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, 8 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 Trusted Execution Environment (TEE) 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. Common commercial hardware-protected TEE technologies can be categorized as local TEE technologies, such as ARM TrustZone, and numerous cloud TEE technologies, such as AMD Secure Encrypted Virtualization (SEV) [27], ARM Confidential Compute Architectrue [7, 32], Intel Software Guard Extensions (SGX) [6], and Intel (TDX) [5].

Among these hardware-protected TEEs, TrustZone is the early and a relatively lightweight TEE design that is widely available in embedded devices or smart phones that use ARM CPU. 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

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

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.

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 7 concludes the paper.

2 BACKGROUND

2.1 ARM TrustZone

ARM TrustZone is ARM's design to offer a hardware protected environment. Specifically, TrustZone offers two distinct execution environments: the normal world and the secure world. By partitioning hardware resources between these environments, it allows the creation and processing of secure data within a Trusted Execution Environment (TEE), so called secure world. The secure world processes secure data operations, takes control of security-related applications, and has higher privileged value compared to the normal world. The normal world, on the other hand, manages all other data and less secure applications. TrustZone features a "world switch" mechanism (Secure Monitor Call) that facilitates toggling between the normal and secure worlds as needed, preserving the integrity of each environment. Due to these capabilities, TrustZone is currently used in various applications and scenarios, including secure boot and personal data protection, offering an efficient, cost-effective security solution based on ARM architecture.

Secure Monitor Call (SMC). Secure Monitor Call (SMC) is the mechanism used in ARM TrustZone architecture to facilitate the transition between the normal world and the secure world. As shown in Figure 2, SMC is one essential part of the world-switch in TrustZone, allowing for controlled access and interfaces to trigger and access secure resources stored and executed in the secure world. The SMC design can help TrustZone developer to easily maintain the integrity and isolation between the two worlds.

Memory Management Unit (MMU). TrustZone architecture incorporates two virtual Memory Management Units (MMUs) to perform access control of the DRAM, 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 an extension in TrustZone to protect I/O security, especially in environments where the integrity and confidentiality of I/O streaming data are

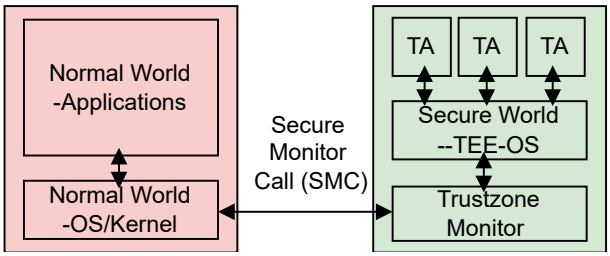


Figure 1: World switch in Arm Trustzone.

paramount. These mechanisms ensure that the I/O device can be only controlled by the secure world and all I/O data are isolated and protected from unauthorized access and manipulation. The Secure I/O is usually implemented by performing access control of I/O related data structure from the MMU.

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 TAs and the normal world. 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 our 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 I/O Work

Here we list some existing work about TrustZone secure I/O. An existing work regarding secure analysis of the data stream is called StreamBox TZ [23], which uses 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. There are some other secure I/O papers which try to protect I/O data from sensor, camera, or GUI, with the help of ARM TrustZone [8, 24, 26, 28, 30].

Notably, Park *et al.*, [22] studies how to enable normal world and secure world display at the same time. Unlike previous work which usually uses MMU to control the permission of the frame buffer [1, 9, 25], RushMore takes advantage of a hardware feature called an Image Processing Unit (IPU) to output to the video display via two layers. The first layer can be output by the normal world, and the secure world can use IPU to overwrite or cover the first layer by writing to the frame buffer owned by the IPU. By doing this, the normal world and the secure world can output to the display at the same time and the normal world won't be able to see what the secure world is currently displaying.

Trusted UI. Excepting secure I/O, how to enable trusted UI is a hot topic in TrustZone [3, 13, 14, 31]. One challenging part of Trusted UI is the latency due to the world switch between the

secure world and the normal world. Even though TrustZone offer the possibility of enabling Trust UI, the potential latency due to SMC call, secure I/O interrupt, and different UI driver make Trusted UI extremely difficult, especially considering nowadays display has higher resolutions.

3 OVERVIEW

In this section, we give an overview of our system design, as well as listing the major design challenges we need to solve.

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 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 [16]. 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*.

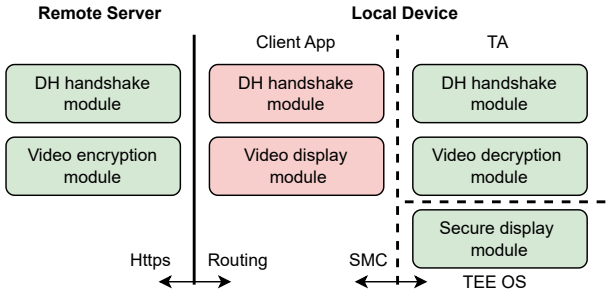


Figure 2: Major modules in SecDisplay.

3.2 Threat Model and System Challenges

Threat model. Our system's threat model aligns with the majority of systems designed around TrustZone technology. This implies that all programs running in the normal world are considered untrustworthy, while those operating in the secure world, including Trusted Applications (TAs) and the TEE Operating System (OS), are deemed trustworthy. It's important to note that Denial of Service (DoS) attacks [4, 15] are not covered by the TEE's threat model, as the normal world can always opt not to trigger the TA, thus bypassing the necessary security operations. Similarly, attacks based on physical access, such as intercepting the motherboard bus or physically monitoring data sent to the display [10, 29], are also excluded from our threat model. Moreover, side channel attacks, such as power-based side channel attacks [2, 33] and timing side channel attacks [11, 12], are also excluded.

System challenges. The primary challenge in designing this system is to ensure end-to-end security. This includes (1) network link security, which entails establishing network connections within the OP-TEE environment and implementing trustworthy network communications between the remote server and the TA, even when the client application is untrusted. (2) I/O data security. The system must handle video data and display in a secure way. This involves decrypting videos within the OP-TEE environment and securely displaying the video content on the video display via the secure world.

4 IMPLEMENTATION DETAILS

In this section, we focus on our implementation details about the three major components and the code modules we have developed, as illustrated in Figure 2.

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

4.1.1 DH handshake module. 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. 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.

4.1.2 Video encryption module. 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. Since the entire server platform is considered as trustworthy and is running in a linux system. We are able to reuse common crypto libraries, such as OpenSSL, to encrypt the video using AES encryption APIs.

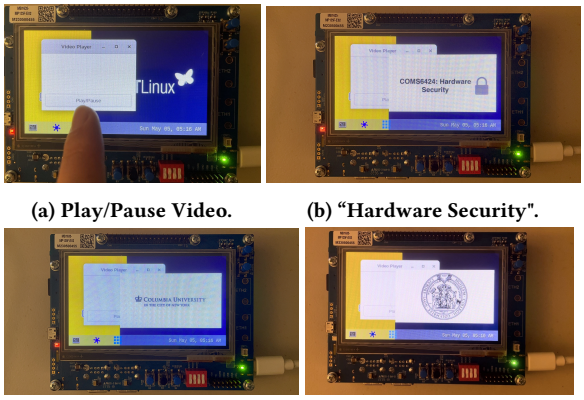
4.2 Client Application

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.

4.2.1 DH handshake/Key exchange module. The client application is an unstrusted application and is in charge of routing and forwarding network packages between the TA and the remote server. Except the first few network packages during DH handshake period is unencrypted network package, the rest of network transmission between the remote server and the TA is all encrypted using a network session key. Similarly, the client applciation also needs to forward the network packages that contain the encrypted video to the TA via SMC call. 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.

Note that in our implementation, the client program can will need to forward the TA's RSA public key to the remote server. This means our current client logic include that code that the client program used to retrieve 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. This is only for demo purposes. In a real-world secure display application, the remote server needs to know the public key of the TA in a secure way. Otherwise, the untrustworthy client application can pretend to be the TA and establish a fake connection with the remote server in order to download and get the protected video stored in the remote server.



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

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

Figure 3: Video display Demo in normal world.

4.2.2 Unprotected Video Display Module. STM32MP135F-DK is a physical board that supports installing the OP-TEE, 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 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

```
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.

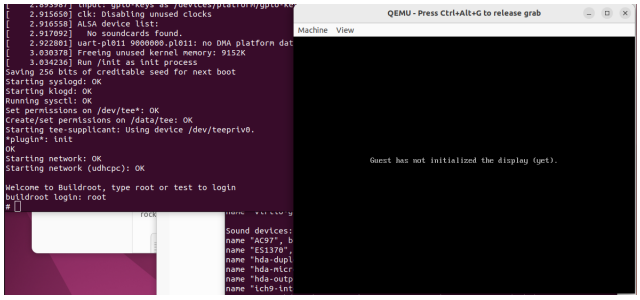


Figure 6: Appending a QEMU-emulated display to OP-TEE QEMU implementation.

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` 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, AOSP is only supported in Hikey development board, which is an old development board and is now unavailable online. Other development boards, such as STM32 series and raspberry Pi 3, are not supported[1]. Many GitHub issues also points out that it is too heavy to merge AOSP and OP-TEE together [17].

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 end [18, 20, 21]. 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.** We then explore how to enable secure display in STM32MP135F-DK Development Board. The STM32MP135F-DK has a GUI-supported OS from STMicroelectronics, known as the OpenSTLinux distribution, which is based on the OpenEmbedded/Yocto Project. Because it already supporting GUI and has relevant display driver, we thought it could have higher chance to be able to configure display.

After some exploring, we found STM32MP135F-DK can use a Device Tree Source (DTS) to protect the frame buffer. The frame buffer used by the secure world 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). We also found related code in their source code. 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.

Then the secure display module can be implemented as illustrated in Figure 7. Specifically, two virtual frame buffers will be maintained. One is for the normal world, and the other is for the

[1]"X is not enabled, and we have not built nor enabled any drivers for graphics" [19]

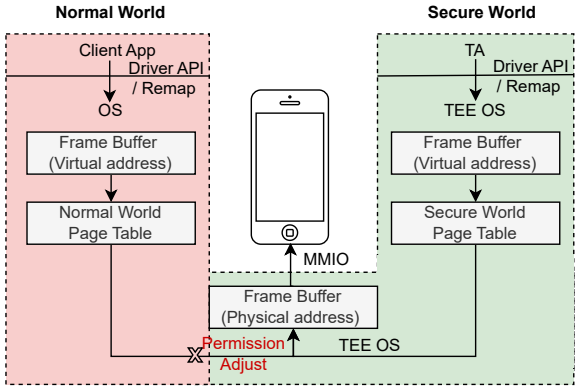


Figure 7: Memory control in SecDisplay.

```
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.

secure world. During the boot time, the TEE OS needs to set the permission of the physical frame buffer area using DTS and TZC technologies. Then during runtime, the TA can uses related interfaces provided by the TEE OS to ask TEE OS re-configure the permission of the physical frame buffer addresses. The physical frame buffer needs to be configured correctly such that when the TA is writing to the virtual frame buffer (displaying the protected video), the normal world cannot access the physical frame buffer areas. Need to note that since all operations here involves the data transmission between the I/O device and the DRAM, we will need to flush the cache line whenever some data is written to the frame buffer. The related APIs can be found using the `TEE_CacheFlush` function from `tee_internal_api_extensions.h`

However, due to time limit, we aren't able to debug and enable secure display in the STM32MP135F-DK development board. We will need to implement a kernel module running in the TEE OS, such that we will be able to adjust the permission of each physical pages.

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

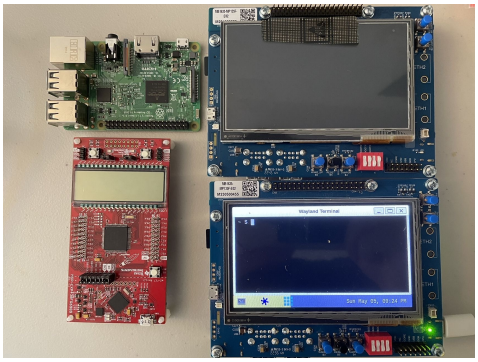


Figure 9: Development boards we have tried.

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.

5 EVALUATION

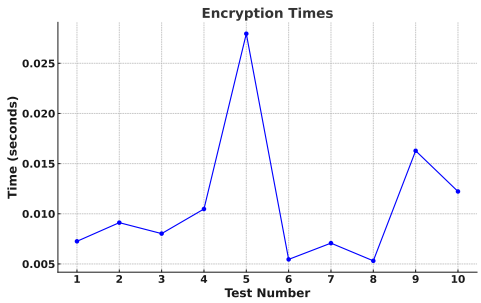
A time evaluation of encrypting and decrypting a string is done to help with understanding the estimated time of video encryption and decryption. The encryption and decryption procedures are done by a TA running inside the secure world. As shown in Figure 10(a), each point represents an encryption time finished by the *aes_ta.c* file; and in Figure 10(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.

For example, the major latency could be due to the world-switch between the normal world and the secure world. So how to choose the correct video segment size might be essential in improving the overall speed of decrypting the video. An proper video segment size could be helpful in balancing the limited memory size of the secure world and reducing the overhead due to the world switch time. Meanwhile, different AES decryption block size (*e.g.*, 128-bit, 192-bit, or 256-bit) and encryption modes (*e.g.*, AES block encryption mode, such as AES-ECB mode and AES-CTR mode, or Chaining mode) could also be key factors here.

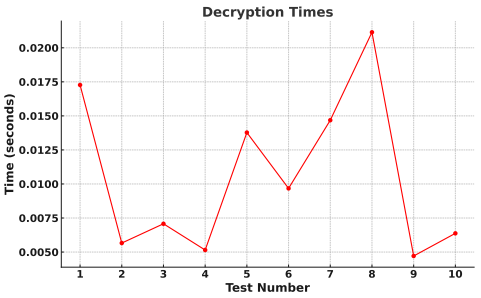
6 DISCUSSION

When developing this system, we realized that physical attacks and side channel attacks could be the main threats to secure display.

The threat from physical attacks might manifest in two main ways. The first is the connection between the display monitor and the ARM CPU. The monitor and CPU are connected via the motherboard and various cables and ports. Although data is well protected



(a) Encryption Time.



(b) Decryption Time.

Figure 10: Encryption and decryption time for ten times.

within the ARM CPU by the access control from TrustZone MMU, once it leaves the CPU, it is no longer protected on the motherboard. Attackers could potentially intercept the video data stream output to the display using an interposer, thus eavesdropping on protected videos. The second aspect is the possibility of recording videos using additional devices. Similar to how we use Snapchat, even though screenshot functionality is disabled, people can still record the screen using external devices.

Side channel attacks also pose a potential threat. We noticed that videos differ significantly in frame rates and colors, which could lead to variations in energy consumption or system reaction times. Attackers might use this data to deduce or fingerprint the video being displayed securely.

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] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and Tong Tong Wu. 2016. I-pic: A platform for privacy-compliant image capture. In *Proceedings of the 14th annual international conference on mobile systems, applications, and services*. 235–248.

[2] Sebanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. 2018. How TrustZone could be bypassed: Side-channel attacks on a modern system-on-chip. In *Information Security Theory and Practice: 11th IFIP WG 11.2 International Conference, WISTP 2017, Heraklion, Crete, Greece, September 28–29, 2017, Proceedings 11*. Springer, 93–109.

[3] Yang Cai, Yuewu Wang, Lingguang Lei, Quan Zhou, and Jun Li. 2019. SuiT: secure user interface based on TrustZone. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[4] Ahmet Cetinkaya, Hideaki Ishii, and Tomohisa Hayakawa. 2019. An overview on denial-of-service attacks in control systems: Attack models and security analyses. *Entropy* 21, 2 (2019), 210.

[5] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel tdx demystified: A top-down approach. *Comput. Surveys* (2024).

[6] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[7] Anthony CJ Fox, Gareth Stockwell, Shale Xiong, Hanno Becker, Dominic P Mulligan, Gustavo Petri, and Nathan Chong. 2023. A Verification Methodology for the Arm® Confidential Computing Architecture: From a Secure Specification to Safe Implementations. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 376–405.

[8] Daniel Hein, Johannes Winter, and Andreas Fitzek. 2015. Secure Block Device–Secure, Flexible, and Efficient Data Storage for ARM TrustZone Systems. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 222–229.

[9] Suman Jana, David Molnar, Alexander Moshchuk, Alan Dunn, Benjamin Livshits, Helen J Wang, and Eyal Ofek. 2013. Enabling Fine-Grained permissions for augmented reality applications with recognizers. In *22nd USENIX Security Symposium (USENIX Security 13)*. 415–430.

[10] Mohamed Amine Khelif, Jordane Lorandel, Olivier Romain, Matthieu Regnery, Denis Baheux, and Guillaume Barbu. 2020. Toward a hardware man-in-the-middle attack on pcie bus. *Microprocessors and Microsystems* 77 (2020), 103198.

[11] Zili Kou, Wenjian He, Sharad Sinha, and Wei Zhang. 2021. Load-step: A precise trustzone execution control framework for exploring new side-channel attacks like flush+ evict. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 979–984.

[12] Ben Lapid and Avishai Wool. 2019. Cache-attacks on the ARM TrustZone implementations of AES-256 and AES-256-GCM via GPU-based analysis. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 235–256.

[13] Wenhao Li, Shiyu Luo, Zhichuang Sun, Yubin Xia, Long Lu, Haibo Chen, Binyu Zang, and Haibing Guan. 2018. Vbutton: Practical attestation of user-driven operations in mobile apps. In *Proceedings of the 16th annual international conference on mobile systems, applications, and services*. 28–40.

[14] Dongtao Liu and Landon P Cox. 2014. Veriui: Attested login for mobile devices. In *Proceedings of the 15th workshop on mobile computing systems and applications*. 1–6.

[15] Neil Long and Rob Thomas. 2001. Trends in denial of service attack technology. *CERT Coordination Center* 648, 651 (2001), 569.

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

[17] OP-TEE. 2024. GitHub Issue: Test optee on raspberry pi; How to project on LCD monitor? https://github.com/OP-TEE/optee_os/issues/3606. (2024).

[18] OP-TEE. 2024. How to write new QEMU bios.bin to boot android platform? https://github.com/OP-TEE/optee_os/issues/736. (2024).

[19] OP-TEE. 2024. OP-TEE: Raspberry Pi 3. https://optee.readthedocs.io/en/latest/building/devices/rpi3.html. (2024).

[20] OP-TEE. 2024. OPTEE with Android on QEMU. https://github.com/OP-TEE/optee_os/issues/605. (2024).

[21] OP-TEE. 2024. Run optee_os with android on QEMU. https://github.com/OP-TEE/optee_os/issues/715. (2024).

[22] Chang Min Park, Donghwi Kim, Deepesh Veersen Sidhwani, Andrew Fuchs, Arnob Paul, Sung-Ju Lee, Karthik Dantu, and Steven Y Ko. 2021. Rushmore: securely displaying static and animated images using TrustZone. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 122–135.

[23] 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.

[24] Sandro Pinto, Pedro Machado, Daniel Oliveira, David Cerdeira, and Tiago Gomes. 2021. Self-secured devices: High performance and secure I/O access in TrustZone-based systems. *Journal of Systems Architecture* 119 (2021), 102238.

[25] Nisarg Raval, Animesh Srivastava, Ali Razeen, Kiron Lebeck, Ashwin Machanavajjhala, and Lanodn P Cox. 2016. What you mark is what apps see. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. 249–261.

[26] Takayuki Sasaki, Koki Tomita, Yuto Hayaki, Seng Pei Liew, and Norio Yamagaki. 2020. Secure IoT device architecture using TrustZone. In *2020 IEEE International Conference on Sensing, Communication and Networking (SECON Workshops)*. IEEE, 1–6.

[27] AMD Sev-Snp. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* 53 (2020), 1450–1465.

[28] Chad Spensky, Aravind Machiry, Marcel Busch, Kevin Leach, Rick Housley, Christopher Kruegel, and Giovanni Vigna. 2020. TRUST. IO: protecting physical interfaces on cyber-physical systems. In *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 1–9.

[29] Zhenyu Xu, Thomas Mauldin, Qing Yang, and Tao Wei. 2021. Runtime detection of probing/tampering on interconnecting buses. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 247–251.

[30] Kailiang Ying, Amit Ahlawat, Bilal Alsharifi, Yuexin Jiang, Priyank Thavai, and Wenliang Du. 2018. Truz-droid: Integrating trustzone with mobile operating system. In *Proceedings of the 16th annual international conference on mobile systems, applications, and services*. 14–27.

[31] Kailiang Ying, Priyank Thavai, and Wenliang Du. 2019. Truz-view: Developing trustzone user interface for mobile os using delegation integration model. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*. 1–12.

[32] Yiming Zhang, Yuxin Hu, Zhenyu Ning, Fengwei Zhang, Xiapu Luo, Haoyang Huang, Shoumeng Yan, and Zhengyu He. 2023. SHELTER: Extending Arm CCA with Isolation in User Space. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6257–6274.

[33] Mark Zhao and G Edward Suh. 2018. FPGA-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 229–244.