PumpChannel: An Efficient and Secure Communication Channel for Trusted Execution Environment on ARM-FPGA Embedded SoC

Jingquan Ge †, Yuekang Li †, Yang Liu ‡ † *, Yaowen Zheng †, Yi Liu †, Lida Zhao †
† School of Computer Science and Engineering, Nanyang Technological University
‡ Zhejiang Sci-Tech University
{jingquan.ge, yuekang.li, yangliu, yaowen.zheng}@ntu.edu.sg, {yi009, LIDA001}@e.ntu.edu.sg

***Abstract*—ARM TrustZone separates the system into the rich execution environment (REE) and the trusted execution environment (TEE). Data can be exchanged between REE and TEE through the communication channel, which is based on shared memory and can be accessed by both REE and TEE. Therefore, when the REE OS kernel is untrusted, the security of the communication channel cannot be guaranteed. The proposed schemes to protect the communication channel have high performance overhead and are not secure enough.**

In this paper, we propose PumpChannel, an efficient and secure communication channel implemented on ARM-FPGA embedded SoC. PumpChannel avoids the use of secret keys, but utilizes a hardware and software collaborative pump to enhance the security and performance of the communication channel. Besides, PumpChannel implements a hardware-based hook integrity monitor to ensure the integrity of all hook code. Security and performance evaluation results show that PumpChannel is more secure than the encrypted channel countermeasures and has better performance than all other evaluated schemes.

***Index Terms*—ARM-FPGA, TrustZone, TEE, REE, Communication, kernel**

I. Introduction

In recent years, ARM-based electronic devices have flooded the market. These products have been deeply integrated into daily life, so that they inevitably store or process sensitive privacy data. To protect these data, ARM TrustZone-based Trusted Execution Environment (TEE) was proposed and continuously enhanced [1]–[5]. Nowadays, as consumers pay more attention to information security, the application scenarios of TrustZone-based TEE are not limited to common ARM-based devices such as smartphones and tablets. Various types of IoT and embedded devices, including drones and vehicle electronic equipment, have begun to deploy TEE in their systems [6], [7].

TrustZone-based TEE is isolated from Rich Execution Environment (REE), so that attackers in the REE cannot access resources of TEE directly. The normal applications run in REE while the Trusted Applications (TAs) are executed in TEE. The only way for an REE process to access the resources of TEE is to invoke TAs. Each TA has a corresponding client application (CA) in REE. To access the resources of TEE, a CA process needs to create its own communication channel to exchange data with the corresponding TA. Normally, this channel is established based on the shared memory which can be accessed by both REE and TEE. However, the communication channel is not secure when REE OS kernel is untrusted. Many researchers have successfully exploited this type of vulnerabilities [8]–[10]. Even without the kernel privilege, attackers in user space can deceive the TAs to process deliberately crafted data in the fake memory addresses [11], [12]. Worse still, the REE OS kernel can access physical addresses freely due to the existence of the *Physmap* mechanism [13]. Therefore, the data in these communication channels can be easily stolen or tampered with by the untrusted REE OS kernel (e.g., man-in-the-middle attack). To protect the communication channel between TEE and REE, researchers have proposed several software defense schemes [12], [14]–[16]. However, these software defense schemes are either not secure enough or have too much performance overhead.

Benefiting from the special architecture combining hardware and software, ARM-FPGA embedded SoC has been widely used in drones, vehicle electronic equipments, machine vision systems and other IoT devices [17], [18]. Similar to smartphones and tablets, these security-sensitive devices also require TrustZone-based TEE to protect sensitive data [6], [7], [19]. Since both the software and hardware on the ARM-FPGA embedded SoC can be freely programmed, it provides researchers with more possibilities to implement secure and efficient hardware/software co-design for TEE.

In this paper, we propose PumpChannel, a hardware-software collaborative design to protect the communication channel of TrustZone-based TEE on ARM-FPGA embedded SoC. PumpChannel consists of a pump in hardware and hooks in software, which can pull/push the sensitive data out of/into the communication channel. When a CPU core enters REE kernel mode from REE user mode or TEE, the data are pumped from the communication channel into the memory owned by PumpChannel. They will be pumped from the memory owned by PumpChannel into the communication channel when a CPU core enters TEE or returns to REE user mode from REE kernel mode. Therefore, the malicious REE kernel cannot steal or tamper with sensitive data in the communication channel. To ensure the integrity of all the hooks, we design the hook integrity monitor in the hardware, which can continuously check whether the code segment of

* The corresponding author.

978-3-9819263-7-8/DATE23/© 2023 EDAA

each hook has been modified. The performance evaluation results show that PumpChannel incurs less than 28% overhead on different CA executions and less than 82% overhead on TA invocation in different payload sizes. The two types of performance overhead are better than other defense schemes. Our main contributions are summarized as follows:

- We create a hardware-software collaborative design—PumpChannel—on ARM-FPGA embedded SoC. It provides a secure communication mechanism for TEE, which can stop the untrusted REE kernel from accessing the communication channel.
- PumpChannel avoids security vulnerabilities and encrypted calculations caused by the use of secret keys, so it has better security and performance.
- We utilize PumpChannel and other schemes to run attack and performance evaluation experiments. The results show that PumpChannel is not only secure but also efficient.

II. Background

This section provides necessary backgrounds of PumpChannel, including the architecture of TrustZone-based TEE, the communication channel between REE and TEE, and the ARM-FPGA embedded SoC. Besides, since the attack experiments in Section V utilize direct mapping in kernel address space (*Physmap*), we introduce its background knowledge in detail.

A. Architecture of TrustZone-based TEE

TrustZone [20] is a hardware security extension of ARM processor, which has been widely deployed in mobile phones, tablets, drones, automotive electronics and other IoT devices. Based on TrustZone technology, the system can be separated into two domains: the Rich Execution Environment (REE) and the Trusted Execution Environment (TEE). Various Client applications (CAs) run in REE, while Trusted Applications (TAs) run in TEE. Each TA in TEE has a corresponding CA in REE to interact with. There are two TEE-related components in REE OS, namely TEE client and TEE driver. TEE client is a dynamic library that provides convenient APIs for the CA, while TEE driver is responsible for the interaction between CA and TA.

B. Communication Channel between REE and TEE

Before REE and TEE exchange data to each other, the communication channel needs to be established between the two domains. This communication channel is based on the cross-domain shared memory [2] that can be accessed by both REE and TEE. It is the communication channel between the corresponding CA and TA. Generally, cross-domain shared memory is a physically continuous memory pool. When the system is running, the shared memory is mainly managed by the TEE driver, including allocation and release. The smallest unit of cross-domain shared memory is a 4KB block, and each block corresponds to a unique ID. The CA process accesses the corresponding shared memory block by specifying the ID. The TEE driver obtains the physical address according to the shared memory ID, and sends the physical address to the TEE. Then, TA gets this physical address and reads the data input of CA. After the TA's calculation process is completed, the calculation result will be stored into the corresponding shared memory. Then, the CA process reads the output data from the shared memory and continues to process the data.

C. Direct Mapping in Kernel Adress Space—Physmap

In the kernel address space, there is a large continuous virtual memory area called "*Physmap*", which contains the direct mapping of part or all of the physical memory. In x86-64 and AArch64 architecture systems, *Physmap* directly maps all of RAM physical addresses [13]. *Physmap* can allocate or deallocate memory without touching the kernel's page table [21], so it is essential to improve kernel performance. In AArch64 architecture, *Physmap* is mapped with RW (readable and writeable) permissions in every kernel version. This leaves an exploitable vulnerability for attackers. The existence of *Physmap* will cause the virtual address of the user process and the kernel virtual address to be mapped to the same physical address, resulting in virtual address aliases or synonyms [22]. Since AArch64's *Physmap* maps all physical memory, malicious kernel modules can access the memory of user processes through kernel-resident synonym. More importantly, *Physmap* mechanism bypasses the page table access mechanism, so all protections based on the page table are invalid.

III. Threat Model and Assumptions

PumpChannel aims to build a secure and efficient trusted communication channel between CA and TA on the ARM-FPGA embedded SoC to resist potential attacks from the untrusted REE OS kernel. PumpChannel is a software/hardware collaborative design, so our security assumptions are divided into software and hardware. In software, we assume that components running in TEE (including TAs, TEE OS, and the secure monitor) are all trusted and booted up by the TrustZone-based secure boot technology [23]. We also assume that the legitimate CA process in REE will not maliciously use TA or leak cross-domain communication data to the REE OS kernel. In hardware, we assume that all the hardware modules of PumpChannel can only be accessed by the hooks or the OP-TEE driver.

In this paper, the attacker with kernel privilege in the REE can modify and monitor the code in the REE OS kernel, but cannot manipulate the process scheduling and OP-TEE driver code. Specifically, an attacker can launch two types of attack on the communication channel for TEE. First, the attacker can obtain the address of the communication channel and access (stealing or tampering with) the sensitive data in it. Second, the attacker can utilize *Physmap* to bypass the protection of the page table and freely access physical addresses. With this ability, the attacker can steal the secret key of the encrypted communication channel.

IV. Design and Implementation

This section details the design and implementation of our hardware/software collaborative design — PumpChannel. We

first describe the components of PumpChannel and the function of each component in Section IV-A. Section IV-B shows the data and control flow of PumpChannel.

A. Overview of PumpChannel

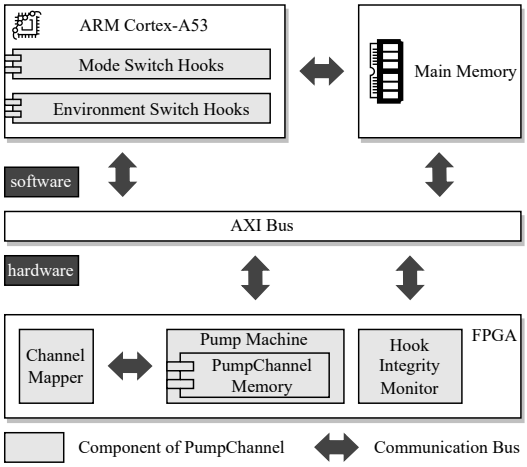

Fig. 1. The architecture of PumpChannel

As shown in Figure 1, PumpChannel consists of 6 modules, of which 4 modules are implemented in hardware and 2 modules in software. The hardware modules of PumpChannel are respectively named *Pump Machine*, *Channel Mapper*, *PumpChannel Memory* and *Hook Integrity Monitor*. While *Mode Switch Hooks* and *Environment Switch Hooks* are the software modules of PumpChannel. Next, we describe the main functions of each module in detail.

1) Hardware Modules of PumpChannel:

Channel Mapper: The $current \rightarrow pid$ of the legal CA and the physical addresses of the corresponding communication channel are stored in it. The purpose of storing $current \rightarrow pid$ is to allow the legitimate CA process to only access their own communication channel.

Pump Machine: It pumps sensitive data into or out of the communication channel according to the process and address information stored in the *Channel Mapper*.

PumpChannel Memory: This memory is implemented based on the block RAM [24] of FPGA and utilized to temporarily store sensitive data pumped out of the communication channel.

Hook Integrity Monitor: It can monitor whether the code segment of every hook function has been modified.

2) Software Modules of PumpChannel:

Mode Switch Hooks: They include two hooks, user-kernel hook and kernel-user hook. The mode switch from user mode to kernel mode of each CPU core is hooked by the user-kernel hook. The kernel-user hook hooks the opposite mode switch. The two hooks trigger the *Pump Machine* module to pump data into or out of the communication channel.

Environment Switch Hooks: They include two hooks, REE-TEE hook and TEE-REE hook. The environment switch from REE to TEE is hooked by REE-TEE hook. The opposite environment switch is hooked by TEE-REE hook. The *Pump Machine* module can be triggered by the two hooks to pump data into or out of the communication channel.

B. Data and Control Flow of PumpChannel

In this section, we detail the data and control flow of PumpChannel. As shown in Figure 2, We divide the entire system into four states, which are REE user mode, REE kernel mode, TEE kernel mode and TEE user mode. *Mode Switch Hooks* are in the middle of REE user mode and REE kernel mode to hook the switch of these two states. *Environment Switch Hooks* hook the switch between TEE and REE, so they are in the middle of REE kernel mode and TEE kernel mode. Both the data and control flow of the entire design has 6 steps. Next, we describe the steps of control and data flow.

1) REE User Mode → REE Kernel Mode: When the system switches from REE user mode to REE kernel mode, step $\{A\}$, $\{1\}$ and $\{2\}$ will be executed. **{A}:** The user-kernel hook sends *pump_out* signal to *Pump Machine*. The *pump_out* signal can trigger the *Pump Machine* module to pump data. **{1}:** When *Pump Machine* receives *pump_out* signal, it will pump the sensitive data out of all the communication channels and store them into *PumpChannel Memory*. **{2}:** After step $\{1\}$ is completed, *Pump Machine* will clear all the data of the communication channels to 0.

2) REE Kernel Mode → REE User Mode: Step $\{C\}$ and $\{6\}$ will run when the system switches from REE Kernel Mode to REE User Mode. **{C}:** The kernel-user hook sends the *pump_in* and $current \rightarrow pid$ signals to *Pump Machine*. The *pump_in* signal can trigger *Pump Machine* to pump data. **{6}:** When *Pump Machine* receives *pump_in* and $current \rightarrow pid$ signals, it will pump the current process's sensitive data from *PumpChannel Memory* into the current process's communication channel.

3) REE Kernel Mode → TEE Kernel Mode: At the moment when REE Kernel Mode switches to TEE Kernel Mode, step $\{B\}$ and $\{3\}$ will run. **{B}:** Same as the kernel-user hook, the REE-TEE hook sends *pump_in* and $current \rightarrow pid$ signals to *Pump Machine*. **{3}:** When *pump_in* and $current \rightarrow pid$ signals are received by *Pump Machine*, the current process's sensitive data will be pumped by *Pump Machine* from *PumpChannel Memory* into the current process's communication channel.

4) TEE Kernel Mode → REE Kernel Mode: When the system switches from TEE kernel mode to REE kernel mode, step $\{D\}$, $\{4\}$ and $\{5\}$ will be executed. **{D}:** The TEE-REE hook sends *pump_out* signal to *Pump Machine*. **{4}:** When *pump_out* signal reaches *Pump Machine*, the sensitive data will be pumped out of all the communication channels and stored into *PumpChannel Memory* by *Pump Machine*. **{5}:** After step $\{4\}$, *Pump Machine* zeros the physical addresses of all the communication channels.

5) Hook Integrity: Step $\{E\}$ and $\{F\}$ is to ensure the integrity of all the hooks in PumpChannel's design. **{E}:** *Hook Integrity Monitor* cyclically checks the static code segment of the *Mode Switch Hooks* and calculates the hash value. If it

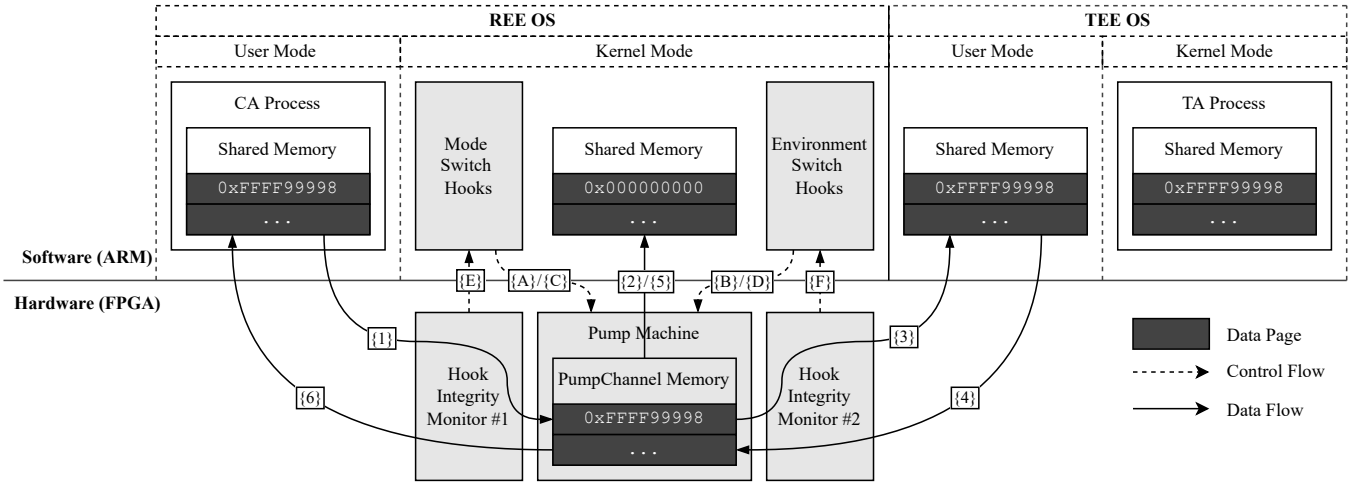

Fig. 2. Data and control flow of PumpChannel

is found that the code segment of the *Mode Switch Hooks* has been modified, the *Hook Integrity Monitor* will correct it immediately. **{F}:** Same as step {E}, the *Hook Integrity Monitor* will check the static code segment of *Environment Switch Hooks* cyclically and correct it in real time.

V. Security Analysis

In this section, we describe a security analysis for PumpChannel. First, we introduce our attack experimental setup. Second, We present the comparative attack results of the original communication channel and PumpChannel. Finally, we show the successful attack on the secret key of SeCReT scheme to present the vulnerability of encrypted communication channel.

A. Attack Experimental Setup

Our experimental platform is the ZCU102 Evaluation Board [25], which is a complete development kit for designers who are interested in exploring designs using Xilinx Zynq UltraScale+ MPSoC [26]. Zynq UltraScale+ MPSoC is the Xilinx second-generation Zynq platform, which combines Cortex-A53 64-bit quad-core processor, Cortex-R5 dual-core real-time processor and FPGA together into a single device. We perform all the experiments on the embedded Linux system running on the Zynq UltraScale+ MPSoC. The version of Linux kernel is 4.14.0-xilinx-v2018.2. The cross compiler we use is petalinux-2018.2. We deploy OP-TEE 3.6.0 on Zynq UltraScale+ MPSoC.

B. Comparative Attacks on the Original Communication Channel and PumpChannel

To prove that PumpChannel can more securely protect sensitive data in the communication channel, we design the comparative attack experiments. The attacker installs his own malicious kernel module in the REE kernel. The attacker's intention is to utilize malicious code in the REE kernel to tamper with the data in the communication channel. The attacker modifies the input data of the communication channel to make the decryption TA output an illegal result. In the attack experiments, we separately attack the original OP-TEE communication channel and PumpChannel. In our attack experiment against the original OP-TEE communication channel, the legitimate CA process is running on CPU core #1 and the attacker process is running on CPU core #2. The result of the experiment is that the input data has been successfully tampered with by the attacker using the *Physmap* method. In contrast, we conduct the same attack experiment on PumpChannel. In this experiment we find that when the attacker process enters kernel mode, all data in the communication channel is 0 before being tampered with. More importantly, after the attacker process tampers with the data in the communication channel, the CA process still get the correct result. It indicates that the attack failed. The results of the comparative attacks prove that PumpChannel can effectively resist malicious kernel tampering with data in the communication channel. So PumpChannel is more secure than the original communication channel of OP-TEE.

C. Attacks on the Encrypted Communication Channel

To prevent directly tampering with the data in the communication channel, researchers proposed two encrypted channel schemes, namely SeCReT [14] and SeCReT_Opt [15]. However, in both schemes, the secret keys used to encrypt the communication channel are not secure enough to resist *Physmap* attacks. To demonstrate this, we specially design attack experiments against SeCReT and SeCReT_Opt. The attacker process utilizes other techniques to obtain the physical address of the key page, and then uses *Physmap* to access these physical addresses. The experimental results show that both the secret keys of SeCReT and SeCReT_Opt have been successfully stolen. PumpChannel avoids the use of secret keys, which also avoids the security vulnerabilities of secret key.

VI. Performance Evaluation

In this section, we evaluate the performance of PumpChannel by running three tests on ZCU102 development board. The

TABLE I
LMBENCH MICRO-BENCHMARK OVERHEAD.

Test Case	*Original Linux*	*SeCReT Enabled*	*SeCReT_Opt Enabled*	*TrustICT Enabled*	*PumpChannel Enabled*
open/close	1.00x	1.63x	1.59x	6.85x	(10.81μs) 2.16x
read	1.00x	3.73x	4.45x	10.28x	(3.43μs) 6.55x
write	1.00x	3.74x	4.24x	≈6.67x	(3.33μs) 8.52x
fork+exit	1.00x	1.18x	1.23x	3.78x	(353.86μs) 1.44x
fork+exec	1.00x	1.18x	1.22x	≈3.78x	(384.64μs) 1.37x

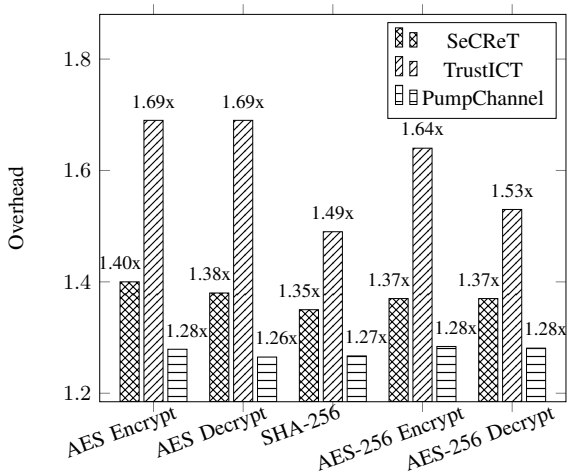

Fig. 3. Time overhead of different CA execution

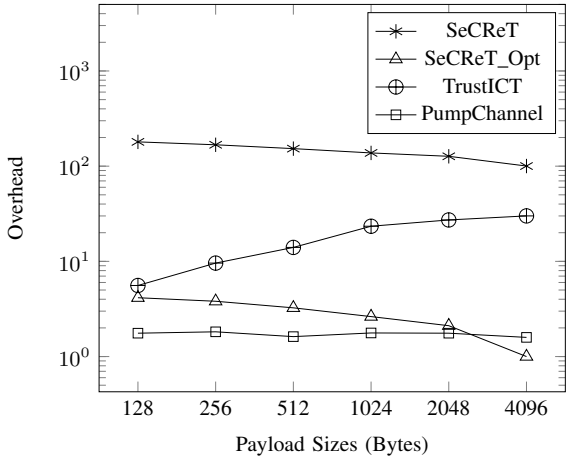

Fig. 4. Time overhead of TA (AES encrypt) invocation in different payload sizes

performance experimental setup are the same as Section V. In general, PumpChannel adds about 198 LOC to REE OS and 49 LOC to TEE OS. To reduce random noise, our experimental data is the average of 10,000 cycles. We compare SeCReT, SeCReT_Opt, TrustICT and PumpChannel in the evaluation tests to show the performance advantages of PumpChannel.

In our performance tests, We compared PumpChannel with three other schemes. In addition to SeCReT and SeCReT_Opt mentioned in Section V, we also include TrustICT [16]. TrustICT utilizes TZASC to dynamically set the access permission of the physical addresses of the communication channel. Since the use of secret keys is avoided, TrustICT is the most close work to PumpChannel. Since none of these three schemes have open source code, their performance evaluation results are directly copied from their papers.

A. Evaluation of REE OS

PumpChannel executes a hook function every time the mode is switched. This mechanism will introduce performance overhead to REE OS. We measure this overhead by running LMBench test suites [27]. Table I shows the results of LMBench. In general, the LMBench overhead of PumpChannel is slightly higher than SeCReT and SeCReT_Opt, but much lower than TrustICT. This overhead is a bit large because *Mode Switch Hooks* need to trigger *Pump Machine* and clear data of all communication channels. The performance overhead of TrustICT is much larger than PumpChannel. The main reason is that TrustICT's multi-core/multi-threaded technology is based on software, which will greatly increase the system performance overhead.

B. Evaluation of CA and TA

PumpChannel modifies the original communication channel, so it will definitely affect the communication between CAs and TAs. We test 5 CA executions using OP-TEE's own test suite, including AES encryption, AES decryption, SHA-256, AES-256 encryption and AES-256 decryption. A complete CA execution contains multiple operations, including loading TA images, preparing key, allocating and freeing shared memory, environment switching, reading and writing shared memory, etc. Figure 3 shows the time overhead of different CA executions. It can be seen from Figure 3 that TrustICT has the largest CA execution time overheads, followed by SeCReT. PumpChannel has the smallest time overheads for CA execution. On average, TrustICT increases its time overhead by approximately 60%, and SeCRet increases by approximately 35%. The time overhead of PumpChannel is less than 28%.

Figure 4 presents the time overhead of the TA invocation in different payload sizes. In this test, we use AES encryption as a typical representative. Overall, PumpChannel has very low overhead under all payload size conditions. The performance advantage of software and hardware coordination keeps the overhead of PumpChannel always below 2.00x. The SeCReT scheme has the worst performance overheads among the four schemes. This is mainly because the secret key protection mechanism of SeCReT is very complicated. Compared with SeCReT, SeCReT_Opt optimizes the secret key protection

mechanism, so its overhead is extremely smaller than SeCReT. TrustICT's overhead is much larger than SeCReT_Opt and PumpChannel. The main reason is that TrustICT's multi-core/multi-threaded scheduling mechanism will greatly increase the time to access the communication channel.

VII. Related Work

TrustZone-based TEE: A number of research works utilize TrustZone-based TEE to improve the security of critical applications in mobile devices, multifunctional embedded and IoT devices. AdAttester [28] provides a verifiable mobile ad framework based on TrustZone. TrustOTP [29] isolates the software-based OTP token in the TEE to realize trusted display of one-time passwords. TrustShadow [30] leverages TrustZone to guarantee secure execution of unmodified applications. Unfortunately, all these works did not consider the security of the communication channel between REE and TEE.

Protection of Communication Channel: For the purpose of resisting malicious attacks from the REE OS kernel, researchers have proposed several protection schemes. Jang *et al.* [14], [15] proposed the SeCReT/SeCReT_Opt countermeasures, which encrypt the communication channel with a specially protected secret key. SOTPM [31] encrypts sensitive data in the application layer before writing the data into the shared memory. However, none of these encryption schemes can guarantee the security of the secret key. To better solve the security issues, Wang *et al.* [16] proposed TrustICT. Using TZASC, TrustICT can dynamically set the access permission of the shared memory. However, they are either not secure enough or have high performance overhead.

VIII. Conclusion

In this paper, we utilize software and hardware collaborative technology to design PumpChannel on ARM-FPGA embedded SoC, which is a secure and efficient communication channel between REE and TEE. PumpChannel can dynamically pump the data into or out of the communication channel, so that the untrusted REE OS kernel cannot steal or tamper with the data in the communication channel. Moreover, PumpChannel can scan critical code segments in real time to ensure that the integrity of all hooks is not damaged. The experiment results show that PumpChannel is more efficient and more secure.

Acknowledgment

This research is supported by National Research Foundation through its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) project under the National Cybersecurity R&D (NCR) Grant award no. NRF2018NCR-NSOE003-0001.

References

[1] Sierraware. (2018) Sierratee trusted execution environment. https://www.sierraware.com/open-source-ARM-TrustZone.html.
[2] Linaro. (2017) optee-os. https://github.com/OP-TEE/opte_os.
[3] Qualcomm. (2017) Qseecomapi.h. https://android.googlesource.com/platform/hardware/qcom/keymaster/+/master/QSEEComAPI.h.
[4] Google. (2017) Trusty tee. https://source.android.com/security/trusty/.
[5] Samsung. (2017) About knox. https://www.samsungknox.com/en/about-knox.
[6] Trustonic. (2017) Not just droning on! the rise of kinibi-m. https://www.trustonic.com/opinion/not-just-droning-rise-kinibi-m/.
[7] ——. (2017) Securing connected cars of the future. https://www.trustonic.com/automotive/.
[8] CVEdetail.com. (2013) cve-2013-3051. https://www.cvedetails.com/cve/CVE-2013-3051/.
[9] D. Rosenberg. (2013) Unlocking the motorola bootloader. http://blog.azimuthsecurity.com/2013/04/unlocking-motorola-bootloader.html.
[10] N. Keltner. (2014) Here be dragons: Vulnerabilities in trustzone. https://atredispartners.blogspot.com/2014/08/here-be-dragons-vulnerabilities-in.html.
[11] K. Lady. (2016) Sixty percent of enterprise android phones affected by critical qsee vulnerability. https://duo.com/blog/sixty-percent-of-enterprise-android-phones-affected-by-critical-qsee-vulnerability.
[12] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y. R. Choe, C. Kruegel, and G. Vigna, "BOOMERANG: exploiting the semantic gap in trusted execution environments," in *NDSS 2017*.
[13] V. P. Kemerlis, M. Polychronakis, and A. D. Keromytis, "ret2dir: Rethinking kernel isolation," in *Proceedings of the 23rd USENIX Security Symposium, 2014*, K. Fu and J. Jung, Eds., pp. 957–972.
[14] J. S. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, "Secret: Secure channel between rich execution environment and trusted execution environment," in *NDSS 2015*.
[15] J. Jang and B. B. Kang, "Securing a communication channel for the trusted execution environment," *Comput. Secur.*, vol. 83, pp. 79–92, 2019.
[16] J. Wang, Y. Wang, L. Lei, K. Sun, J. Jing, and Q. Zhou, "Trustict: an efficient trusted interaction interface between isolated execution domains on ARM multi-core processors," in *SenSys*. ACM, 2020, pp. 271–284.
[17] Enclustra. (2017) Zynq ultrascale+ drone controller. https://www.enclustra.com/en/projects/zynq-ultrascale-drone-controller/.
[18] Xilinx. (2017) Unleash the unparalleled power and flexibility of zynq ultrascale+ mpsocs. https://www.xilinx.com/support/documentation/white_papers/wp470-ultrascale-plus-power-flexibility.pdf.
[19] TrustedFirmware. (2019) how to run optee on zynqmp zcu10x and ultra96 board. https://optee.readthedocs.io/en/latest/building/devices/zynqmp.html.
[20] ARM. (2009) Arm security technology building a secure system using trustzone technology. https://developer.arm.com /documentation/PRD29-GENC-009492/c?lang=en.
[21] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.
[22] E. J. Koldinger, J. S. Chase, and S. J. Eggers, "Architectural support for single address space operating systems," in *ASPLOS*, 1992.
[23] ARM. (2019) Trusted firmware-a. https://github.com/ARM-software/arm-trusted-firmware.
[24] Xilinx. Block memory generator v8.4 logicore ip product guide. https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_4/pg058-blk-mem-gen.pdf.
[25] ——. (2019) Zcu102 evaluation board user guide. https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf.
[26] ——. (2020) Zynq ultrascale+ device technical reference manual. https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf.
[27] L. W. McVoy and C. Staelin, "lmbench: Portable tools for performance analysis," in *Proceedings of the USENIX Annual Technical Conference, 1996*, pp. 279–294.
[28] W. Li, H. Li, H. Chen, and Y. Xia, "Adattester: Secure online mobile advertisement attestation using trustzone," in *MobiSys*, G. Borriello, G. Pau, M. Gruteser, and J. I. Hong, Eds. ACM, 2015, pp. 75–88.
[29] H. Sun, K. Sun, Y. Wang, and J. Jing, "Trustotp: Transforming smartphones into secure one-time password tokens," in *CCS*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 976–988.
[30] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "Trustshadow: Secure execution of unmodified applications with ARM trustzone," in *MobiSys*. ACM, 2017, pp. 488–501.
[31] D. Shim and D. H. Lee, "SOTPM: software one-time programmable memory to protect shared memory on ARM trustzone," *IEEE Access*, vol. 9, pp. 4490–4504, 2021.