OCRAM-assisted Sensitive Data Protection on ARM-based Platform

Dawei Chu[1,2,3], Yuewu Wang[1,2,3], Lingguang Lei[1,2,3] ⋆, Yanchu Li[1,2,3], Jiwu Jing[4], and Kun Sun[5]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
[2] Data Assurance and Communication Security Research Center, Chinese Academy of Sciences
[3] School of Cyber Security, University of Chinese Academy of Sciences
[4] School of Computer Science and Technology, University of Chinese Academy of Sciences
[5] Center for Secure Information Systems, George Mason University

Abstract. On mobile devices, security-sensitive tasks (e.g., mobile payment, one-time password) involve not only sensitive data such as cryptographic keying material, but also sensitive I/O operations such as inputting PIN code via touchscreen and showing the authentication verification code on the display. Therefore, a comprehensive protection of these services should enforce a Trusted User Interface (TUI) to protect the sensitive user inputs and system outputs, in addition to preventing both software attacks and physical memory disclosure attacks. In this paper, we present an On-Chip RAM (OCRAM) assisted sensitive data protection mechanism named Oath on ARM-based platform to protect the sensitive data, particularly, sensitive I/O data, against both software attacks and physical memory disclosure attacks. The basic idea is to store and process the sensitive data in the OCRAM that is only accessible to the TrustZone secure world. After figuring out how to enable TrustZone protection for iRAM, we develop a trusted user interface with an OCRAM allocation mechanism to efficiently share the OCRAM between the secure OS and the rich OS. A prototype implemented on the OP-TEE system shows that Oath works well and has a small system overhead.

Keywords: OCRAM(iRAM) · TrustZone · Cold Boot Attacks · Trusted User Interface

1 Introduction

To facilitate security-sensitive tasks such as mobile payment, mobile wallet, etc., mobile terminal vendors have integrated hardware-assisted Trusted Execution Environment (TEE) [3] features in their devices. For instance, SAMSUNG utilizes ARM TrustZone technology [12] to secure SAMSUNG PAY [8], FIDO relies on TEE technology to protect the authentication procedure [20], and Google adopts TEE to implement a secure cryptographic service for Android apps [2]. TEE takes advantage of hardware isolation mechanism to defeat various software attacks. As one hardware TEE solution, the ARM TrustZone security extension [12] can isolate the system resources including CPU, memory, and peripherals between two execution environments, namely, the *secure world* and the *normal world* (or *non-secure world*). Thus, the TEE-based solutions [23, 31, 34, 36, 38, 47, 50, 51, 57] can effectively protect various secure software running in the secure world from the untrusted rich OS running in the normal world.

Though the TEE-based solutions can effectively defeat software attacks, they suffer from physical memory disclosure attacks such as cold boot attacks [32, 43] and bus snooping attacks [28], since the sensitive data are stored in DRAM as plaintext. Therefore, with physical access to the stolen or lost mobile devices, the adversary may have opportunity to obtain sensitive data such as credit card numbers and user passwords from the system DRAM. To mitigate these attacks, several solutions have been proposed by constraining the storing and processing of sensitive data within the on-chip storage, such as CPU registers [44,48], CPU cache [19,30,56], and on-chip RAM (OCRAM, also known as internal RAM (iRAM)) [19]. The on-chip storage is naturally immune to the bus snooping attacks, since the data needs not to be transmitted from the CPU to the DRAM. Also, due to a smaller data remanence rate than DRAM [19], the on-chip storage can provide a

⋆ Corresponding Author

better protection against cold boot attacks. However, it remains as a challenge to provide a comprehensive protection on both sensitive data such as cryptographic keys and sensitive I/O operations such as inputting PIN code via touchscreen or showing the authentication verification code on the display. Most on-chip storage based solutions do not resolve the software attacks [19, 19, 30, 44, 48] or are not able to protect the I/O operations [56].

In this paper, we provide an on-chip storage based security solution that could protect the sensitive data including sensitive I/O data against both software attacks and physical memory disclosure attacks. We develop an OCRAM assisted mechanism named Oath, whose basic idea is to protect the sensitive data by processing it in the TrustZone secure world and further locking the sensitive data in the OCRAM (i.e., iRAM). Because the sensitive data is only processed in the secure world, it can prevent the software attacks from the normal world. Moreover, since the secure sensitive data only resides in OCRAM, it can prevent physical memory disclosure attacks.

We choose to use OCRAM for two major reasons. First, different from cache, OCRAM can be addressed the same manner as DRAM. Thus, it can be used to support the TUI operations that rely heavily on DMA mechanism to directly access the I/O devices. For instance, it is difficult for the Image Processing Unit (IPU) to address and read the display framebuffer in cache and then send it to the external display device [56]. Second, OCRAM has been widely used in ARM processors, and the size of OCRAM is usually larger than the available number of registers that are basically too small to accommodate the associated sensitive data. For example, modern high-end processors typically have from 128 KB to 2 MB OCRAM memory [1, 4, 9, 10, 25, 46, 52, 56]. In the following, we use OCRAM and iRAM interchangeably.

We solve three major challenges on developing Oath. First, to defeat the software attacks from malicious rich OS, the iRAM region containing sensitive data should be isolated from the normal world. However, iRAM is by default used by rich OS and set as non-secure, and there lacks available public documentations on enabling the TrustZone protection for iRAM. We have to figure it out by trial and error on a real hardware platform. Second, we observe that iRAM has been used by rich OS in situations such as system sleeping/resuming, video playing, etc., and it may impose negative function and performance impacts on rich OS when we use iRAM to store sensitive data. To minimize the impacts on rich OS, we develop a dynamic iRAM allocation mechanism to adjust the size of the secure iRAM region on demand and return the unused iRAM memory to rich OS. Third, due to the small size of available iRAM (e.g., the Cortex A9 ARM processors usually accommodate 256 KB iRAM [4, 10, 46]), it is difficult to store multiple secure applications into iRAM. To solve this problem, we develop a memory splitting mechanism that can partition the application's data into sensitive part and nonsensitive part, and only store the sensitive data in the secure iRAM. In addition, current mobile phones are usually equipped with high-resolution screen, which can easily demand more than 1 MB memory for displaying an image with minimum pixel format. We solve it by enabling the dual display mode and displaying the sensitive data as an size-adjustable foreground image on a portion of the screen.

We implement a system prototype of Oath on the i.MX6Quad platform, with an OP-TEE OS 2.2.0 [6] ported to the secure world and an Android OS 6.0.1 (Linux kernel 4.1.15) ported as rich OS in the normal world. The experimental results show that our defense mechanism incurs a small overhead over both OP-TEE OS and Android OS.

In summary, we make the following contributions in this paper.

- We develop an iRAM-assisted sensitive data protection solution that can effectively defeat both software attacks and physical memory disclosure attacks. It also provides a trusted path to protect the user's I/O interactions with the mobile devices.
- We develop a memory splitting mechanism and a dynamic iRAM allocation mechanism to improve the usage efficiency of the small-capacity iRAM, so that it can better serve both the secure OS and the rich OS. We provide detail guidance on how to use TrustZone to protect iRAM, which can promote the usage of iRAM in the TrustZone.
- We develop a system prototype on the OP-TEE OS system, which is compliant with the GlobalPlatform TEE specifications [3] and is compatible with many other hardware platforms.

The remaining of the paper is organized as follows. Section 2 introduces necessary background knowledge. Section 3 describes the threat model and assumptions. Section 4 presents an overview of system design. A prototype implementation is detailed in Section 5. Section 6 discusses the

evaluation of our work. Discussion and future work are illustrated in Section 7. We describe related works in Section 8. Finally, we conclude the paper in Section 9.

2 Background

We first introduce the ARM TrustZone hardware security extension. Then we discuss the internal Random Access Memory (iRAM) in ARM processors and the protection of iRAM with TrustZone technology. Next, We introduce the Open Portable Trusted Execution Environment (OP-TEE) system.

2.1 ARM TrustZone

TrustZone is a hardware security extension since ARMv6 architecture to provide a complete isolation environment for secure code execution. The security is achieved by partitioning the resources of the ARM System-on-a-Chip (SoC) including processor, memory, and peripherals, so that they can exist in one of two worlds - the *secure world* and the *normal world*. The normal world cannot access the secure world's resources such as memory, I/O devices etc., while the secure world can access the resources of both worlds. The normal world is usually used to run a commodity OS (also referred to as rich OS), while the secure world is used to run an isolated secure OS. Switching of the two worlds is supervised by a *Secure Monitor* running in monitor mode. The entry to monitor mode can be triggered by software executing a dedicated instruction, i.e., the Secure Monitor Call (SMC) instruction, or by emitting a secure interrupt request. Normally, Interrupt Request (IRQ) is configured as a normal world interrupt and Fast Interrupt Request (FIQ) is configured as a secure world interrupt. When a secure interrupt arrives, TrustZone switches the processor core to the monitor mode in order to handle it.

2.2 iRAM in ARM processors

iRAM is an On-Chip Random Access Memory (OCRAM) constructed with a fast and expensive Static Random Access Memory (SRAM). iRAM is a common component on the ARM processors usually used in two situations. First, it is used to store the information such as system suspend/resume codes, DDR frequency modification codes etc. Second, it could be used to accelerate the multimedia processing, such as video playing. Modern high-end processors typically have 128 KB to 2 MB iRAM memory [1,4,9,10,25,46,52,56], for example, the Cortex A8 and A9 ARM processors usually have 128 KB and 256 KB iRAM, respectively [4, 10, 25, 46, 56].

iRAM can be protected through two on-SoC components, i.e., *TrustZone Memory Adapter (TZMA)* and *TrustZone Protection Controller (TZPC)* [13]. The TZMA is designed to secure a region within an on-SoC static memory such as a SRAM, and it only supports the partition of one secure region. The TZPC is a configurable signal control block to dynamically control the security states of on-SoC components, such as iRAM, TZMA etc. Theoretically, protection of iRAM could be achieved in two steps. In the first step, we can utilize TZMA to enable TrustZone protection on iRAM and partition it into two regions, i.e., secure region and non-secure region. For convenience, we refer to the secure region as secure iRAM and to the non-secure region as non-secure iRAM. However, normal world codes can still have the access to the secure iRAM. In the second step, we can set the permission of iRAM as *secure access only* through TZPC. In addition, TZPC can be used to dynamically change the size of secure iRAM at run-time by sending signals to TZMA.

2.3 OP-TEE System

GlobalPlatform TEE specification [3] has been proposed to facilitate the adaptation of TrustZone security features by providing common APIs to be called in mobile apps. OP-TEE is an open source TEE system that is compliant with the GlobalPlatform TEE specification and compatible with many hardware platforms [7].

OP-TEE is a typical TEE implementation based on the ARM TrustZone technology. Figure 1 shows the architecture of OP-TEE, which consists of three components, i.e., *TEE Client*, *TEE*

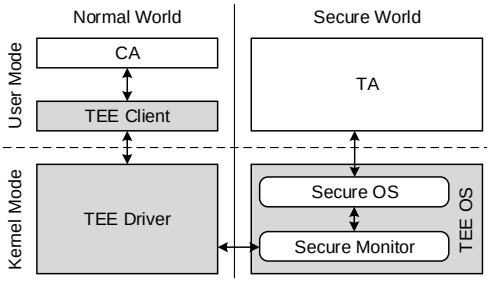


Fig. 1. Architecture of OP-TEE

Driver, and *TEE OS*. The TEE client and the TEE driver are executed in the normal world, and the TEE OS is executed in the secure world. The TEE client executes in the user mode and provides APIs for Client Apps (CAs) in the normal world, and it communicates with the TEE driver running in the kernel mode. The TEE driver is in charge of the communication between TEE client and TEE OS, e.g., by executing the SMC privileged instruction. The TEE OS is composed of a Secure Monitor and a Secure OS. The secure monitor is responsible for context switching between the normal world and the secure world, and the secure OS communicates with Trusted Apps (TAs) according to the requests from the TEE client. OP-TEE supports two types of TAs, i.e., the static TA and the dynamic TA. The former is statically compiled and loaded along with TEE OS image when the secure OS boots up, while the later could be dynamically loaded from file system at run-time.

3 Threat Model And Assumptions

We assume the attackers can launch sophisticated multi-vector attacks, including software attacks and physical memory disclosure attacks. For software attacks, the attackers can compromise the rich OS, thus gaining unrestricted access to not only DRAM but also on-chip memory (e.g., CPU cache, registers, OCRAM etc.) in the normal world. However, the attackers cannot access DRAM and on-chip memory in the secure world due to the protection of TrustZone. The attackers may also launch physical memory disclosure attacks, such as cold boot attacks [32, 43], to steal the sensitive code and data in DRAM. The attackers can also snoop the data transferred on the bus between CPU and DRAM [28]. However, we assume the attackers could not physically access the data on the on-chip memory. Side-channel attacks such as timing-based and power-based analysis are out of the scope of this paper.

We assume the ARM-based platform supports the TrustZone security extension, and the hardware implementations of TrustZone are correct and can be trustworthy. We also assume the TEE OS including the secure OS and the secure monitor in the secure world is trusted and it can boot up securely via the *secure boot* technology of TrustZone. In this work, we focus on protecting the confidentiality and integrity of the sensitive data, such as the cryptographic key used to perform encryption and decryption, the authentication verification code shown on the display devices, and the PIN code input by the user via touchscreen. We assume the TA running in the secure world can be trusted and will not leak sensitive data deliberately to the rich OS. Moreover, since the TA image is stored in the file system of normal world without encryption, we assume no sensitive data is statically stored in the TA binary image.

4 System Design

Our goal is to construct a TEE system that can ensure sensitive data (including sensitive I/O data) against both software attacks and physical memory disclosure attacks. The basic idea of Oath is to ensure that the sensitive data is only stored in the secure iRAM and processed by the secure CPU cores, so that it can protect sensitive data against software attacks from the rich OS and the physical memory disclosure attacks. Our design is based on the OP-TEE system to be compliant with the GlobalPlatform TEE specifications [3]. In general, Oath is composed of four modules

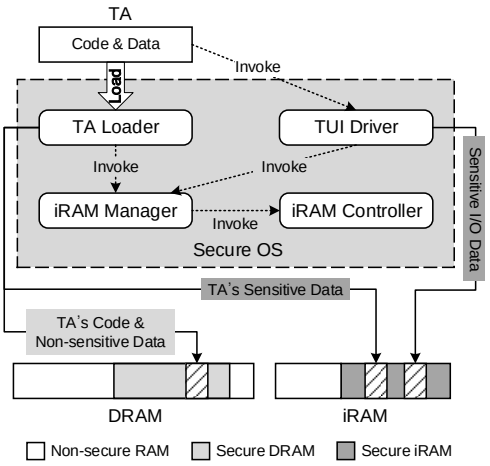


Fig. 2. Architecture of Oath

implemented in the secure OS of OP-TEE, as shown in Figure 2. One module is implemented by modifying the *TA Loader* of the secure OS, and the other three modules, i.e., *TUI Driver*, *iRAM Manager*, and *iRAM Controller*, are newly added into the secure OS. In the following, we provide a description of each module and present how they work together to achieve the design goals.

TA Loader. In the OP-TEE system, the TA loader is responsible to load a TA into memory and allocate the dynamic memory like stack and heap for it. Normally, the memory will be allocated from secure DRAM. To defend against the physical memory disclosure attacks, we modify the TA loader to store the TA in the secure iRAM. However, the small-capacity iRAM memory makes it difficult to support the simultaneous execution of multiple TAs, so we develop a memory splitting mechanism in the TA loader. Specifically, the TA's memory is split into an sensitive part (i.e., memory containing sensitive data) and a nonsensitive part, and only sensitive part is stored in secure iRAM. The splitting mechanism is achieved based on the observation that the TA image is originally stored in the unprotected file system, so it becomes a nature choice for the developers to store nonsensitive data in the constant parts of the TA image, e.g., the .text and .rodata sections of the ELF file. As such, Oath counts the memory containing the constant data as nonsensitive. To further optimize the utilization efficiency of iRAM, we enable the TA loader to selectively load sensitive data of specific TAs into the secure iRAM.

TUI Driver. The TUI driver is responsible for supporting trusted user interactions, e.g., trusted display and trusted input. To ensure security, the TUI driver stores the sensitive I/O data only in the secure iRAM. However, the sensitive data is sometimes too large to be stored in the small-capacity iRAM memory, such as the large-sized sensitive image to be securely displayed on the screen. The TUI driver enables trusted display based on the observation that most platforms support dual display mode, i.e., blending a foreground image and background image to form the final image. Although displaying an image on the entire screen needs a large memory, the size of foreground image is adjustable and could be smaller. Therefore, when receiving a trusted display request, the TUI driver enables the dual display mode and displays the sensitive data as a foreground image on only a portion of the screen.

iRAM Manager. The iRAM manager module manages the secure iRAM for multiple TAs by allocating and freeing memory spaces from the secure iRAM. It is invoked by the TA loader when loading/unloading a TA and by the TUI driver when performing trusted user interactions.

iRAM Controller. The function of the iRAM controller is two-fold. First, it enables the TrustZone protection for iRAM when the TEE OS boots up. Second, to reduce the impacts on the performance and function of rich OS, it contains a dynamic iRAM allocation mechanism to minimize the size of secure iRAM at run time. Originally, most iRAM memory is set as non-secure. During system execution, the iRAM controller will be invoked by the iRAM manager to enlarge or shrink the secure iRAM region. The sensitive data on the secure iRAM region will be wiped out before it is set as non-secure and accessed by the rich OS.

5 Implementation

We implement an Oath prototype on the FreeScale i.MX6Quad platform with an OP-TEE 2.2.0 OS system in the secure world and an Android 6.0.1 system deployed in normal world. The Android OS with Linux kernel 4.1.15 is ported from the secure world to the normal world.

5.1 iRAM Controller

In general, the iRAM controller performs two tasks, namely, enabling the TrustZone protection for iRAM and dynamically adjusting the size of secure iRAM. In Section 2.2, we briefly describe how to control the TrustZone protection on iRAM through two on-SoC components, i.e., TZMA and TZPC. On the i.MX6Quad platform, the function of TZMA is integrated into an on-SoC multiplex controller called I/O Multiplexer Control (IOMUXC), which is shared by multiple I/O devices. The function of TZPC is implemented through an on-SoC component called Central Security Unit (CSU). CSU can only be set by the secure world, which sets individual security access privileges on each of the peripherals. In the following, we present the implementation details of the iRAM controller.

Enabling TrustZone Protection for iRAM. The IOMUXC controller on the i.MX6Quad platform consists of a few registers, and the TrustZone protection for iRAM is controlled through the General Purpose Register 10 (GPR10). Specifically, OCRAM_TZ_ADDR and OCRAM_TZ_EN fields on GPR10 define the start address of the secure iRAM and if TrustZone protection on iRAM is enabled, respectively. The end address of secure iRAM is not configurable, and its value is the highest address of the iRAM. By default, IOMUXC is a non-secure peripheral that could be read/written by rich OS in the normal world. Therefore, two other fields are provided in GPR10, i.e., LOCK_OCRAM_TZ_ADDR and LOCK_OCRAM_TZ_EN, which ensure the values of OCRAM_TZ_ADDR and OCRAM_TZ_EN fields cannot be modified until reset.

Besides the above configuration through IOMUXC, we need to set the access privilege of the iRAM as secure through the central security unit (CSU). In total, there are 40 Config Security Level (CSL) registers on CSU. Each CSL consists of two fields that determine the access permissions for two peripherals. For example, the first field of CSL6 and CSL27 corresponds to IOMUXC and HDMI, respectively. When the permission of a peripheral is set as secure, it could only be accessed when the CPU works in the secure mode.

Since the reference manual of i.MX6 families does not provide clear instructions on what CSL controls the access permission of the iRAM, we perform experiments to figure it out. First, we set the access permissions of all peripherals as secure, except the Universal Asynchronous Receiver/-Transmitter (UART1) used to print the log in the normal world. Next we access the secure iRAM from the normal world, and find the secure iRAM could not be read or written. It means iRAM is indeed protected by one of the 40 CSLs. Then we identify the corresponding CSL using a binary search algorithm to narrow down the protected peripherals in half each time. Finally, we find that the second field of CSL26 corresponds to iRAM, while it is marked as reserved in the reference manual for i.MX6 families.

Adjusting the Size of Secure iRAM. Since we can lock the OCRAM_TZ_ADDR and OCRAM_TZ_EN fields, the sensitive data in secure iRAM could not be attacked by the rich OS at run time via either disabling TrustZone protection on iRAM or shrinking the size of secure iRAM. However, it also constrains the dynamic adjustment on the size of secure iRAM, and it may affect efficient utilization of the small-capacity iRAM. When allocating a fixed large size of secure iRAM, it will affect the rich OS on running some operations such as multimedia processing; when allocating a fixed small size of secure iRAM, it will affect the execution of multiple TAs and the trusted display. Therefore, it is necessary for Oath to be able to dynamically adjust the size of secure iRAM. To accomplish this, we achieve security through setting the access permission of IOMUXC as secure, rather than locking the start address of secure iRAM. Thus, the size of secure iRAM can be dynamically adjusted by the secure OS only.

However, since setting IOMUXC as secure peripheral will fail the reading and writing of IOMUXC in rich OS. we must port these operations from the normal world to the secure world. It is not an easy task, since IOMUXC is a multiplexer controller shared by multiple I/O devices, such as HDMI, ENET, USB, UART, SD, etc. The codes to read or write IOMUXC scatter among the

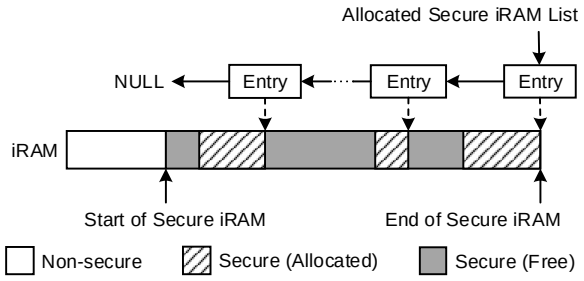


Fig. 3. Management of the Secure iRAM

rich OS, such as initializing the devices during system boots up. After analyzing the source code, we port the related codes as follows. In general, the registers of IOMUXC could be divided into two types, i.e., device dedicated registers vs. general purpose registers. Access of the former ones is concentrated through the device tree mechanism, so they could be ported uniformly. Though the operations on the later ones are scattered, we can identify them via keyword searching such as GPR, MUX, etc.

In total, 102 IOMUXC operations are ported. Specifically, we substitute the reading and writing of IOMUXC in rich OS with two methods *secure_readl* and *secure_writel*, whose pseudo codes are depicted in Listing 1.1. When being invoked, they first switch to the secure world, perform the read/write operations, and then switch back to the normal world. Since these two methods may be abused by the attackers as a springboard into the secure world, we must check the input *regAddr* parameter to ensure that they can only be used to read or write portions of the IOMUXC registers that are not sensitive-critical. For example, writing to the OCRAM_TZ_ADDR and OCRAM_TZ_EN fields in GRP10 register will be blocked.

Listing 1.1. Secure Read And Write Methods

```
//return value of the register "regAddr"
secure_readl(regAddr);
//write register "regAddr" with value "val"
secure_writel(val, regAddr);
```

5.2 iRAM Manager

The iRAM manager is responsible to manage the secure iRAM. The detailed implementation is illustrated in Figure 3. The memory range between *Start of Secure iRAM* and *End of Secure iRAM* is the secure iRAM, while the address lower than *Start of Secure iRAM* is non-secure iRAM. The allocated memory is managed with a list named *Allocated Secure iRAM List*, and each entry in the list specifies an allocated memory slot, marked with slash in Figure 3. Free memory slots are marked gray. The allocation and release of memory is achieved by inserting and deleting an entry to and from the list, respectively. Basically, we adopt the simple but efficient first-fit algorithm [17] to search a free space for allocation. The searching starts from *End of Secure iRAM*, traverses the *Allocated Secure iRAM List*, and stops until finding the first free space whose size is equal or larger than the required size.

The iRAM manager is also responsible for invoking iRAM controller to dynamically adjust the size of secure iRAM region. To enhance the efficient usage of iRAM for both secure OS and rich OS, we need to first figure out the exact utilization of iRAM in rich OS. After investigating the i.MX6 application processor's reference manual [26] and the rich OS source codes, we identify that iRAM is used by rich OS in two main scenarios. First, the lowest 20 KB iRAM memory is statically allocated when rich OS boots up, to store the information such as system suspend/resume codes, DDR frequency modification codes, etc. Second, another 204 KB iRAM is allocated at run time when playing the videos via hardware decoder. The remaining 32 KB iRAM has not been used by rich OS. To make full use of the iRAM memory, the iRAM manager reserves the lowest 20 KB for rich OS and highest 32 KB for secure OS. To minimize the impacts on rich OS, the remaining 204

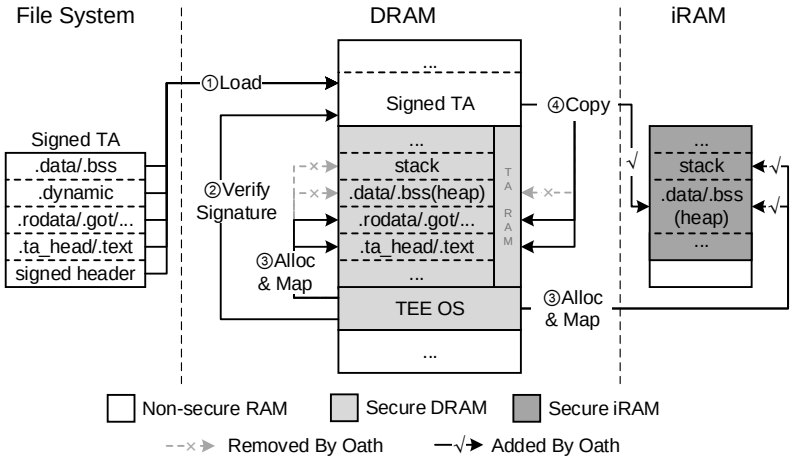


Fig. 4. Procedure to Load a TA

KB will be set as non-secure when the system boots up. When receiving a TA loading or trusted user interaction request, iRAM controller will set them as secure temporarily, and restore them as non-secure once the sensitive operations are done. In addition, before restoring the iRAM region as non-secure, the iRAM manager erases the iRAM data to prevent potential data leakage.

5.3 Memory Splitting in TA Loader

To support multi-TA execution, we develop a memory splitting mechanism to reduce the secure iRAM usage of each TA. The basic idea is to split the data of a TA into sensitive part and nonsensitive part, and allocate secure iRAM to store and process the sensitive data only. Before implementing the memory splitting mechanism, we first present the normal working flow of the TA loader.

Normal Working Flow of TA Loader. A dynamic TA will be loaded into memory through four steps, as depicted in Figure 4. Originally, the TA image is stored in the file system of rich OS, consisting of a signed header and an ELF file. When receiving a loading request, the entire image is firstly loaded to the non-secure memory by the TEE client, as shown in Figure 1. Then the TA loader reads the signed header and verifies the signature of the ELF file. If the verification passes, the TA loader allocates a stack memory from the TA RAM (a secure DRAM region dedicated for TAs) to store the segments in the ELF file marked as PT_LOAD, and then constructs the virtual to physical memory mapping for them. The size of the stack memory is specified in the .ta_head section of ELF file. Finally, the PT_LOAD segments are copied from the non-secure memory to the allocated TA RAM.

Memory Splitting in TA Loader. We first identify the sensitive memory that may contain sensitive data via two following analysis. First, we investigate the TA loading procedure of popular TEE systems including QSEE and OP-TEE, and find that the TA images are stored without encryption in the rich OS's file system. Due to the concerns on data confidentiality, the developers should not store sensitive data directly in the read-only segments, such as .text, .rodata, etc. Otherwise, iRAM or TrustZone can hardly protect them, since the attackers can directly analyze the static images. Second, this work is done cooperatively with a real smartphone vendor. With the access to its commercial TEE platform, we also discover the same sensitive data characteristics by analyzing their TAs, including the TAs used for trusted payment, social interaction, etc. Due to the concern of business confidentiality, related results are excluded from the paper.

In this paper, we assume the sensitive data will only appear in the writable memory, including the initialized or uninitialized global/static variables and the dynamically created stack and heap variables. Therefore, the sensitive sections including the .bss section storing heap variables and the uninitialized global and static variables, the .data section storing the initialized global and static variables, and the stack section are allocated from the secure iRAM memory. We implement a memory splitting mechanism by modifying the last two steps of the TA loader in Figure 4.

Specifically, the TA loader allocates secure iRAM for the sensitive sections including .bss, .data, and stack, with the help of the iRAM manager. The remaining sections are still allocated from the TA RAM. Accordingly, the virtual to physical memory mappings for the .bss, .data, and stack sections are modified. Moreover, the initialized data in the .data and .bss sections are copied to the allocated secure iRAM, while other sections remain unchanged.

5.4 TUI Driver

TUI operations should protect user interactions against the software attacks from compromised rich OS. In addition, we enhance the security of TUI operations to defend against the physical memory disclosure attacks. Basically, the TUI driver stores the sensitive data input and output in the secure iRAM. The current implementation includes two typical user interaction operations, i.e., trusted input and trusted display. The implementation of trusted input is straightforward, as the size of input data is usually small and could be easily stored in the secure iRAM. However, there are two challenges for implementing trusted display. The first one is on the contradiction between the large-size image and the small-capacity iRAM memory, and the other one is on enabling the access of secure iRAM from DMA controller that is needed for displaying. In the following, we introduce the detailed implementation of trusted input and trusted display.

Trusted Input. . Generally, it is achieved through the cooperation of a TA and the TUI driver in the secure world. The TA is responsible to provide a software PIN pad with numbers 0 - 9 on the touchscreen and derive the input number from the touch event. The TUI driver takes charge to collect the touch events securely and pass them to the TA. First, the trusted input operation is initiated by the user through an app (i.e., CA) in the normal world. Next, after the execution context is switched to the secure world, the TA displays the PIN pad. As such, once the screen is touched, an X-Y coordinate pair representing the position touched is stored into the Analog-to-Digital Converter (ADC) register. Then, the TUI driver can read the coordinate value through the Inter-Integrated Circuit (IIC) bus and store it in the secure iRAM. Finally, the parsing procedure in the TA can derive the corresponding number based on the coordinate value and location of each number. To ensure security, the TUI driver sets the touchscreen's interrupt as secure once the software PIN pad is displayed. When there is a touch on the screen, an interrupt arises in the secure domain and the coordinate value could be obtained in the interrupt handler. When the PIN pad is closed, the interrupt will be restored to non-secure, so that rich OS can use the touchscreen. Moreover, we utilize an on-board LED light (set as a secure peripheral) as an indicator when the TA's PIN pad is displayed, to prevent the malicious rich OS from displaying a fake PIN pad and launching phishing attacks [50].

Trusted Display. We resolve the two challenges for trusted display with two mechanisms. First, we enable dual display and display the sensitive data as a size-adjustable foreground image. Second, during a trusted display, we temporarily set the iRAM memory containing sensitive framebuffer as non-secure and pause the non-secure cores and non-related DMA operations. We resume them when the trusted display is done.

a) Configuring IPU for Trusted Display. The on-SoC Image Processing Unit (IPU) works as part of the video and graphics subsystem on most of i.MX products including i.MX6Quad. It reads the framebuffer data in memory, processes the data, and presents the final image on the display devices such as LCD. In our implementation, we first figure out how data is processed and displayed through IPU. As shown in Figure 5, IPU can work in single display mode and dual display mode, where the former displays only the background image while the later merges both the foreground and background images. In general, five components of IPU are involved to perform image display, including *Image DMA Controller (IDMAC)*, *Display Multi FIFO Controller (DMFC)*, *Display Processor (DP)*, *Display Controller (DC)*, and *Display Interface (DI)*.

As shown in Figure 5, IDMAC is in charge of controlling the memory port and transferring data from system memory. There are total 64 IDMAC channels for each IPU, in which channel 23 (CH23) and channel 27 (CH27) are responsible to transfer the background and foreground data, respectively. DMFC controls multiple FIFOs for the IDMAC channels related to the display system, which relays the data from IDMAC channels to DP. DP performs the processing required for data sent to a display, such as combining 2 graphics planes and adjusting the brightness, contrast, color saturation, etc. DP has two input FIFOs holding the data of full plane and partial plane,

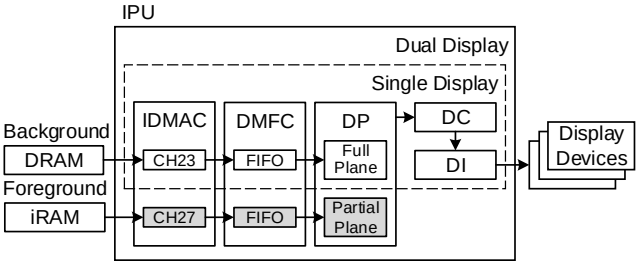


Fig. 5. Image Displaying Through IPU

where the former one corresponds to a fixed size background, while the later one corresponds to the foreground whose size is adjustable. DC controls the display ports and thereby specifies the interface to display the image. DI converts the data from the DC to a format suitable for the specific display interface.

Normally, the IPU works in the single display mode. Therefore, in the TUI driver, we switch the IPU to dual display mode, and display the sensitive data as a foreground image. As shown in Figure 5, three components, i.e., IDMAC, DMFC, and DP, need to be configured to enable dual display, and DC and DI can remain unchanged. To reduce the run-time computations and minimize the memory requirement, we utilize the Look-Up Table (LUT) [5] for the image displaying. With LUT, the Bits Per Pixel (BPP) could be reduced to 4 on our i.MX6Quad platform. It means we can display the sensitive data as a foreground image on the sharpness screen with resolution up to 1920*1080 pixels, and it can cover 1/6 screen with only 168.75 KB memory required.

Listing 1.2. Configuring IPU for Trusted Display

```
//width, height: foreground width and height
//xp, yp: x and y position of foreground
//trans: transparency of foreground
ConfigIPUForTrustedDisplay(width,height,xp,yp,trans) {
    //Preparation
    OriginalIPUContextSave();
    LookUpTableCreate();
    //Configuration
    Fg_IDMAC_Config(FrameBufAddr, width, height, LUTMode);
    DMFC_Config(FgCH27);
    DP_Config(xp, yp, trans);
    WaitForStop();
    //Restoration
    OriginalIPUContextRestore();
}
```

The details on configuring the IPU for sensitive data displaying is illustrated in Listing 1.2. Originally, the rich OS is displaying the User Interface (UI) through IPU. When receiving a trusted display request, the TUI driver first saves the context of IPU and creates the LUT. Then, the IDMAC, DMFC, and DP are configured to display the sensitive data as a foreground image. Specifically, the IDMAC is configured through the `Fg_IDMAC_Config()` method. The parameter `FrameBufAddr` points to the address of foreground framebuffer in secure iRAM, `width` and `height` set the size of the foreground image, `LUTMode` tells the system to adopt LUT mode for displaying. DMFC is then configured to relay the foreground data from CH27 to the DP through the method `DMFC_Config()`. `DP_Config()` method configures the position of the foreground on the screen and the transparency. Now the foreground will display the sensitive data stored in the iRAM, while the background is unaffectedly displaying the UI in rich OS. The TUI driver keeps the trusted display until receiving a stop signal from user and then restores the context of IPU.

The configuration in Listing 1.2 also works well when the IPU is originally running in the dual display mode. In this situation, the `OriginalIPUContextSave()` and `OriginalIPUContextRestore()` methods will save and restore the IPU context for both foreground and background. When displaying the sensitive data, the background image will not be affected, while the original foreground will be temporarily interrupted.

b) Enabling the Access of Secure iRAM from IPU. Generally, there are two ways to enable the access of secure iRAM from DMA devices like IPU. First, we can set the IPU as a secure master peripheral, so that it can access the secure memory. Second, we can temporarily set the iRAM containing the sensitive foreground framebuffer as non-secure. The first solution can be achieved by configuring the assess policy of IPU as *secure access* through setting the secure access register in CSU. However, the same register also controls the other 5 DMA devices including *VDOA*, *VPU*, *GPU2D*, *GPU3D* and *OPENVG*. Thereby, the configuration enlarges the Trusted Computing Base (TCB) size by allowing the access of secure memory (including secure iRAM and DRAM) to 6 DMA devices. The second solution can be achieved by shrinking the size of secure iRAM through the iRAM controller. However, i.MX6Quad has a quad-core processor, temporarily setting the sensitive iRAM as non-secure might cause data disclosure to the compromised rich OS. To ensure security, the non-secure cores and non-related DMA operations should be temporarily paused before setting the sensitive iRAM as non-secure, which will thence affect the performance. After considering the trade-off between performance and security, we mainly implement the second solution in this work, i.e., temporarily pausing the execution of rich OS and setting the iRAM containing the sensitive foreground framebuffer as non-secure before conducting the configuration operations in Listing 1.2 and restoring them once the trusted display ends.

As described in Section 2.1, one processor core enters the monitor mode when receiving a secure interrupt. Thereby, pausing of non-secure cores could be achieved by sending an inter-processor or inter-core secure interrupt from the secure core. Specifically, the TUI driver first configures the interrupts numbered 8 - 15 as secure FIQs for each core when the TEE OS boots up. Then, it can pause the non-secure cores by sending an FIQ numbered between 8 and 15. In order to resume the non-secure cores in time, the TUI driver stores a global *flag* valued 0 in secure DRAM before sending the FIQ, and sets it to 1 once trusted display ends. The corresponding interrupt handler polls the value of *flag* and switches the cores to the normal world once *flag* is not set to 0.

There are two types of DMA controllers on our i.MX6Quad platform. One is the controller shared by peripherals, i.e., Smart Direct Memory Access (SDMA). The other one is internal controller associated with dedicated peripheral, e.g., IDMAC of IPU. Both types of DMA controllers could be paused and resumed by writing corresponding registers. For example, the SDMA could be paused and resumed by writing the SDMAARM_STOP_STAT and SDMAARM_HSTART registers, respectively. And each channel of IDMAC could be closed and opened by writing the corresponding IPUx_IDMAC_CH_EN_1 and IPUx_IDMAC_CH_EN_2 registers. The TUI driver saves the original states of these DMA controllers in secure DRAM before pausing them, and restores them once trusted display ends.

6 Evaluation

We evaluate Oath by conducting extensive experiments on the prototype implemented on the FreeScale i.MX6Quad sabre development board. The board is equipped with a quad-core ARM Cortex-A9 processor running at 1.2GHz with 1GB DDR3 SDRAM and 256 KB onboard internal RAM. The secure world is deployed with the OP-TEE OS 2.2.0, and the normal world is installed with a FreeScale Android 6.0.1 system with a 4.1.15 Linux kernel. The board is also connected with a screen whose resolution is 1024*768 pixels. To minimize the noise involved during our experiment, we run each test with 1,000 iterations and take the average values as our measurement results.

6.1 Function Impacts

This section explores the function impacts of Oath upon the rich OS and OP-TEE systems.

Impacts on OP-TEE. Oath is implemented in the secure OS using the OP-TEE system, so we first evaluate if functions of OP-TEE system are affected with the modification introduced by Oath. A test suite named *xtest* (optee_test) [11, 35] is shipped with the OP-TEE source codes as a standard test tool for a complete test on the TEE-solution. The test suite includes test cases for both performance test and function test. Here, we utilize only the function-related test cases, whose number is 22647 in total. Among them, 28 test cases fail on the original OP-TEE system, since they need to invoke the static TAs or hardware components that are missing on our platform.

Finally, 22619 test cases are used for evaluation. Generally, the tests are carried out by invoking 12 TAs included in the test suite. The original OP-TEE system loads these TAs in the secure DRAM, and Oath loads .bss, stack, and .data sections of the TAs in the secure iRAM and other parts in the secure DRAM.

Table 1. Function Impacts on OP-TEE

Category	Original	Oath
Main Functions(8815)	√	√
TEE Internal API(12894)	√	√
TA Storage(268)	√	√
Shared Memory(125)	√	√
Key Derivation(225)	√	√
Sanity Test(292)	√	√

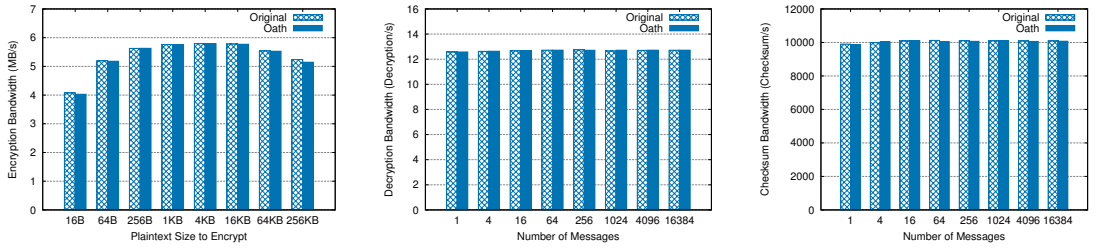
To give a clear comparison, we divide the test cases into six categories according to the functions involved. As shown in Table 1, Oath passes all tests that succeed on the original system, which means Oath introduces no negative impacts on the functions of the OP-TEE system. One thing needs to be mentioned here is that 2 among the 12 TAs have a .bss section whose size exceeds 256 KB (the size of iRAM on our i.MX6Quad platform), and therefore they could not be loaded in iRAM successfully. After a detailed analysis, we find these two TAs are set with a huge .bss section to test the concurrent processing capability of the system, e.g., how many TA instances could execute concurrently. It is not a normal requirement for a usual TA. Our investigation on a commercial TEE vendor also shows the .bss sections of real TAs are commonly no more than 32 KB. Therefore, in our experiments, we modify the .bss size of these two TAs to 32 KB. This causes a smaller concurrent number being output in the results, but does not affect the tests of other functions.

Table 2. Memory Demands of TAs

TA Name	Total(T) (KB)	iRAM(I) (KB)	Ratio (1-I/T)
aes_perf	111.04	47.04	57.64%
concurrent	110.95	46.95	57.68%
*concurrent_large	2126.95	2062.95	3.01%
create_fail_test	89.95	45.95	48.92%
crypt	135.52	47.52	64.94%
*os_test	1122.69	926.69	17.46%
rpc_test	96.96	48.96	49.50%
sha_perf	110.95	46.95	57.68%
sims	128.98	80.98	37.21%
storage	114.95	46.95	59.16%
storage2	114.95	46.95	59.16%
storage_benchmark	98.97	46.97	52.54%

* iRAM memory demand exceeds 256 KB.

Though Oath does not affect the basic functions of the OP-TEE, it does indeed enforce a limitation on the number of the TAs that could be executed concurrently and the size of image that could be securely displayed on the screen, mainly due to the small capacity of the OCRAM memory. Table 2 lists the total memory demands and memory demands of the 12 TAs after deploying Oath. Except two TAs whose iRAM memory demands exceed 256 KB for the requirement of concurrency tests, the average demand of iRAM memory for each TA is 50.52 KB. Therefore, it is able to execute 4 TAs concurrently with the maximum secure iRAM of 236 KB. The *Ratio* column illustrates the effectiveness of the memory splitting mechanism introduced in Oath, which reduces 56.49% iRAM memory demand on average. As for the trusted display, our platform is equipped with a screen whose resolution is 1024*768 pixels, therefore it needs only 96 KB to display an image covering 1/4 screen.



(a) AES Speed Comparison (b) RSA Speed Comparison (c) SHA256 Speed Comparison

Fig. 6. Speed Comparisons on Cryptographic Algorithms

Impacts on Rich OS. Oath introduces two function impacts on rich OS. The first one is to introduce a transient suspension of the rich OS during trusted display. Generally, 6 actions are performed for an image to be displayed securely. The request is first initiated by user from the normal world, which causes a world context switching. Then, the TUI driver suspends non-secure cores and non-related DMA operations, shrinks secure iRAM to exclude the sensitive framebuffer, saves the IPU context, and configures IPU for trusted display. As such, the image will be displayed on the screen through IPU. We measure the time consumed for each action with the Performance Monitoring Unit (PMU) in Cortex-A9 processor. The time breakdown is illustrated in Table 3. Totally, it takes 17293.04 μs for the sensitive data to be displayed on the screen. The duration of rich OS suspension is 2884.44 μs, which includes the actions marked with *.

The rich OS continues to be suspended until the user requests to end trusted display. When receiving a stop request, 5 actions are involved to restore the system. First, the TUI driver will enlarge the secure iRAM to include the sensitive framebuffer, restore the IPU context for normal world, and resume the execution of the DMA operations and non-secure cores. Then, the context will be switched to normal world. Totally, it takes 84.63 μs and the duration of rich OS suspension is 50.17 μs. Therefore, the necessary suspension time for each trusted display operation is 2934.61 μs plus the time to wait for the stop request from user.

Table 3. Time Breakdown of Trusted Display

Action	**Display(μs)**	**Restore(μs)**
World Switching	14408.60	34.46
*Core Suspending/Resuming	3.47	42.24
*DMA Suspending/Resuming	3.24	3.02
*Secure iRAM Adjusting	0.08	0.20
*IPU Context Saving/Restoring	1.44	4.71
*IPU Configuring	2876.21	-
Total	17293.04	84.63

* Rich OS is suspended.

The second functional impact on the rich OS is due to the occupation of the iRAM memory. As discussed in Section 5.2, the lowest 20 KB of iRAM will always be reserved for rich OS. Therefore, the only impact introduced by occupying iRAM is the hardware-based video playing. The experiment results show that the hardware video decoder on our platform utilizes 204 KB iRAM, which means the hardware-based video playing will fail when the available non-secure iRAM is less than 204 KB. However, this problem could be compensated for the following two reasons. First, most video players support two types of decoder, i.e., the hardware decoder and the software decoder. Software video decoder usually does not utilize the iRAM memory. Therefore, it is able to continue video playing through software decoder when TAs or trusted user interactions are executed in the secure OS. On our platform, the videos with resolution not exceeding 1280*720 pixels could be played normally through software decoder. Second, in the normal Android OS, video playing would be suspended once the user shifts focus to another application. Therefore, temporarily suspending video playing can hardly impair user experience when another app is

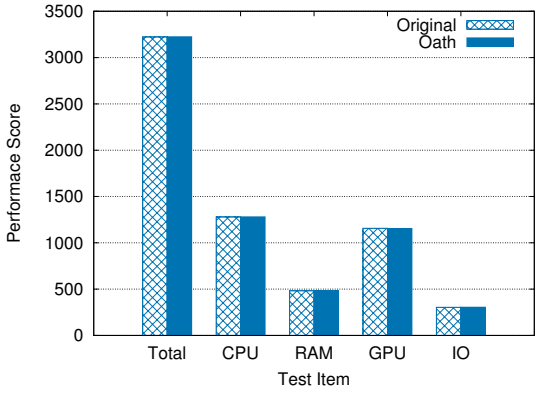


Fig. 7. Comparison of Rich OS Performance

performing TUI operations, since the user interface is commonly multiplexed in a time-sharing fashion among different apps.

6.2 Performance Impacts

We also study the performance impacts of Oath on the original OP-TEE and rich OS systems. We treat the performance observed from native OP-TEE and Android systems as our baseline and compare it with that observed from Oath.

Impacts on OP-TEE. Generally, the impacts on the OP-TEE system are in two aspects. The first one is the time to load a TA, caused by modifying the TA loading procedure. We test the average loading time for the 10 normal TAs in Table 2 with the help of PMU. The result shows that the time to load a TA with and without Oath is 111.27 ms and 109.77 ms, respectively. In other words, Oath introduces an 1.37% overhead.

The second one is the TA execution performance, as the sensitive TA data is now read from or written to iRAM rather than DRAM. We figure it out by implementing the AES, RSA, and SHA 256 algorithms in three TAs, respectively, and comparing their execution performance on the systems with and without Oath deployed. Figure 6 illustrates the speed comparisons on the three cryptographic algorithms, which shows Oath imposes a small overhead on the TA execution. For example, the highest overhead is introduced when making AES encryption with the plaintext size of 256 KB, and the overhead is 1.8%.

Impacts on Rich OS. Oath introduces two differences in rich OS, i.e., porting the IOMUXC operations into the secure world and occupying part of the iRAM memory. As discussed in Section 6.1, the later one is more likely to have function impacts on rich OS rather than performance impacts. However, since the porting introduces two additional world context switching between the secure world and the normal world for each IOMUXC operation, it indeed imposes some overhead on rich OS. First, we compare the system booting time with and without Oath. This is achieved by enabling the *CONFIG_PRINTK_TIME* configuration in Android OS, and obtaining the booting time from the output logs. The result shows that Oath imposes 0.16% overhead in rich OS booting.

Second, we study the overall performance of rich OS through a comprehensive benchmark suite, i.e., AnTuTu 2.9.4. It measures the performance in integer computation, float point operation, 2D and 3D graphic rendering, etc. The results are illustrated in Figure 7, which shows Oath introduces negligible overhead (less than 0.5%) on the execution of rich OS. There is no changes on the computation overhead when we increase the size of secure iRAM in 32 KB increments.

6.3 Security Analysis

Our system targets at protecting the secure sensitive data, including the user input and output data, against both software attacks and physical memory disclosure attacks. In most case, the sensitive data is only stored in secure iRAM and processed by the secure CPU core, so it is immune against both attacks. However, there are two exceptions.

First, when performing the trusted display operations, the iRAM region containing sensitive framebuffer needs to be accessed by the IPU and thus temporarily set as non-secure. However, the data will remain secure. IPU is also an on-SoC component and all data processed by IPU is stored in the internal buffers that are not accessible from outside. Therefore, the data on IPU could resist both attacks. Moreover, the rich OS is suspended when the sensitive iRAM region is temporarily set as non-secure, thereby it cannot carry out software attacks.

Second, during the trusted input process, since the rich OS is not suspended, the coordinate value in ADC register might be leaked if the rich OS polls the register with a small interval. However, since the ADC register is not writable and will be zeroed once it is read, if the attacker gets the value first, the interrupt handler in the secure world can detect it since the register has been zeroed and stop the trusted input operation. If the value is obtained by the normal secure interrupt handler, the attacker can only obtain value 0.

Both code and data integrity protection depend on the TCB of the Oath system, which is protected by the ARM TrustZone technique. Oath introduces four small modules into the secure OS. In total, Oath adds 764 source line of code (SLOC), where 637 SLOC is on implementing the TUI driver.

7 Discussion And Future Work

Transient Suspension During Trusted Display. During the trusted display procedure, Oath temporarily suspends the execution of rich OS for a few seconds to prevent rich OS from stealing the sensitive display data. Two options could be adopted to optimize the user experience during trusted display. First, we can interleave two mutually exclusive activities, running rich OS and displaying trusted output, with a short time interval, instead of totally pausing the rich OS when waiting until the user finishes. A similar solution has been proposed in TrustOTP to shorten the suspension time [50]. Second, the TZ-RKP solution [14] could be adopted to not affect the availability of rich OS for trusted display. The basic idea is to monitor the memory operations, and block the ones accessing the sensitive iRAM memory region. The optimized solution will cause only transient performance degradation during trusted display rather than entire suspend. We leave these optimization solutions as future work.

Portability on Other Hardware Platforms. We design Oath according to the specification of TrustZone architecture. When manufacturers have different implementations based on TrustZone specification, our detailed implementation may need to be changed accordingly. The dual display technology adopted by Oath to achieve trusted display is not dedicated to i.MX6Quad, which has been supported in many platforms, e.g., NXP i.MX 6Quad/7Dual/8DualXPlus/8QuadXPlus, TI DRA71x/DRA72x, and HiSilicon Kirin 960, etc.

Size Limitation of iRAM. Oath shares the same size-limitation problem as other on-chip storage based solutions. We mitigate the size-limitation problem by introducing a memory splitting mechanism, i.e., locking only the sensitive data rather than entire TA in iRAM. In practice, it works well on most of TAs. Furthermore, we can extend Oath to support TAs with a larger size by introducing another level of virtual memory, i.e., utilizing DRAM as a backup storage for encrypted iRAM pages. A similar work has been proposed in CaSE [56] (a cache-based solution) to support larger sensitive applications.

8 Related Work

One research effort focuses on constructing a trusted execution environment through a high privileged entity, such as hypervisor [18, 33, 40, 53–55] or System Management Mode (SMM) [15]. For example, [18, 33] encrypt address space for an application through a hypervisor, so that a hostile OS can only view the address space of the application in ciphertext. However, hypervisor-based solution can not provide the best performance for the resource-constrained mobile platforms. In addition, the hypervisor is already struggling with its own security problems due to increasing TCB size [21, 22]. In this work, Oath utilizes the TrustZone technology to shield applications from potentially compromised OS, which eliminates complex, error-prone resource allocation in a hypervisor.

Hardware-assisted protection is also widely explored to shield the applications from untrusted OSes. Flicker [41] takes advantage of Intel Trusted Execution Technology (TXT) and AMD Secure Virtual machine (SVM) to construct a secure isolated execution environment for protecting security sensitive applications. Haven [16] safeguards applications through Intel Software Guard eXtension (SGX) [42], which provides an efficient secure enclave for isolating sensitive applications and automatically encrypts the data stored outside of enclave. For the mobile devices running on ARM processors, TrustZone technology is widely utilized for shielding applications [23,31,34,36,38,47,50,51,57]. While these works take advantage of TrustZone, they require modifications to applications, and could not tackle the physical memory leakage attacks [28,32,39,43]. CaSE [56] realized similar security goal as this paper, by constructing an isolated environment in the L2 cache of the processor. However, occupation of the cache downgrades the performance of normal OS. According to their evaluation, the system becomes unavailable when more than 60% L2 cache (153.6 KB) is used for CaSE applications. Moreover, as cache is not addressable, CaSE hinders the support of TUI operations as many I/O devices work through DMA mechanisms which could only access the addressable memory. Oath tackles these problems by shielding the application on the on-SoC iRAM memory, which is rarely utilized by the rich OS and can be addressed as normal DRAM.

To defend against the physical memory disclosure attacks, several research works [19,27,29,30, 44,48,49,56] are proposed, which protect the sensitive application in the on-SoC memory, such as register, cache and OCRAM. For example, Sentry [19] and CaSE [56] use cache locking function for CPU-bound execution. Solutions such as ARMORED [27,29,44,48,49] store the sensitive data in the register. Most of the SoC-bound execution solutions are based on the assumption that the mobile OS is trusted, while Oath assumes the OS could be compromised.

A number of research works [36, 37, 45, 50] devote to provide TUI support in the isolated execution environment. For example, Pawel et al. [45] provide trusted input and output functions through a customized PANTA display processor [24]. It can ensure a strong I/O isolation between two execution environments, however it may have compatible issues and increase the cost, as it depends on a dedicated co-processor. TrustOTP [50] enables trusted display and input, however the sensitive data displayed or input is stored in the DRAM rather than OCRAM as done by our Oath, therefore it may suffer physical memory disclosure attacks. Li et al. [37] propose a trusted path design named TrustUI for mobile devices, which enables secure interaction between end users and services against the software attacks. Based on TrustUI, AdAttester [36] detects and prevents well-known ad frauds by providing unforgeable clicks and verifiable display with the help of TrustZone. AdAttester focuses on remote attestation of the UI operations, while Oath focuses on confidentiality and integrity of the sensitive data (including I/O data).

9 Conclusion

To facilitate the security-critical tasks, vendors have integrated TEE systems on their mobile devices. However, the existing TEE systems mainly focus on defending against the software attacks with the help of the TrustZone mechanism. In this paper, we construct an iRAM-based TEE system named Oath that can protect sensitive data against both software attacks and physical memory disclosure attacks. To be compliant with the GlobalPlatform TEE specifications, we implement a prototype of Oath based on the OP-TEE system. Our experimental results show that Oath introduces negligible function and performance overhead on the OP-TEE and Android systems running in the secure world and the normal world, respectively.

Acknowledgment

This work is supported by the National Key Research and Development Program of China under Grant No.2016YFB0800102 and No.2017YFB0802401, the National Natural Science Foundation of China under Grant No.61802398, the National Cryptography Development Fund under Award No.MMJJ20180222 and MMJJ20170215, the U.S. ONR grants N00014-16-1-3214 and N00014-16-1-3216, and the NSF grants CNS-1815650.

References

1. Quad-core Cortex-A15 SoC features 6MB on-chip RAM. http://linuxgizmos.com/quad-core-cortex-a15-soc-features-6mb-on-chip-ram/ (2014)
2. Android KeyStore System. https://developer.android.com/training/articles/keystore.html (2017)
3. GlobalPlatform made simple guide: Trusted Execution Environment (TEE) Guide. https://www.globalplatform.org/mediaguidetee.asp (2017)
4. i.MX 6Dual/6Quad Applications Processors Reference Manual. http://\www.nxp.com/products/microcontrollers-and-processors/arm-based-processors-and-mcus/i.mx-applications-processors/i.mx-6-processors (2017)
5. Lookup Table. https://en.wikipedia.org/wiki/Lookup_table#Lookup_tables_in_image_processing (2017)
6. optee-os. https://github.com/OP-TEE (2017)
7. Platforms Supported by OP-TEE. https://github.com/OP-TEE/optee_os#3-platforms-supported (2017)
8. Press Guidance Samsung Pay. http://security.samsungmobile.com/doc/Press_Guidance_Samsung_Pay.pdf (2017)
9. ARM1176JZF Development Chip On-Chip Memory. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0375a/Cegegajh.html (2018)
10. Arria 10 SoC Hard Processor System. https://www.altera.com/products/soc/portfolio/arria-10-soc/arria10-soc-hps.html (2018)
11. OP-TEE sanity testsuite. https://github.com/OP-TEE/optee_test (2018)
12. Alves, T., Felton, D.: TrustZone: Integrated hardware and software security. ARM white paper **3**(4) (2004)
13. ARM.: TrustZone Secure White Paper. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf (2005)
14. Azab, A.M., Ning, P., Shah, J., Chen, Q., Bhutkar, R., Ganesh, G., Ma, J., Shen, W.: Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world pp. 90–102 (2014)
15. Azab, A.M., Ning, P., Zhang, X.: SICE: a hardware-level strongly isolated computing environment for x86 multi-core platforms. In: ACM Conference on Computer and Communications Security. pp. 375–388 (2011)
16. Baumann, A., Peinado, M., Hunt, G.C.: Shielding applications from an untrusted cloud with haven. ACM Trans. Comput. Syst. **33**(3), 8:1–8:26 (2015). https://doi.org/10.1145/2799647, http://doi.acm.org/10.1145/2799647
17. Bays, C.: A comparison of next-fit, first-fit, and best-fit. Communications of The ACM **20**(3), 191–192 (1977)
18. Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J.S., Ports, D.R.K.: Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems pp. 2–13 (2008)
19. Colp, P., Zhang, J., Gleeson, J., Suneja, S., De Lara, E., Raj, H., Saroiu, S., Wolman, A.: Protecting Data on Smartphones and Tablets from Memory Attacks. architectural support for programming languages and operating systems **50**(4), 177–189 (2015)
20. Coombs, R.: FIDO&TEE:Simpler, Stronger, Authentication. http://www.armtechforum.com.cn/2014/sz/A-8_FIDOandTEE-SimplerStrongerAuthenticat-ion.pdf (2017)
21. CVEdetails.com.: Vmware: Vulnerability statistics. http://www.cvedetails.com/vendor/252/Vmware.html (2018)
22. CVEdetails.com.: Xen: Vulnerability statistics. http://www.cvedetails.com/vendor/6276/XEN.html (2018)
23. Electronics, S.: Samsung KNOX. http://www.samsung.com/global/business/mobile/solution/security/samsung-knox (2018)
24. Evatronix: Evatronix Launches Display Processor based on Latest ARM Security Technology. http://www.electronicsweekly.com/noticeboard/general/\evatronix-launches-display-processor-based-on-latest-arm-security-techno-logy-2012-05/ (2012)
25. Freescale: Hardware Reference Manual for i.MX53 Quick Start. https://www.nxp.com/docs/en/reference-manual/IMX53QSBRM.pdf (2011)
26. Freescale: i.MX 6Solo/6DualLite Applications Processor Reference Manual. http://cache.freescale.com/files/32bit/doc/ref_manual/IMX6SDLRM.pdf (2015)
27. Garmany, B., Müller, T.: PRIME: private RSA infrastructure for memory-less encryption. In: Annual Computer Security Applications Conference, ACSAC '13, New Orleans, LA, USA, December 9-13, 2013. pp. 149–158 (2013). https://doi.org/10.1145/2523649.2523656, http://doi.acm.org/10.1145/2523649.2523656

28. Gogniat, G., Wolf, T., Burleson, W., Diguet, J., Bossuet, L., Vaslin, R.: Reconfigurable Hardware for High-Security/ High-Performance Embedded Systems: The SAFES Perspective. IEEE Transactions on Very Large Scale Integration Systems **16**(2), 144–155 (2008)
29. Götzfried, J., Müller, T.: ARMORED: cpu-bound encryption for android-driven ARM devices. In: 2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg, Germany, September 2-6, 2013. pp. 161–168 (2013). https://doi.org/10.1109/ARES.2013.23, https://doi.org/10.1109/ARES.2013.23
30. Guan, L., Lin, J., Luo, B., Jing, J.: Copker: Computing with Private Keys without RAM. In: network and distributed system security symposium (2014)
31. Guan, L., Liu, P., Xing, X., Ge, X., Zhang, S., Yu, M., Jaeger, T.: Trustshadow: Secure execution of unmodified applications with ARM trustzone. In: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'17, Niagara Falls, NY, USA, June 19-23, 2017. pp. 488–501 (2017). https://doi.org/10.1145/3081333.3081349, http://doi.acm.org/10.1145/3081333.3081349
32. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Communications of The ACM **52**(5), 91–98 (2009)
33. Hofmann, O.S., Kim, S., Dunn, A.M., Lee, M.Z., Witchel, E.: InkTag: secure applications on an untrusted operating system. In: ASPLOS. pp. 265–278 (2013)
34. Jang, J.S., Kong, S., Kim, M., Kim, D., Kang, B.B.: Secret: Secure channel between rich execution environment and trusted execution environment. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015 (2015), https://www.ndss-symposium.org/ndss2015/secret-secure-channel-between-rich-execution-environment-and-trusted-execution-environment
35. Joakim Bech: Testing a Trusted Execution Environment. https://www.linaro.org/blog/testing-a-trusted-execution-environment/ (2016)
36. Li, W., Li, H., Chen, H., Xia, Y.: AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone. In: Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys 2015, Florence, Italy, May 19-22, 2015. pp. 75–88 (2015). https://doi.org/10.1145/2742647.2742676, http://doi.acm.org/10.1145/2742647.2742676
37. Li, W., Ma, M., Han, J., Xia, Y., Zang, B., Chu, C.K., Li, T.: Building trusted path on untrusted device drivers for mobile devices. In: Proceedings of 5th Asia-Pacific Workshop on Systems. p. 8. ACM (2014)
38. Marforio, C., Karapanos, N., Soriente, C., Kostiainen, K., Capkun, S.: Smartphones as Practical and Secure Location Verification Tokens for Payments. In: 21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014 (2014), https://www.ndss-symposium.org/ndss2014/smartphones-practical-and-secure-location-verification-tokens-payments
39. Markuze, A., Morrison, A., Tsafrir, D.: True IOMMU Protection from DMA Attacks: When Copy is Faster than Zero Copy. architectural support for programming languages and operating systems **50**(2), 249–262 (2016)
40. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V.D., Perrig, A.: TrustVisor: Efficient TCB Reduction and Attestation. In: IEEE Symposium on Security and Privacy. pp. 143–158 (2010)
41. McCune, J.M., Parno, B., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: an execution infrastructure for tcb minimization. In: EuroSys. pp. 315–328 (2008)
42. McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative instructions and software model for isolated execution. In: HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013. p. 10 (2013). https://doi.org/10.1145/2487726.2488368, http://doi.acm.org/10.1145/2487726.2488368
43. Muller, T., Spreitzenbarth, M.: Frost: forensic recovery of scrambled telephones. In: applied cryptography and network security. pp. 373–388 (2013)
44. Muller T, Freiling F C, D.A.e.a.: Tresor runs encryption securely outside ram. In: usenix security symposium (2011)
45. Pawel Duc: Secure Mobile Payments - Protecting display data in TrustZone-enabled SoCs with the Evatronix PANTA Family of Display Processors. http://www:design-reuse:com/articles/30675 (2013)
46. Samsung: Samsung Exynos 4412. http://linux-exynos.org/wiki/Samsung_Exynos_4412 (2017)
47. Santos, N., Raj, H., Saroiu, S., Wolman, A.: Using ARM trustzone to build a trusted language runtime for mobile applications. In: Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1-5, 2014. pp. 67–80 (2014). https://doi.org/10.1145/2541940.2541949, http://doi.acm.org/10.1145/2541940.2541949

48. Simmons, P.: Security through amnesia: a software-based solution to the cold boot attack on disk encryption. In: annual computer security applications conference. pp. 73–82 (2011)
49. Simmons, P.: Security through amnesia: a software-based solution to the cold boot attack on disk encryption. In: Twenty-Seventh Annual Computer Security Applications Conference, ACSAC 2011, Orlando, FL, USA, 5-9 December 2011. pp. 73–82 (2011). https://doi.org/10.1145/2076732.2076743, http://doi.acm.org/10.1145/2076732.2076743
50. Sun, H., Sun, K., Wang, Y., Jing, J.: Trustotp: Transforming smartphones into secure one-time password tokens. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015. pp. 976–988 (2015). https://doi.org/10.1145/2810103.2813692, http://doi.acm.org/10.1145/2810103.2813692
51. Sun, H., Sun, K., Wang, Y., Jing, J., Wang, H.: TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices pp. 367–378 (2015)
52. TEXAS INSTRUMENTS: AM5K2E0x Multicore ARM KeyStone II System-on-Chip (SoC) DataSheet. http://www.ti.com/lit/ds/symlink/am5k2e04.pdf (2015)
53. Vasudevan, A., Parno, B., Qu, N., Gligor, V.D., Perrig, A.: Lockdown: towards a safe and practical architecture for security applications on commodity platforms pp. 34–54 (2012)
54. Wang, Z., Jiang, X.: HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity. ieee symposium on security and privacy pp. 380–395 (2010)
55. Yang, J., Shin, K.G.: Using hypervisor to provide data secrecy for user applications on a per-page basis pp. 71–80 (2008)
56. Zhang, N., Sun, K., Lou, W., Hou, Y.T.: Case: Cache-assisted secure execution on arm processors. In: ieee symposium on security and privacy. pp. 72–90 (2016)
57. Zhou, Y., Wang, X., Chen, Y., Wang, Z.: ARMlock: Hardware-based Fault Isolation for ARM. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014. pp. 558–569 (2014). https://doi.org/10.1145/2660267.2660344, http://doi.acm.org/10.1145/2660267.2660344