devices up and down repeatedly for about 2 minutes.

We evaluated the system for this experience using the first movement using several different thresholds for the acceleration difference between the two devices.

Table I: FRR and TAR for different thresholds in Movement 1.

Threshold	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
FRR	0.98	0.43	0.23	0.20	0.15	0.13	0.08
TAR	0.02	0.57	0.77	0.80	0.85	0.87	0.92

As we can see in Table I, as the threshold increases, *FRR* decreases, while *TAR* increases, reflecting an improvement in usability but a potential decrease in security. This means that at low thresholds, the system indicates overly strict criteria that can exclude genuine unlock cases. At high thresholds, the system shows a more balanced performance. However, high thresholds can increase the likelihood that false impostor cases will be accepted, which could compromise the system.

In order to obtain performance metrics about our system, we used a similar approach as the first movement in the second movement.

Table II: FRR and TAR for different thresholds in Movement 2.

Threshold	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
FRR	0.73	0.67	0.60	0.52	0.42	0.36	0.28
TAR	0.27	0.33	0.40	0.48	0.48	0.64	0.72

As we can see in Table II, similar to the first scenario, as the threshold increases, *FRR* decreases, while *TAR* increases. It is possible to see that there is a trade-off between rejecting valid users and accepting them. Furthermore, at the highest threshold (0.50%) the system achieved its best *TAR* of 72% and its lowest *FRR* of 28%. This shows that the system performs better at higher thresholds in terms of user acceptance.

Table III: FRR and TAR for different thresholds in Movement 3.

Threshold	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
FRR	0.70	0.67	0.58	0.48	0.40	0.33	0.25
TAR	0.30	0.33	0.42	0.52	0.60	0.67	0.75

As we can see in Table III, similar to the second scenario, as the threshold increases, *FRR* decreases, while *TAR* increases. Lower thresholds are more suitable for application that require high security (as our system), while higher thresholds are better for convenience use cases.

The overall results for this first experiment are shown in Table IV.

Table IV: Overall *FRR* and *TAR* for different thresholds in Genuine Score.

Threshold	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
FRR	0.80	0.59	0.47	0.40	0.32	0.27	0.20
TAR	0.20	0.41	0.53	0.60	0.68	0.73	0.80

In Figure 4, we show the curve that combines the *FRR* and the *TAR* scores.

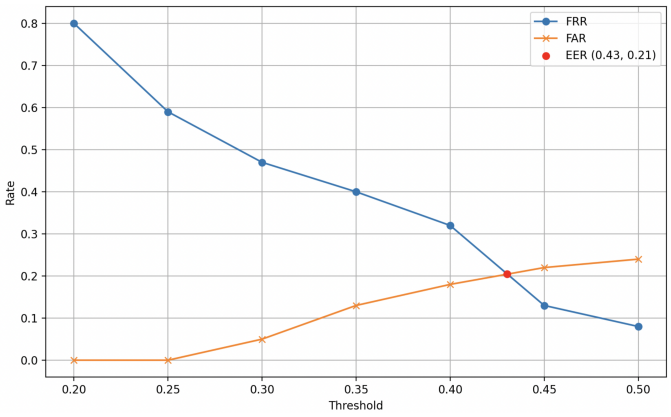

Figure 4: *FRR* and *FAR* vs Threshold (with exact *EER* Marked)

As we can see in Figure 4, the graph shows average per threshold of the accuracy values obtained from the three different scenarios (or movements). It shows the relationship between *FRR* and *FAR* across various thresholds, with the *Equal Error Rate* (*EER*) marked at threshold of 0.43 and an error rate of 21%. The *EER* represents the threshold where both error rates are balanced, offering a trade-off between security and usability. The performance of the system demonstrates a reasonable balance between this two evaluation metrics.

C. Experiment 2: Determine the number of unlocks for different hands and impostor user scenarios

The goal of this experiment is to measure the accuracy of our system when the system considers the movement as not being an unlock when the devices' movements don't match. We decided to test this experiment using two different movements: (i) the user holds the iPhone and the *Apple watch* on different hands with the devices pointing to each other while moving the hands in a half circular movements perpendicular to each other repeatedly for about 2 minutes, and (ii) an impostor tries to mimic the genuine user behaviour while holding the iPhone and the genuine user holds the *Apple watch*.

We evaluated the system for this experience using the first movement using several different thresholds for the acceleration difference between the two devices.

Table V: FAR for different thresholds in Different Hands.

Threshold	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
FAR	0	0	0.08	0.20	0.27	0.34	0.37

As we can see in Table V, in this scenario, lower thresholds maintain a very low *FAR*, ensuring high security. However, as the threshold increases, *FAR* rises significantly, reaching 37% at the threshold 0.50, which compromises the system's security.

In order to obtain performance metrics about our system, we used a similar approach as the first movement in the second movement.

Table VI: FAR for different thresholds in Actor/Impostor scenario.

Threshold	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
FAR	0	0	0.02	0.05	0.08	0.10	0.11

As we can see in Table VI, the *FAR* remains 0 for thresholds 0.20 and 0.25, which indicates that the system is highly secure at low thresholds with no false acceptances. In addition, as the threshold increases, *FAR* begins to also increase.

The overall results for this first experiment are shown in Table VII.

Table VII: Overall *FRR* and *TAR* for different thresholds in Impostor Score.

Threshold	**0.20**	**0.25**	**0.30**	**0.35**	**0.40**	**0.45**	**0.50**
FAR	0.00	0.00	0.05	0.13	0.18	0.22	0.24

The overall results shown in Table VII show that the gradual increase in *FAR* highlights the trade-off between maintaining security and improving usability. For our type of system, since security is a critical requirement, thresholds bellow 0.30 would be better to minimize false acceptances.

VI. Conclusions

In conclusion, the *TwistIt* wristband-based authentication system seamlessly integrates the Apple Watch as a secure

token to unlock an iPhone through synchronized motion detection. By analysing accelerometer and gyroscope data, we ensured precise identification of synchronized movements, providing users with a secure authentication experience. Challenges such as clock drifting and axis alignment were addressed, enabling the system to maintain high reliability across various scenarios. The source code of this project is available in the *Github*, by clicking here.

REFERENCES

[1] Apple, *Take an ECG with the ECG app on Apple Watch*, Apple company, September 03, 2024. Available at: https://support.apple.com/en-us/120278

[2] S. Yi, Z. Qin, N. Carter and Q. Li, "WearLock: Unlocking Your Phone via Acoustics Using Smartwatch," 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 2017, pp. 469-479, doi: 10.1109/ICDCS.2017.183. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7979992

[3] Y. Li, F. Ferreira and M. Xie, "Flick me once and I know it's you! Flicking-based Implicit Authentication for Smartwatch," 2023 International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 2023, pp. 318-323, doi: 10.1109/ICNC57223.2023.10074521. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10074521

[4] Apple Company, CMMotionManager, Available at: https://developer.apple.com/documentation/coremotion/cmmotionmanager

[5] UbiComp'13, BlueEye – A System for Proximity Detection Using Bluetooth on Mobile Phones, Available at: https://dl.acm.org/doi/pdf/10.1145/2494091.2499771

LIST OF FIGURES