Gesture Authentication in Trusted Execution Environment

Richard Hanulewicz, Derek Palmerton, and Chang Min Park

I. Introduction

As the mobile platform grows ever-larger, security problems have become extremely important. Lots of prior work has focused on various layers—application, platform, and kernel. Despite this effort, vulnerabilities remain at every layer. This is due to the necessity of exposing data to the operating system. The larger consumer operating systems grow in scope and scale, the more difficult it is to verify them for security.

In this project, we use ARM TrustZone technology that supports hardware isolation. The *Trusted Execution Environment (TEE)* is the small, easily verifiable, secure operating system running the the Secure World. As long as TEE is strongly secured, vulnerabilities and exploits in the Normal World cannot compromise applications in the Secure World. Using this technology, our system supports gesture authentication in TEE. To do that, we migrate IMU sensor drivers and write new syscalls in TEE. We also implement the*Dynamic Time Warping* algorithm for more accurate gesture comparison. Following sections cover all the details and challenges we have encountered during this project.

II. Problem Motivation

As the security of our mobile devices has become an integral issue due to the amount of personal information that can be accessed through them, more and more complex ways of authenticating users have come to the forefront. Originally a passcode was used, or even a pattern, but recent developments in phone security have been more focused on biometrics with the advent of fingerprint scanners and facial recognition. If we could combine a passcode with a biometric in some consistently reproducible way, the security benefits would be great. IMU sensor gestures have been researched extensively [3] for their biometric capabilities, and to bring this technology to production one must make the sensor input secure. In order to do so ARM TrustZone is employed, which matches the current security model used by fingerprint sensors.

Figure 1 shows an architecture of *ARM TrustZone*. This technology provides system-wide hardware isolation. It has two environments, *Rich Execution Environment (Normal World)* and *Trusted Execution Environment (Secure World)*. The Secure World is not accessible from the Normal World. Mobile vendors lock the Secure World on their commercial mobile phones and provide SDKs to trustworthy third-parties for their Trusted Application to interface with the Secure World. Therefore, we use a Hikey 960 development board that has the *TrustZone* feature enabled and unlocked for development purposes.

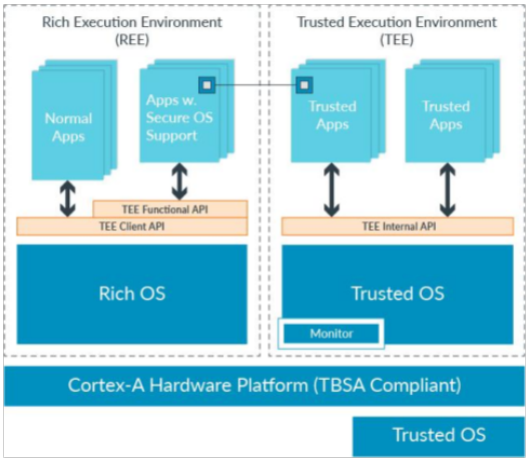

Fig. 1. ARM TrustZone Architecture

III. Related Work

In this section, we compare our system with existing work that uses *TrustZone* in two aspects—authenticated feature and whether or not a server is required.

TABLE I
Comparison to Existing Work

Name	Authenticated Feature				Does Not Require a Server
	Sensor	I/O	UI	Ad	
Our System	✓				✓
Viola [7]	✓				✓
SchrodinText [1]		✓			
SeCloack [4]		✓			✓
Brasser [2]		✓			
TruzDroid [8]			✓		
VButton [6]			✓		
AdAttester [5]				✓	

Authenticated Feature: There are mainly 4 features that existing work have authenticated, and the authentications are done in the *Trusted Execution Environment*. **(1) Viola [7]** authenticates sensor usage. Whenever a particular sensor is being used, it notifies by blinking. Our system also authenticates an accelerometer sensor used by gesture recognition. **(2) Other work [1], [4], [2]** authenticate I/O communication and peripheral devices. These work by either protecting sensitive textual contents or controlling the usage of peripheral devices. **(3) TruzDroid [8] and VButton [6]** encrypt UI actions and provide new verified UIs by moving a display driver and a touch pad controller into the Secure World. **(4) AdAttester [5]** enables unforgeable clicks and a

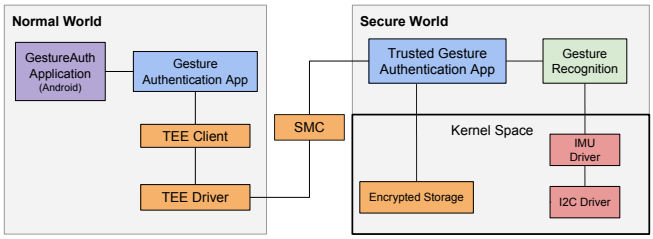

Fig. 2. System Architecture

verifiable display to prevent ad fraud.

Server Requirement: Whether a system requires a server or not depends on what is being authenticated. It's important to keep an assumption that everything in the Normal Wrld can be malicious. Even if an authentication is done in the Secure World, a result may be polluted while being passed to an application layer in the Normal World. Viola [7] and SeCloack [4] do not require a server because they authenticate entirely in the Secure World, and it does not matter whether results are polluted while being passed. However, other work [1], [2], [8], [6], [5] require a server to encrypt their results being passed to an application in the Normal World. Normally, both applications in the Normal World and the Secure World get public key and private key from a server to encrypt and decrypt a result. Actually, our system requires encryption of the authentication decision when it is passed from the Secure World to the Normal World. We do not have an implementation or specific design for how this encryption is to be done, and leave it as a future work for now.

IV. Design and Implementation

In this section, we present an overview of our system's design and implementation details.

A. Design

Figure 2 shows our system's architecture. Different colored components have different characteristics. We divide this architecture into three parts—communication between normal world and secure world, gesture recognition in secure world, and migration of sensor drivers.

Communication between Normal World and Secure World: The purple colored GestureAuth from Figure 2 is an android application requesting a gesture authentication. It calls the gesture authentication app in the Normal World. Blue colored applications communicate with each other between the different worlds using *SMC (Secure Monitor Call)*, which enables context switching. Including *SMC*, all components colored with orange are features given by *OPTEE* build, and those features support communication between different worlds and provide secure storage. When the context is switched to the Secure World, the trusted gesture authentication app starts to recognize a gesture using the Gesture Recognition module colored in green. After a gesture is recognized, the Trusted Gesture Authentication App returns a gesture name or authentication decision (depending on use case) and switches the context to the Normal World.

```
struct serial_chip {
    const struct serial_ops *ops;
};

struct serial_ops {
    void (*putc)(struct serial_chip *chip, int ch);
    void (*flush)(struct serial_chip *chip);
    bool (*have_rx_data)(struct serial_chip *chip);
    int (*getchar)(struct serial_chip *chip);
};

struct serial_driver {
    /* Allocate device data and return the inner serial_chip */
    struct serial_chip *(*dev_alloc)(void);
    /*
     * Initialize device from FDT node. @parms is device-specific,
     * its meaning is as defined by the DT bindings for the characters
     * following the ":" in /chosen/stdout-path. Typically for UART
     * devices this is <baud>{<parity>{<bits>{<flow>}}} where:
     *    baud    - baud rate in decimal
     *    parity  - 'n' (none), 'o', (odd) or 'e' (even)
     *    bits    - number of data bits
     *    flow    - 'r' (rts)
     * For example: 115200n8r
     */
    int (*dev_init)(struct serial_chip *dev, const void *fdt,
                    int offset, const char *parms);
    void (*dev_free)(struct serial_chip *dev);
};

struct io_pa_va {
    paddr_t pa;
    vaddr_t va;
};
```

Fig. 3. Driver API Example

Gesture Authentication in Secure World: The IMU sensor data is read from the sensors directly from the Secure World. This way, it is not exposed to the Normal World. This sensor data is then to be passed into the gesture recognition algorithm. We use the Dynamic Time Warping (DTW) algorithm to calculate a distance from other previously recorded gestures that are stored in the persistent Encrypted Storage. A sample of gesture data is stored and used as a time-series of 3-dimensional accelerometer data. In an authentication scenario, a window of 3D time-series input from the sensors is captured and compared to the stored gesture data for the user attempting to be authenticated. If the distance between the input data and the stored data is below some threshold, then we authenticate the user. If this distance is too large, as in greater than the threshold, then authentication fails. The threshold may vary based on user and gesture. A gesture that is more difficult to reproduce, or a user that is simply worse at reproducing their own gesture, may receive a larger tolerance threshold for authentication.

Migration of Sensor Drivers: OpTEE drivers work much like any other driver as they are simply a hardware interface that operates in kernel space. Creating a driver of a type that already exists (UART for example) is often times more simple than creating a new class of drivers because the infrastructure is already created for UART drivers. This infrastructure includes things like an overarching API for the class of drivers (Fig. 3), syscalls, and examples of how to implement the driver. Creating a new class of drivers involves developing each of these from scratch.

B. Implementation

Root Permissions: Since developing an Android app is not complicated, we do not present the details here. This Android application calls the Gesture Authentication App in the Normal World (binary executable file) using:

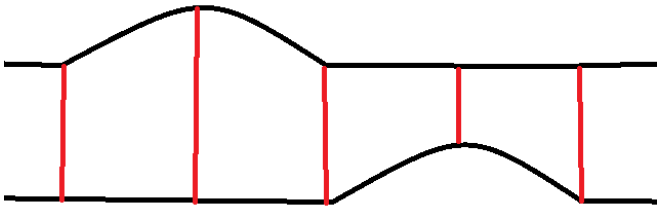

Fig. 4. Time-Series Points Compared Using Euclidean Distance

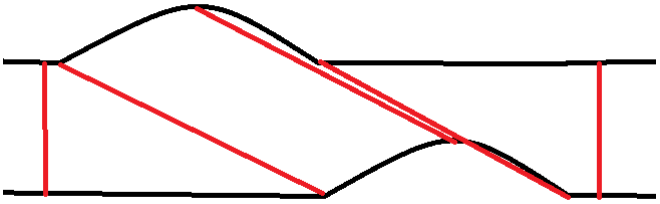

Fig. 5. Time-Series Points Compared Using Dynamic Time Warping

```
Process Runtime.getRuntime().exec();
```

However, *TEE* client APIs used in the app require root permissions, and getting root permissions in a specific app is not simple unlike in *adb* shell. This is because giving such permission to the specific app is extremely dangerous. We solve this problem by changing the mode of *SELinux* to permissive mode.

Gesture Authentication Apps between Normal World and Secure World: After opening a session for a communication, the normal Gesture Authentication App invokes two *TEE* commands, one for calculation and another one for requesting a gesture name. A calculation invoking command passes a data number to calculate in *TEE*. It should pass the data number because sensor data sets are predefined in the Secure World for now. This invoke will not be needed when the sensor drivers are successfully implemented in the Secure World later. If a result is passed right after the calculation, it throws an error. Since debugging a process in the Secure World is hard, we have not debugged the issue yet and make two invoke commands instead.

Gesture Comparisons: Due to complications in development, the entire scope of the design for Gesture Authentication laid out in the last section was not achieved. For one, we could not read the sensor data directly into the Secure World due to complications with migrating the sensor drivers. This is discussed in more detail later. Therefore, instead of a live data feed, we recorded data on an external device that had IMU sensors and then stored that data statically in the Secure World for testing purposes. We also did not maintain a database of users and their stored gestures, nor did we implement the thresholding system proposed. This was all due to time constraints placed on us by our other setbacks. What we do successfully is compare two stored gestures in the Secure World using the DTW algorithm and return the calculated distance between them to the Normal World. This demonstrates both the secure comparison of two gestures in the Secure World as well as the ability to send information back to the Normal World. In a real system you would not want to send the distance back to the Normal World, but rather an authentication decision made using that distance in the Secure World. The details of the DTW algorithm are discussed next.

Calculation of *DTW (Dynamic Time Warping)*: Dynamic Time Warping is an algorithm designed to overcome the misalignment of time-series data. For example, consider we have two very similar time-series, but they are offset from each other by a few seconds. Using the regular euclidean distance between data-points, such as in Figure 3, we see that very dissimilar data points get compared to one another. The similarity of these two time-series is completely missed in this calculation! The goal of DTW is to find the distance between the points representing similar features of the time-series', effectively re-aligning them, as shown in Figure 4. This process is approximate, and the bigger the offset or more noisy the data, the less accurate this re-alignment will be.

The algorithm proceeds by first constructing an M x N cost matrix, where M is the length of the first time-series, which we'll call A, and N is the length of the second time-series, which we'll call B. The length of a time-series is how many discrete data-points it has. Starting from the coordinate (0,0), the matrix, which we'll call CM, is filled in from left-to-right then bottom-to-top according to the following rule:

```
CM[i, j] = abs(Ai - Bj) + min(CM[i-1, j-1],
                              CM[i-1, j],
                              CM[i, j-1])
```

Simply put, the value from B at the current coordinate is subtracted from that of A. Then, the minimum value of all previous adjacent locations is added to the absolute value of the result. (An unwritten detail is if the value being calculated is at the edges of the matrix so there is no previous adjacent location, we use infinity (practically, MAX_FLT) as a placeholder to effectively exclude it from the min calculation).

Now that we have constructed our cost matrix, we must now find the optimal path through the matrix and keep a running sum of all the values on that path. We start from the maximum coordinate in the matrix (top-right, in this case) and work backwards. We add the current value to the running sum, then look at the 3 previous adjacent values and choose the minimum value. Repeat this process for each chosen value until you reach the bottom-left of the matrix. You have followed the optimal path and should have a sum of all the values along it. This sum is the DTW distance, and is what is returned by the DTW algorithm.

I2C Driver: Inter-integrated circuit (I2C) interfaces are fairly common among hardware devices as it is second only to UART as the most simple to implement and use.

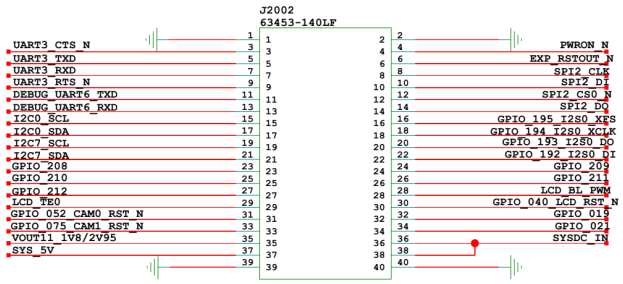

Fig. 6. Hikey960 External Connector Schematic

Processors will typically provide a number of I2C interface pins and a controller to deal with all of the hardware real-time requirements of the protocol. The HiKey960 is no exception and even provides I2C7s pins broken out to an external connector on pins 19 and 21 (Fig. 6). The base address for these I2C registers can be found in the HiKey960 SoC Reference Manual. Synopsis Designware I2C controller is used and its register definitions can be found in the HiSilicon Hi6220V100 Multi-Mode Application Processor Function Description document. From here the drivers init, read, and write functions can be written by reading each of the registers descriptions. Special considerations need to be taken to disable the controller before writing to some registers that change the configuration. Similarly, the controller uses FIFOs to queue data when receiving or transmitting. Before requesting more data the receive FIFO should be checked and before sending more data the transmit FIFO should be checked.

IMU Sensor Driver: The IMU sensor driver leverages the I2C driver previously described to communicate with the SparkFun LSB9DS1 chip. To change the configuration of an I2C device one must initiate an I2C write. Each I2C device has a hard coded address for each major operation (read/write) as well as many minor addresses to index into its register bank. Fig. 7 shows a generic write to the device: a start condition is sent by the master, the slave address plus a write bit is sent by the master, slave acknowledges, the sub-address is sent by the master, slave acknowledges, the data to set the register to is sent by the master, slave acknowledges, and the master sends the stop condition. Similarly for the read in Fig. 8, the master sends a repeated start condition with the read bit set on the second. This IMU sensor has the ability to stream data until the Master requests a stop. There are also many configuration options for this sensor to send data at a faster rate (10- 1000Hz) as well as different schemes for determining which data to send; The chip can send accelerometer only or acclerometer and gyroscope data. The chip also provides on-board FIFOs that can either be put in bypass mode (get the latest data), FIFO mode (FIFO will not be updated once filled), and continuous mode (FIFO will be overwritten once filled). The most useful modes for gesture recognition are bypass and continuous because the latest data is the most important. In FIFO mode the data

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Fig. 7. IMU Sensor Write Sequence

Master	ST	SAD+W		SUB		SR	SAD+R			MAK		MAK		NMAK	SP
Slave			SAK		SAK			SAK	DATA		DATA		DATA		

Fig. 8. IMU Sensor Read Sequence

could be stale and could cause false negatives.

Writing Syscalls in OpTEE: In order to provide trusted applications (TA) with access to hardware interfaces and other low level operations OpTEE implements a static library called libutee, which indirectly exposes syscalls to the developer. Syscalls are created using the `UTEE_SYSCALL` macro in assembly providing an entry point name, the syscall number, and the number of arguments. The entry point name argument is defined in `utee_syscalls`.h as a function prototype and the macro creates the function definition which switches contexts to kernel space. In kernel space the `SYSCALL_ENTRY` macro is used to define a new syscall in the array `tee_svc_syscall_table`. This table of syscall entries must be in the same order of syscall numbers defined in libutee. The new entry point created in this table is defined in its own file, and mirrors the syscall entry point defined in libutee; it has the same number of arguments in order to pass data from user space to kernel space.

There are two approaches to creating syscalls for a custom module or driver. The first is to create a syscall for each major operation to provide greater granularity and clarity to the user. The second is to create only one syscall and use one of the arguments as an `operation_type`. Since the OpTEE architecture provides libutee as a shim layer between the user and the syscalls a combination of these methods was used. A single syscall is created that takes an `operation_type` argument, but the libutee API provides multiple calls to the TA developer at a more granular level. These methods each call the same syscall, but with different operation types. This allows for greater code re-use in the driver and a generally more simple architecture.

V. Evaluation

In our most basic comparison example, we show how two similar gestures can be authenticated when compared to two dissimilar gestures. We had Person A record a circular gesture 10 times. Then we had Person A record a Z-shaped gesture 10 times. We found that the average distance between Person A's circular gestures when compared to the other 9 circular gestures was around 1232 with a standard deviation of 227. Likewise, the average distance between Person A's Z-shaped gestures when compared to the other 9 Z-shaped gestures was around 1865 with a standard deviation of 401. However, the average distance between Person A's circular gestures when compared to the 10 Z-shaped gestures was 2790. This is significantly larger than when comparing similar gestures, and shows us clearly that these can be

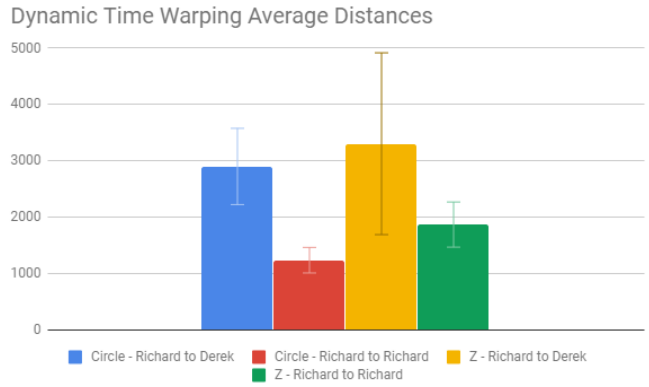

Fig. 9. Dynamic Time Warping Average Distances Between Users and Gestures

recognized as different gestures by using DTW distance.

Things became a little more interesting when we added a Person B to the mix. We had Person B record 10 circles and 10 Z-shaped gestures. Person B was instructed to try to perform them in the same way Person A had. We found that the average distance between Person A's circular gestures when compared to Person B's circular gestures was a whopping 2898 with a standard deviation of 678. We also found that the average distance between Person A's Z-shaped gestures when compared to Person B's Z-shaped gestures was an even larger 3301 with a standard deviation of 1614.

To discover the source of this, we looked closer at the data. It turns out, the reason Person A's gestures were so much different than Person B's gestures, despite being conceptually the same, has to to with the fact that Person B's gestures were far less consistent. Person B's circles had an average distance of 2562 among themselves, and Person B's Z's had an average distance of 3328 among them. However, Person B's circles when compared to Z's have an average distance of 5042. What this shows us is that we can still recognize when Person B's gestures do not match, however, this is an example where, because Person B is less skilled at reproducing the exact gesture over and over again, the threshold for authentication would have to be raised to suit him. What kind of security implications this may have is left up to future research.

This information also points subtly to a biometric component to authentication itself. As you can see, Person A's circles vs Person B's circles still had a higher distance between them than Person B's circles against themselves. Same goes for Z's. This does present a small but present biometric component to gesture authentication, in that it is potentially difficult for someone to faithfully reproduce someone else's gesture. This effect may be more pronounced with more complex and personalized gestures, such as signing one's name in the air. Exploring this idea is left up to future research.

VI. Conclusions

In this paper, we present a gesture authentication system using *TrustZone*. It successfully handles gesture recognition and calculation in the Secure World that cannot be easily forgeable. Also, our evaluation section covers statistics gathered from *Dynamic Time Warping* distances on sample data. We believe that previous sections successfully show a feasibility of gesture authentication using Secure World technology similar to how it is used in state-of-the-art fingerprint and facial recognition.

Even we could not finish building drivers for the IMU sensors in OP-TEE and leave some parts for future work. All project team members have enjoyed learning about TrustZone technology and have been satisfied to demo successfully running gesture authentication in the Secure World.

References

[1] A. Amiri Sani. Schrodintext: Strong protection of sensitive textual content of mobile applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, pages 197–210, New York, NY, USA, 2017. ACM.

[2] F. Brasser, D. Kim, C. Liebchen, V. Ganapathy, L. Iftode, and A.-R. Sadeghi. Regulating arm trustzone devices in restricted spaces. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 413–425, New York, NY, USA, 2016. ACM.

[3] C.-W. F. Ho-Man Colman LEUNG and P.-A. HENG. Twistin: Tangible authentication of smart devices via motion co-analysis with a smartwatch. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, Volume 2 Issue 2, June 2018, pages 72:1–72:24, New York, NY, USA, 2018. ACM.

[4] M. Lentz, R. Sen, P. Druschel, and B. Bhattacharjee. Secloak: Arm trustzone-based mobile peripheral control. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, pages 1–13, New York, NY, USA, 2018. ACM.

[5] W. Li, H. Li, H. Chen, and Y. Xia. Adattester: Secure online mobile advertisement attestation using trustzone. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 75–88, New York, NY, USA, 2015. ACM.

[6] W. Li, S. Luo, Z. Sun, Y. Xia, L. Lu, H. Chen, B. Zang, and H. Guan. Vbutton: Practical attestation of user-driven operations in mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, pages 28–40, New York, NY, USA, 2018. ACM.

[7] S. Mirzamohammadi and A. Amiri Sani. Viola: Trustworthy sensor notifications for enhanced privacy on mobile systems. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 263–276, New York, NY, USA, 2016. ACM.

[8] K. Ying, A. Ahlawat, B. Alsharifi, Y. Jiang, P. Thavai, and W. Du. Truz-droid: Integrating trustzone with mobile operating system. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, pages 14–27, New York, NY, USA, 2018. ACM.