COMS6424: ARMTrustZone Secure Display

Yizhi Wang
yw4174@columbia.edu
Columbia University
New York, USA

Qiuhong Chen
qc2335@columbia.edu
Columbia University
New York, USA

ABSTRACT

Nowadays, IoT devices and smartphones are becoming increasingly common in our daily lives outdoors. These portable or lightweight devices can collect and acquire information through various I/O peripherals and accessories such as sensors and cameras, providing convenient and intelligent services. However, security remains a challenge for these devices, especially since they often run multiple applications on the same physical device. This can pose a risk to the entire device if one of those applications is malicious. Therefore, decoupling and providing security isolation is an essential solution to these issues. One popular solution today is the Trusted Execution Environment (TEE), which offers secure software services protected and monitored by CPU hardware. Among many hardware-implemented TEE products, ARM TrustZone is the mainstream choice for embedded devices.

In this paper, we explore the implementation of a secure video display system using ARM TrustZone, called SecDisplay, which consists of three main components: a trustworthy remote server, an untrusted local client application, and a Trusted Application (TA) within the TEE OS in the secure world. Each component is vital for establishing a secure channel, encrypting video transmission, invoking the TA, and displaying video content. Although our OP-TEE setup did not support video output, as detailed in Section 4, this research allowed us to experiment with real-world TrustZone devices and deepen our understanding of TrustZone's operational and development environments. We also gained a clearer understanding of the design principles underlying hardware-software integrated systems. Looking ahead, we believe that a coordinated hardware-software co-design approach and expertise in full-stack development will become increasingly crucial.

ACM Reference Format:
Yizhi Wang and Qiuhong Chen. 2024. COMS6424: ARMTrustZone Secure Display. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As digital technology advances, a vast amount of data is generated and stored on digital devices. While much of this data is benign and harmless if accessed by others, some private and critical data requires secure protection and storage. This need has led to the development of ARM TrustZone technology, designed to isolate sensitive data from regular data and applications by dividing the computing environment into a normal, unprotected environment and a secure environment through hardware enforcement.

In this paper, we explore the use of ARM TrustZone for enabling trustworthy video displays. This system has potential widespread applications in various scenarios, such as smart home cameras that monitor home security. These cameras' network interfaces may be vulnerable to potential attacks, resulting in compromising the entire system. A trustworthy video display can address this issue effectively. By utilizing ARM TrustZone's support for secure I/O operations, the data flow from camera capture to display can be protected under TrustZone. We also consider a scenario where a device owner wants to download and play videos from a remote server. The entire process of communicating with the server and decrypting the video is strictly protected within TrustZone, similar to applications like Snapchat or some scenarios, such as playing videos from iCloud.

To implement the system described in the second scenario, we primarily designed and implemented three main components: a trusted remote server, an untrusted local client application, and a Trusted Application (TA) with TEE OS in the secure world. To ensure secure transmission between the remote server and the TA, the remote server uses a Diffie-Hellman handshake module for network communication to securely encrypt and send videos. For transmission between the local client application and the TA, the local client routes network packets and stores the encrypted video locally. The TA and TEE OS then operate securely to handle video decryption and secure display, using modules for establishing secure communication, decrypting videos, and managing access to the video display frame buffer.

The major efforts and contributions of the paper can be summarized as follow:

- This paper explores and proposes a secure video display system leveraging ARM TrustZone. The system is called SecDisplay.
- The paper designs and assesses the interaction and roles of three major components: a trusted remote server, an untrusted local client application, and a Trusted Application (TA) within a Trusted Execution Environment (TEE). The remote server responsible for video encryption and secure key negotiation; the untrusted local client that handles packet routing; and the TA that ensures secure video decryption and display.
- The system shows how TA can be maintained for video decryption and perform memory page access control in the presence of untrusted normal world OS, using technologies like Diffie-Hellman handshake modules, AES for secure key exchange and video decryption, and MMU for display frame buffer permission control.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

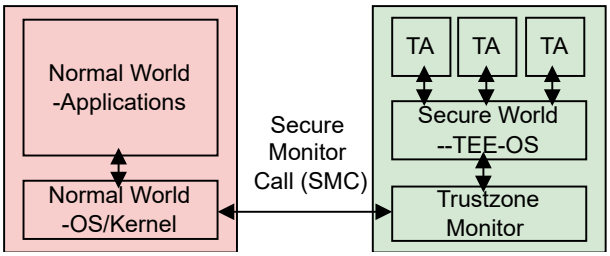


Figure 1: Mode switch in Arm Trustzone.

Road map. The rest of the paper is organized as follows: Section 2 introduces related background about ARM TrustZone and OP-TEE. Section 3 presents the overview of the SecDisplay design and design challenges. Section 4 details how SecDisplay implements the three components and their related modules. Section 5 conducts an evaluation of SecDisplay's performance. Section 6 discusses future works and some limitations. Lastly, Section **??** concludes the paper.

2 BACKGROUND

2.1 ARM TrustZone

TrustZone offers two distinct execution environments: the normal world and the secure world. By partitioning hardware resources between these environments, it allows the creation of secure data within a Trusted Execution Environment (TEE), as defined by GlobalPlatform. The secure world processes secure data operations, while the normal world manages all other data. TrustZone features a "world switch" function that facilitates toggling between the normal and secure worlds as needed, preserving the integrity of each environment. Due to these capabilities, TrustZone is utilized in various applications including secure boot and personal data protection, offering an efficient, cost-effective security solution based on ARM architecture.

SMC Secure Monitor Call (SMC) is an instruction used in ARM architecture to facilitate the transition between the normal world and the secure world. This is part of the security operations in TrustZone, allowing for controlled access to secure resources while maintaining the integrity and isolation between the two worlds.

Memory Management Unit (MMU) The system incorporates two virtual Memory Management Units (MMUs), one for each world. These MMUs manage memory access rights and address translation, ensuring secure separation of data. Each environment utilizes its own translation tables to map virtual addresses to physical addresses independently. The Translation Lookaside Buffers (TLB) are tagged with Secure/Non-secure (S/NS) bits to indicate the security status of memory blocks, further enhancing the isolation between secure and non-secure memory areas.

Secure I/O Secure Input/Output (I/O) mechanisms are integral to maintaining data security, especially in environments where data integrity and confidentiality are paramount. These mechanisms ensure that all I/O operations in the secure world are isolated and protected from unauthorized access and manipulation, supporting secure peripherals and access controls.

OP-TEE OP-TEE is a software realization of a TEE that complies with GlobalPlatform standards and is based on the ARM architecture. It implements APIs for communication with Trusted Applications (TAs) and the TEE itself. OP-TEE includes optimized functions for secure storage, encryption, decryption, and more, safeguarded by secure APIs. Its portability makes it suitable for initial secure deployments and can be easily adapted for physical boards after successful emulation testing. In the development process, QEMU serves as the emulation environment. It replicates the ARM architecture and supports TrustZone, providing a testing ground before deployment on actual hardware.

2.2 TrustZone Existing Secure/IO Work

[2] An existing work regarding secure analysis of the data stream is called StreamBox TZ, which uses the TrustZone to isolate the data generated from edge computing and avoid the affection of other malicious software stacks in the normal world. In this work, the authors designed a Data Plane to protect the analysis and computation of secure data, which utilized the TEE provided by TrustZone to ensure the security and integrity of data. Similarly, to achieve a secure display, it is also important to isolate the confidential data from the regular data using TEE.

3 OVERVIEW

In this section,

3.1 System Overview

In our system, there are three main components: a trustworthy remote server, an untrusted client application in the normal world, and a Trusted Application (TA) with the TEE OS in the secure world. To facilitate secure video playback, the modules implemented by each component are illustrated in Figure 2.

- **Remote Server.** The remote server is a trusted component located on a remote machine. It is responsible for establishing an encrypted network connection with the TA and negotiating a symmetric AES key for video encryption. The server then encrypts the video and transmits the encrypted video to the TA. Therefore, on the server side, the primary code components to be developed include two modules: a *DH handshake module* for setting up the encrypted network communication with the TA and negotiating the symmetric AES key, and a *Video encryption module* that uses this AES key to encrypt the video and transmit it to the TA.
- **Local Client Application.** The local client application in the normal world is an untrusted component. It is responsible for routing network packets between the TA and the remote server. The communication with the remote server occurs over a standard network connection, while communication with the TA is facilitated through SMC calls provided by TrustZone's secure monitor. Additionally, when the TA is not actively displaying anything, the local client application is also permitted to display content on the screen. Therefore, the primary code components to be developed for the local client application include two modules: a *Diffie–Hellman (DH) handshake module* for network packet

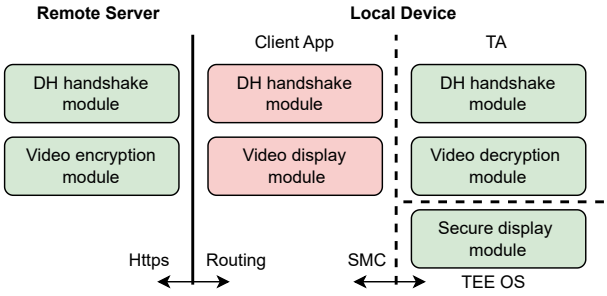


Figure 2: Major modules in SecDisplay.

routing, and a *Video display module* that allows the application to display content on the screen when the TA is inactive.

- **Local Trusted Application (TA) and TEE OS.** The local TA and TEE OS are trusted components operating within the secure world. The TA defines multiple functional interfaces that the client application can trigger via SMC calls. There are three major modules that need to be protected within the secure world. First, the TA is responsible for using a *DH handshake module* to establish a secure channel with the remote server. Although all network packets are routed through the untrusted client application, the TA and remote server can securely negotiate an HTTPS session key and a symmetric AES key for video encryption and decryption with the help of DH handshake procedure [1]. Second, the TA contains a *Video decryption module* that utilizes the AES key to decrypt the encrypted video received from the remote server. Lastly, the TEE OS is tasked with adjusting the permissions for the physical address of the video display frame buffer, ensuring that only the secure world has access to the frame buffer when the TA is secure displaying the decyprted video. This module is called *Secure display module*.

3.2 System Challenges

4 IMPLEMENTATION DETAILS

4.1 Remote Server

For the details of the secure display, the server would have the public key of TA and finish the Diffie-Hellman(DH) protocol invoked by the client. Then, the server encrypts the video using the AES Key generated from the DH protocol, and the encrypted video will be sent back to the client and stored there until it is displayed. Whenever it needs to be displayed, it will be sent to the TA chunk by chunk and decrypted there using the AES key generated from the DH protocol. The remote server is responsible for:

(1) Managing connections with multiple video clients.
(2) Diffie-Hellman Key exchange with clients.
(3) JPEG encoding and RSA encryption
(4) Streaming video frames to video clients

We haven't debugged the Diffe-Hellman protocol, and our current version sticks to RSA asymmetric encryption, i.e. we use the client-side key-pair for encryption and decryption. Luckily, the application only requires server-to-client secret transmission, so RSA encryption will satisfy the need.

The test server is implemented using *Python3.10*. It has 3 threads to satisfy the basic functionalities, which include: 1) listening to client connection requests, 2) Diffie-Hellman handshaking with the clients, and 3) streaming video frames to the client. Each video is 240×320 resolution and 30 FPS. The video frame is encoded in JPEG with *openCV2* and encrypted in PKCS#1 OAEP.

The RSA key size is set to 1024 bits. As PKCS#1 OAEP utilizes SHA hashing, the plain text size is limited to $\lfloor n/8 \rfloor - 2 \times \lceil h/8 \rceil - 2$. We use a 1024-bit RSA key and SHA-256 ($h = 256$), so the longest binary that it can encrypt is 62 bytes.

The video frame is in 62-byte segments and encrypted separately. 5 encrypted frames are combined into a video chunk, which is sent through the client connection.

4.2 Client Program

The client program is responsible for:

(1) Diffie-Hellman key exchange with the server.
(2) Receiving video frames and pass to the trusted program.
(3) Video display.

The client program retrieves the RSA public key from the trusted program, which includes a modulus N and an exponent E, and forwards it to the server. For a 1024-bit RSA key, the modulus is 1024 bits. The public key of the server is kept in the client program memory. The client then computes the shared AES key.

The client program retrieves the video frames from the received chunk and gets the frame segments. It then sends the segments to the trusted program for decryption. At last, it assembles the video frames to get the JPEG image.

4.2.1 Video Display Module. STM32MP135F-DK is a physical board that supports installing the OPTEE, and this board is used to create an interactive user interface for users to control and display a short video. The GNOME Toolkit (GTK) is utilized in this part of the design for developing the display interface since it supports the displaying function. For the realization, after installing the needed dependencies, the DrawingArea widget is installed for the area of video displaying and a button is added for stop and play the video. A function named *on_realize*() is used for adapting different environments of video displaying. A short video is successfully displayed on this STM32MP135F-DK board, which shows a feasible solution of video display in a resource-limited environment.

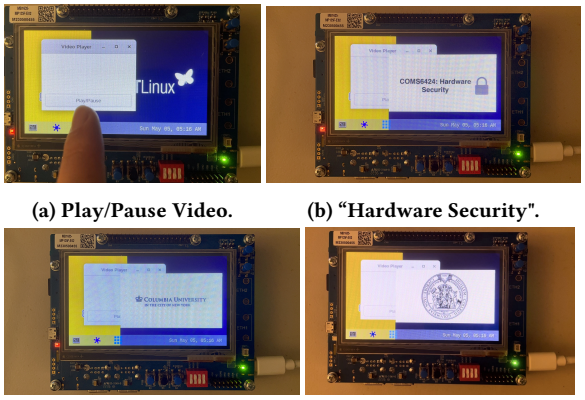
4.3 Trusted Application and TEE OS

The trusted application is responsible for:

(1) Generating and storing the RSA key pair.
(2) RSA decryption for frame segments.
(3) Secure display.

Trusted Application uses the*GlobalPlatform TEE Client API* to perform secure crypto operations and communicate with the client program. In each video session, the trusted application generates a new RSA key pair with *TEE-GenerateKey* method and stores it in secure memory. The trusted application uses the *TEE-AsymmetricDecrypt* method to decrypt the video segment.

4.3.1 Video Decryption Module. Based on the built environment, another consideration for the design was the video encryption and



(a) Play/Pause Video. (b) "Hardware Security".

(c) "Columbia University". (d) "Columbia Logo".

Figure 3: Video display Demo in normal world.

decryption that should be done in the server and the TA ends. Diffie-Hellman, then, is being selected to achieve the goal of not leaking the data even if in the normal world which could be attacked by adversaries. This requires the server and the TA to have each other's public key, and then generate an AES key using their private key. By applying the Diffie-Hellman key exchange protocol, the server can encrypt the data using the generated AES Key, and the TA can decrypt the encrypted data using the same key, and the TA is the only one that has the key. The implementation details are explained in the Design part below.

4.3.2 Secure Display Module. The primary objective of the secure display module is to accurately configure the permissions for the frame buffer. Specifically, it's essential to ensure two things: (1) the frame buffer is exclusively accessible by the secure world when the TA decrypts part of the video and displays it on the screen, and (2) the normal world is able to display on the screen when the TA is not displaying an encrypted and protected video. To accomplish this, we need to configure the Memory Management Unit (MMU) and associated memory access control features, such as the TrustZone Address Space Controller (TZC). In this section, we discuss how we explored the implementation of a secure display module on two different platforms supporting TrustZone: the OP-TEE's QEMU implementation and the `Stm32mp135f-dk` development board.

① **QEMU Implementation.** Our initial implementation of the DH handshake exchange was mainly conducted within the OP-TEE QEMU emulator. This led us to explore how to implement a secure display module within the OP-TEE on QEMU. As a powerful device emulator, QEMU can simulate major physical devices. In our testbed using OP-TEE's QEMU v8, QEMU emulates an Armv8-A processor. It also simulates a storage device to store the corresponding OP-TEE system (including both normal-world OS and secure-world TEE OS), and the firmware boot stage during startup (Uboot).

However, it is very challenging to implement display capabilities within OP-TEE on QEMU. Through our exploration, we have listed our findings and reasons:

- **Default Serial Ports in OP-TEE.** The default and only output of OP-TEE is two serial ports, which sequentially and automatically print system logs and messages from the normal world and the secure world, respectively. We believe this is because many embedded devices that support TrustZone do not actually include a physical video display monitor. Consequently, in OP-TEE's implementation, video display support is absent for both the normal world and the secure world.
- **No Display Emulation.** Because the basic OP-TEE only supports serial ports, when launching OP-TEE using the default command line provided by OP-TEE with QEMU, QEMU only emulates a disk storage device and does not simulate a display or monitor.
  Therefore, we attempted to add a video display emulation to the QEMU-emulated OP-TEE device. We achieved this by (1) reconfiguring OP-TEE's QEMU version to support video display, (2) using QEMU to emulate a video display and attaching it to QEMU-emulated Arm CPU.
  As shown in Figure 5, OP-TEE's default QEMU version only supports two display options. One is called Curses, which provides terminal-based text-like output for the user interface. The other is called dbus, which facilitates inter-process communication but does not offer graphical display capabilities. Thus, we reconfigured OP-TEE's QEMU to enable GIMP Toolkit (GTK) and Simple DirectMedia Layer (SDL). Both help QEMU to have the ability to manage graphical user interfaces and enable video, audio, and input device emulation.
  We then modify the command used to start QEMU:
  ```
  QEMU_BASE_ARGS +=
  -display gtk -device virtio-gpu-pci -vnc :0
  ```
  This command causes QEMU to emulate a video display monitor that supports the virtio-gpu-pci driver through GIMP Toolkit (GTK), which is a widely used graphics library for creating graphical user interfaces. The output of the video display is forwarded to port 0 of a Virtual Network Computing (VNC) server using the VNC protocol, allowing a remote or local display via an VNC client application.
- **Lack of Display Driver Support.** A display driver is crucial in the functioning of a display; it is responsible for recognizing the device at startup and using Memory-Mapped I/O (MMIO) to map the frame buffer in memory. For the same reasons, OP-TEE does not include display drivers when compiling the Linux kernel used in the normal world.
  Therefore, we reconfigured the Linux kernel configuration file to enable support for the `virtio-gpu-pci` driver. As shown in Figure 5, after reconfiguring and recompiling the linux kernel, the normal world's kernel now includes the `DRM_VIRTIO_GPU` support.
  Even with these configurations, the emulated video output did not display any images, as shown in Figure 6. We think there could be two reasons for this: (1) The lack of an operating system that supports a graphical user interface (GUI) might mean that while the video display was correctly connected to the QEMU-emulated device that supports TrustZone, there was no program writing to the frame buffer. We later ruled out this possibility because we checked the "/dev" directory and used the `lspci`

```
trustzone@trustzone:~/optee/qemu/build$ ./qemu-system-aarch64 -display help
Available display backend types:
none
curses
dbus
```

Figure 4: Original available display backend supported by original OP-TEE QEMU v8.

```
# zcat /proc/config.gz  | grep VIRTIO_GPU
CONFIG_DRM_VIRTIO_GPU=m
CONFIG_DRM_VIRTIO_GPU_KMS=y
#
```

Figure 5: Modify kernel configuration to enable GPU driver in OP-TEE's normal world.

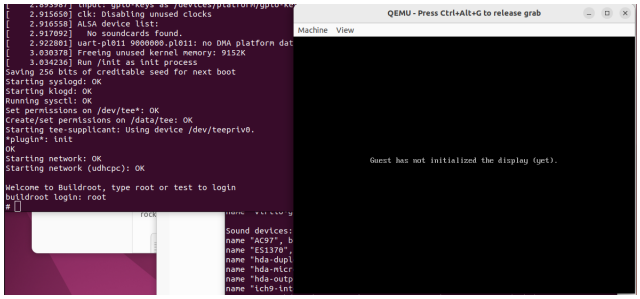


Figure 6: Appending a QEMU-emulated display to OP-TEE QEMU implementation.

command to list all PCIe devices connected to the emulated ARM processor. This check did not reveal any additional video display devices, which indicates the first reason might not be the case. (2) OP-TEE might not properly recognize the display during the boot process (Uboot), leading to the display drivers being unable to correctly set the frame buffer via MMIO Addressing this issue likely requires a deep understanding of firmware startup and driver implementation, making it quite challenging for us to resolve.

- **Lack of GUI-supported OP-TEE.** Another approach we considered was using an operating system that supports a graphical user interface, such as the Android Open Source Project (AOSP). However, Todo, talk about hikey etc, many github issues also points out that "X is not enabled, and we have not built nor enabled any drivers for graphics."

Based on the reasons mentioned above, adding a video display to an ARM CPU emulated in QEMU presents a significant challenge, especially since there are very few tutorials online on how to modify the UBoot boot process. Additionally, some GitHub issues indicate that adding display emulation to OP-TEE QEMU is particularly difficult, leading many developers to give up in the endadd reference. Instead, we decided to purchase some development boards online to see if it is possible to enable trusted I/O and secure display on actual hardware.

② **`Stm32mp135f-dk` Development Board Implementation.** Device Tree Source (DTS)

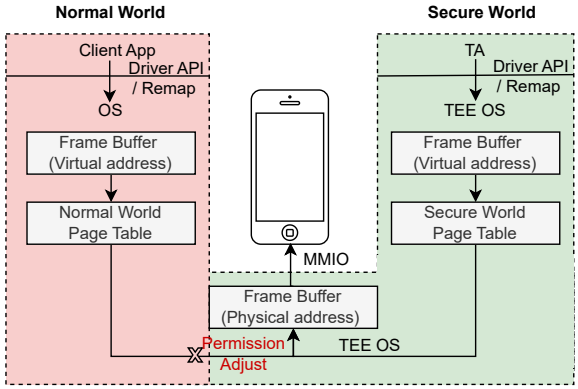


Figure 7: Memory control in SecDisplay.

```
model = "STMicroelectronics STM32MP135F-DK Discovery Board";
compatible = "st,stm32mp135f-dk", "st,stm32mp135";
reserved-memory {
    #address-cells = <1>;
    #size-cells = <1>;
    ranges;
    optee_framebuffer: optee-framebuffer@dd000000 {
        /* Secure framebuffer memory */
        reg = <0xdd000000 0x1000000>;
        st,protreg = <TZC_REGION_S_RDWR 0>;
    };
};
```

Figure 8: Device Tree Source description of the frame buffer in core/arch/arm/dts/stm32mp135f-dk.dts.

After some exploring, the OP-TEE frame buffer is a protected and secure memory region used for display. The frame buffer region is set during the boot period and protected by TrustZone Address Space Controller (TZC).

As shown in Figure 8, This defines a node named optee_framebuffer, mapping a specific memory region for a secure framebuffer. The reg property specifies the start address (0xdd000000) and size (0x1000000, i.e., 16MB) of the memory. The st,protreg is a custom attribute that may relate to STMicroelectronics' hardware protection features. TZC_REGION_S_RDWR indicates that this memory region can be accessed for reading and writing in a secure state.

Needs to flush cache using the `TEE_CacheFlush` from `tee_internal_api_exte`

Three Development boards we have tried. As shown in Figure 9, we have tried 3 different models of development boards in total. Among these development boards, OP-TEE does not support the MSP430 (the bottom left board). For the Raspberry Pi 3B (the top left board) and STM32MP135F-DK (the right two boards), the versions of OP-TEE capable of running TrustZone, as listed on the official OP-TEE website, do not include support for GUIs or corresponding display drivers. Directly flashing the compiled OP-TEE image onto the STM32MP135F-DK development board allows it to operate and boot up correctly. However, it does not produce any video output. The current display system in the STM32MP135F-DK is a GUI-supported OS from STMicroelectronics, known as the OpenSTLinux distribution, which is based on the OpenEmbedded/Yocto Project.

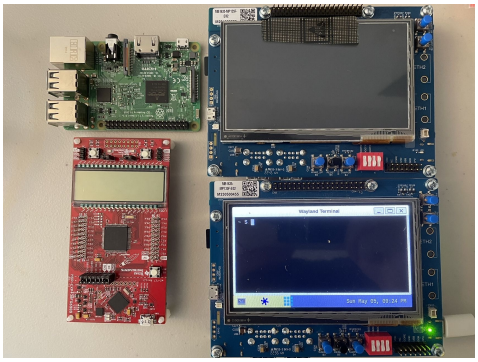
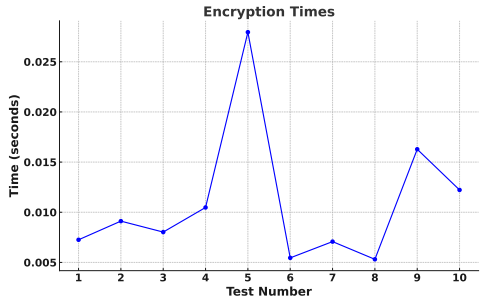
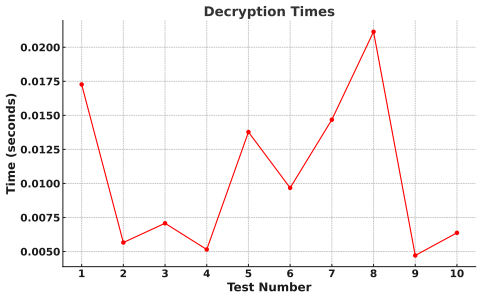


Figure 9: Development boards we have tried.



(a) Encryption Time.



(b) Decryption Time.

Figure 10: Encryption and decryption time for ten times.

5 EVALUATION

@qiuhong, please help fill your content in this section in this section

A time evaluation of encrypting and decrypting a string is done to help with understanding the estimated time of video encryption and decryption. As shown in Figure 10(a), each point represents an encryption time finished by the $aes_t a.c$ file; and in (b), each point represents the decryption time. By repeating the test ten times, the result shows that the encryption times are lower than the decryption time in a general view of the data, which may caused by the extra tests and comparisons that should be done during the decryption processes, such as integrity verification and error checks, and it also may be caused by accessing the storage more frequently in decryption process. This result is a reminder that the decryption process of the video should be optimized in order to reduce users' waiting time. The result not only shows the encryption and decryption time of a simple string, but it also reveals some limitations that should be handled in further steps.

6 DISCUSSION

7 CONCLUSION

This paper presented SecDisplay, a secure video display system using ARM TrustZone. The system's architecture, comprising a trustworthy remote server, an untrusted local client application, and a TA within the TEE OS, demonstrates how to enable security isolation for video transmission and display. While our OP-TEE setup did not support direct video output, the project provided valuable insights into TrustZone's capabilities and the importance of a hardware-software co-design approach. Future work will aim to figure out how to setup a GUI-supported OS and securely display video in a real-world TrustZone device.

ACKNOWLEDGMENTS

We want to thank Professor Simha Sethumadhavan for helping us to understand Hardware Security's knowledge and the weekly instruction and support based on our project progress, and Ryan, our TA, for his support through our course studies.

REFERENCES

[1] Ueli M Maurer and Stefan Wolf. 2000. The diffie–hellman protocol. *Designs, Codes and Cryptography* 19, 2 (2000), 147–171.

[2] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. 2019. StreamBox-TZ: Secure stream analytics at the edge with TrustZone. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 537–554.