IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 19, 2022, accepted April 12, 2022, date of publication April 22, 2022, date of current version May 4, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3169767

Trusted Execution Environment Hardware by Isolated Heterogeneous Architecture for Key Scheduling

TRONG-THUC HOANG[1,2], (Member, IEEE), CKRISTIAN DURAN[2], (Graduate Student Member, IEEE), RONALDO SERRANO[2], (Student Member, IEEE), MARCO SARMIENTO[2], (Graduate Student Member, IEEE), KHAI-DUY NGUYEN[2], (Student Member, IEEE), AKIRA TSUKAMOTO[1], KUNIYASU SUZAKI[1,3], (Member, IEEE), AND CONG-KHA PHAM[2], (Member, IEEE)

[1]National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 135-0064, Japan
[2]Department of Computer and Network Engineering, The University of Electro-Communications (UEC), Tokyo 182-8585, Japan
[3]Technology Research Association of Secure IoT Edge Application Based on RISC-V Open Architecture (TRASIO), Tokyo 101-0022, Japan

Corresponding author: Trong-Thuc Hoang (thuc@vlsilab.ee.uec.ac.jp)

This work was supported by the New Energy and Industrial Technology Development Organization (NEDO) under Project JPNP16007.

ABSTRACT A Trusted Execution Environment (TEE) sets a platform to secure applications based on the Chain-of-Trust (CoT). The starting point of the CoT is called the Root-of-Trust (RoT). However, the RoT implementation often relies on obscurity and provides little flexibility when generating keys to the system. In this paper, a TEE System-on-a-Chip (SoC) architecture is proposed based on a heterogeneous design by combining 64-bit Linux-capable processors with a 32-bit Micro-Controller Unit (MCU). The TEE is built on the 64-bit cores, while the 32-bit MCU takes care of sensitive data and activities. The MCU is isolated from the TEE side by an Isolated Bus (IBus) that sits above the conventional System Bus (SBus). Besides the 32-bit processor, the isolated sub-system contains a Random Access Memory (RAM), a Read-Only Memory (ROM) for storing the boot program, and another ROM for storing root keys. For cryptography accelerators, we have 512-bit Secure Hashing Algorithm 3 (SHA3-512), 128/256-bit Advanced Encryption Standard (AES-128/256), Ed25519, and True Random Number Generator (TRNG) attached to the Peripheral Bus (PBus). Additionally, besides the public channel, the TRNG module also has a private channel that goes directly to the IBus. With RoT implemented inside the isolated sub-system, the RoT is inaccessible from the TEE side after boot. Furthermore, the hidden MCU's secure boot program makes the key generation flexible and could be updated for many security schemes. To summarize, the proposed design features a flexible and secure boot procedure with complete isolation from the TEE domain. Moreover, exclusive secure storage for the root key and cryptographic accelerators are available for the boot process. The implementation was tested on a Virtex-7 XC7VX485T Field-Programmable-Gate-Array (FPGA). It was also synthesized in a Very Large-Scale Integrated (VLSI) circuit with the ROHM-180nm process library.

INDEX TERMS Heterogeneous, isolation, RISC-V, secure-boot, trusted execution environment.

I. INTRODUCTION

A Trusted Execution Environment (TEE) is currently the most common security extension for an Operating System (OS) [1], [2]. The idea of it is to provide isolation between applications, thus creating a barrier between programs. The barrier could be implemented at multiple levels and in many ways. The most basic implementation is the isolation done by using privilege separation at the supervisor-mode (i.e., kernel-space), a pure software approach. Nowadays, recent TEE models are gearing up with barrier enforcers implemented in software and hardware at many architectural levels.

The associate editor coordinating the review of this manuscript and approving it for publication was Seok-Bum Ko.

With TEE, the computer system would run a code with the intended privileges only if that code was authenticated. Hence, an unauthenticated or infected code cannot run on the trusted side or gain any special right. The most common TEE models nowadays are the Intel Software Guard eXtensions (SGX) [3]–[6] with its variations (i.e., Haven [7]), ARM TrustZone [8], [9] with its modifications (i.e., Komodo [10]), and AMD Secure Encrypted Virtualization (SEV) [11] with its descendants (i.e., AMD SEV-ES [12] and AMD SEV-SNP [13]). However, they are closed-source TEEs; thus, we cannot modify their boot flow or their hardware primitives. Recently, with RISC-V emerging, many newly developed RISC-V-based TEE models were proposed, such as Hex-Five MultiZone [14], Sanctum [15], TIMBER-V [16], Keystone [17], and CUstomizable and Resilient Enclaves (CURE) [18].

The strength of TEE relies on the Chain-of-Trust (CoT), which is a series of cryptographic functions such as hashing, signing, verification, and encryption. At each operating layer (i.e., machine-mode, supervisor-mode, and user-mode), the authentication is signed by the previous lower layer (higher privilege) and verified by the next higher layer (lower privilege), thus creating the chain. The beginning of the CoT is the first authentication at reset called the Root-of-Trust (RoT). For security reasons, RoT should be inaccessible from the Rich Execution Environment (REE) or even the TEE processors after boot. RoT could be anything from a randomly generated value, an asymmetric secret key, to a device's certificate pre-signed by a root Certificate Authority (CA). For simplicity, in most crypto-systems, the RoT is often elaborated from a hard-coded Read-Only Memory (ROM) or generated locally before boot. As a result, the confidentiality of the root key and the integrity of the whole TEE system rely on the secure boot process to establish the RoT.

In a cryptography system, the root key is the most sensitive data. Therefore, the innovations for protecting them have been explored and implemented from time to time. Several attempts have tried to protect the RoT by adding obscurity to bury the RoT deep under many layers, thus increasing the cost for attackers. For example, Noorman *et al.* [19] used a trusted authority Intellectual Property (IP) core for key management. The master key was generated and salted inside the IP. Then, a Key Derivation Function (KDF) was used for obtaining subsequent keys for each core while the generated master keys are safe inside the IP. Zhao *et al.* [20] presented an RoT by using Physical Unclonable Function (PUF), which is based on Static Random Access Memory (SRAM) to provide better secure storage. Qin *et al*. [21] proposed a TEE by combining dynamic measurement and integrity flow control with a PUF device for storing keys and returning program functions' addresses. Benhani *et al*. [22] evaluated the security of ARM TrustZone in Field-Programmable Gate Array (FPGA), thus recommending isolation between secure IPs and non-secure IPs to evade various kinds of attack models.

In the implementations mentioned above, because the RoT is still in the TEE system, uncovering the root key becomes a matter of time and depends on a budget of attackers. To truly put the RoT out of the TEE system, the boot process and key management must be done not by the TEE processors but by another party inaccessible from the TEE-side after boot. In the end, TEE is just an isolated environment, and it cannot be the RoT. As a general rule of thumb, a secure boot process with RoT is recommended to be run by hardware primitives. Many TEE models begin with the assumption that their trusted firmware was securely loaded into the stack by a trusted hardware-assisted secure boot process. In practice, common TEEs use extra hardware or third-party IPs. For example, Intel SGX relied on the Intel Active Management Technology (AMT) [23], ARM TrustZone used ARM CryptoCell [24], AMD SEV required Platform Security Processor (PSP) [25], and for RISC-V, many RoT IPs were presented, such as Rambus CryptoManager [26] and OpenTitan [27]. To summarize, the secure boot with RoT in TEE is still an ongoing problem that needs research and development.

With a RISC-V architecture, custom hardware can be tailored for custom TEE, thus creating many potentials for solving long-lasting problems. Several noteworthy security-driven RISC-V-based hardware implementations are the ITUS [28], [29], SiFive WorldGuard [30], HECTOR-V [31], and CURE [18]. ITUS [28], [29] proposed a fixed hardware solution for the secure boot process with RoT. WorldGuard [30] presented a computer system with security primitives developed for strengthening the isolation in TEEs. HECTOR-V [31] focused on the idea of separating REE and TEE processors. And CURE proposed a new TEE model together with newly developed hardware primitives to provide multiple types of enclaves (i.e., kernel-space, user-space, and self-contained sub-space enclaves).

Although various RISC-V-based TEEs were proposed as mentioned above, some of them relied on obscurity for preserving the root key [18], [30], [31], and some others used a fixed circuit with a specific set of constraints to do the RoT [19], [27]–[29]. Therefore, they did not provide the flexibility for the key generation scheme, and the attack surface still exists because the RoT is still in the TEE domain. In this paper, we propose a secure boot solution with hardware RoT in an isolated architecture based on RISC-V. The proposed architecture is a continuous development of the design in [32]. Compared to the previous work, the True Random Number Generator (TRNG), the isolated sub-system, and the secure boot procedure are the three new things added. The main design idea is the heterogeneous architecture of a small and isolated 32-bit Micro-Controller Unit (MCU) with conventional 64-bit Linux-capable TEE processors. The hidden MCU will take care of the secure boot process and schedule the keys, thus achieving a flexible boot program with complete isolation from the TEE domain. At reset, the isolated MCU will boot first, do the first authentication, and then generate subsequent keys salted with random numbers. After that, the TEE processors will be woken up to follow the conventional TEE boot flow. The main bus of the isolated sub-system is called the Isolated Bus (IBus), and it connects

to the conventional System Bus (SBus) via a master-only TileLink protocol [33]. Therefore, the MCU can access all the sub-modules in the System-on-Chip (SoC), but the TEE processors cannot access the IBus peripherals. As a result, the isolated sub-system is ideal for having secure storage exclusively for the boot program and the root key. On the TEE side, several crypto-cores are attached to the Peripheral Bus (PBus), such as 512-bit Secure Hashing Algorithm 3 (SHA3-512), 128/256-bit Advanced Encryption Standard (AES-128/256), Ed25519, and TRNG. The TRNG core used in this paper is based on the previous work [34]. Because the National Institute of Standard and Technology (NIST) standard requires the key generation and TRNG to be in the same environment [35], the peripheral that wraps the TRNG core was designed with two interfaces, one for the PBus and one for the IBus. In TRNG peripheral wrapper, the IBus interface has a higher priority than the PBus interface. Whenever the isolated core issues a command to the TRNG module, the IBus transaction will override the PBus transaction. After finishing the IBus command, the peripheral will reset the TRNG core to resume the PBus transaction. As a result, the TRNG module can be shared between TEE and isolated domains while maintaining security and performance.

To summarize, there are three main contributions of the proposed implementation:

- **TEE with isolated RoT.** The heterogeneous architecture was presented with a hidden MCU for the secure boot flow and key generation. Due to the hierarchy-bus isolation, the TEE processors cannot access the peripherals in the isolated sub-system, thus providing exclusive storage for the boot code and the root key. By taking the RoT and the secure boot process out of the TEE domain, the complete isolation between secure boot and TEE was made, and the secure boot program is flexible to adapt with future attack vectors.
- **Key management scheme.** A completed keys scheduling was presented. Combining with the proposed heterogeneous architecture, the secure boot flow was realized with a hardware RoT signed by the trusted hardware manufacturer. Furthermore, hardware manufacturers and OS providers can securely update their boot program and TEE model after the chip was fabricated. Finally, an end-user can attest the device or chip's integrity even down to the silicon level.
- **Two-channel TRNG peripheral.** The two-channel TRNG implementation satisfies the NIST standard that requires the TRNG and key generation to be in the same environment. The TRNG core's randomness quality was proven not to be affected by the two-channel design [34], and the peripheral wrapper maintains the security of the isolated channel over the public channel.

The remainder of this paper is organized as follows. Section II reviews the related works. Section III gives an overview of the TEE hardware system. Section IV presents the isolation architecture. Section V describes the secure boot flow with the proposed key management scheme. Section VI shows the experimental results with comparison and discussion. Section VII gives a thought on the future work based on the current limitations. And finally, Section VIII concludes the paper.

II. RELATED WORKS

This section briefly explains the most recent security-driven RISC-V-based computer systems, including CURE [18], HECTOR-V [31], WorldGuard [30], and ITUS [28], [29].

A. CURE

CURE was proposed by Bahmani *et al.* [18]. It provides a strong enclaves isolation for TEE based on new hardware security primitives using RISC-V architecture. In CURE, we can have many types of enclaves coexisted in the same system (i.e., kernel-space, user-space, and sub-space). To achieve that, three main hardware modifications were added. They are the core registers for checking enclave execution, system bus arbitrator for controlling bus accesses, and shared cache partitioning. With three proposed hardware primitives, CURE can strengthen the isolation for TEE and assist in protection against Side-Channel Attacks (SCAs). The underlying mechanism is based on enclave IDs stored in the core's registers and propagated throughout the system. Therefore, we can identify which enclave is currently executed by which core. The ID values are set during the enclave's setup, teardown, and context switch. The bus arbitrator checks the access rights based on the enclave ID whenever a memory access request happens. If there is a violation, the transaction will be redirected to an unused, forbidding the transaction to continue. For the enclave-to-peripheral binding, since no party can access the memory without permission, no encryption or authentication is done on the communication between the enclave and the peripheral. For mitigating SCAs, two main methods were used, including L1 flushing and L2 cache partitioning using a way-based approach. For the RoT, CURE didn't do the RoT but assumed that the secure boot process was done on reset. And the first bootloader stored in ROM would verify and load the firmware, including the Secure Monitor (SM), to the designated Random Access Memory (RAM).

B. HECTOR-V

HECTOR-V was introduced by Nasahl *et al.* [31]. It is a RISC-V-based SoC tailoring for the isolation effect of the existing TEEs. Unlike CURE, HECTOR-V was not proposing a new TEE model; it enhances the security strength for the conventional TEEs at the architectural level. HECTOR-V design came from two main ideas, the heterogeneous multi-core architecture and the security-hardened RISC-V Secure CoProcessor (RVSCP). The RVSCP was designed exclusively for TEE with many SCA resilient features, and the heterogeneous design separated TEE and REE domains. Hence, the two processors of TEE and REE are coupled tightly together with many hardware primitives, enforcing strong isolation between them. The underlying mechanism of

HECTOR-V centers around the trusted I/O path design [31]. The trusted I/O paths were done by the ID-identifier in the communication fabric, thus making fine-grain protection between cores, peripherals, and Direct Memory Access (DMA) devices. Based on the IDs, every transaction is checked by the SM module, and any illegitimate access will be turned down. The core processor ID is permanently fixed in hardware, and the process and peripheral IDs are assigned at run time. Since the core IDs are hard-coded directly to the bus interface, no attacker could change those IDs. Due to the clean separation of TEE and REE processors, all cache-based and micro-architecture-based SCAs are prevented because no sensitive components (i.e., caches, branch predictors, and execution pipelines) are shared between the two domains. The key difference of HECTOR-V from the CURE is that a hardware module does the SM, and the concept of SM ownership is introduced. Therefore, the SM ownership can be dynamically transferred between participants, thus providing a variety of uses. Only one SM owner was allowed at a time, and only the SM owner could define the access rights of resources.

In HECTOR-V, the secure boot process was achieved by only allowing the first virtual core of TEE (VC0) to access the secure storage. The access right is permanently hard-coded and exclusively owned by the VC0. The other virtual TEE cores can only fetch the codes from the claimable Block RAM (BRAM). At reset, the reset unit configures the VC0 to be the SM owner while keeping the REE processors halted. Then, VC0 executes the Zero Stage BootLoader (ZSBL) in the secure storage, thus making the first authentication of the system; this is the RoT of HECTOR-V. Subsequently, ZSBL configures the Memory Protection Unit (MPU) for external memory access rights, like the Secure Digital card (SD-card) or Double Data Rate (DDR) memory. Then, VC0 compares the hash value of the Berkeley BootLoader (BBL) with the expected one in the secure storage. If the verification is successful, the BLL will be loaded into the main memory, and the VC0 will release the SD-card driver together with claimed DDR memory regions. Finally, VC0 transfers the SM owner over the REE processors and triggers the reset unit to start the REE. Compared to CURE, HECTOR-V has a clear secure boot procedure, while in CURE, the RoT is assumed by reset. Furthermore, HECTOR-V guarantees that the secrets stored in the secure storage are still protected after boot.

C. SIFIVE WorldGuard

WorldGuard [30] was developed by SiFive for strengthening the TEE isolation. It resembles CURE and HECTOR-V in many ways because it also relies on the IDs implementation. In WorldGuard, each core has a world ID, and each process on the core has a process ID. Similar to CURE and HECTOR-V, these IDs are propagated throughout the system, including cores, caches, buses, memories, peripherals, and DMA devices. The difference is that WorldGuard leverages the Physical Memory Protection (PMP) and Physical Memory Attributes (PMA) in the RISC-V Instruction Set Architecture (ISA), while CURE and HECTOR-V are not. Furthermore, HECTOR-V has an exclusive processor reserved only for TEE, while the TEE and REE share the same processors in WorldGuard.

Regarding the secure boot process, WorldGuard uses a similar approach with HECTOR-V, storing the first bootloader and root keys in ROM at the time manufactured. At reset, the first bootloader verifies and loads the SM into RAM for further processing. The difference is that because WorldGuard didn't have an exclusive processor for the boot process, no secure storage was exclusively assigned for the boot procedure.

D. ITUS

ITUS is a RISC-V-based SoC aiming for a secure boot process with RoT in TEE. The new hardware primitives were first introduced by Kumar *et al.* in [28], and then the complete secure boot procedure was implemented by Yajya *et al.* in [29]. Compared to the implementations mentioned above, ITUS is not proposing a new TEE model like CURE or focusing on isolating environments like HECTOR-V and WorldGuard. It attempted to solve the RoT problem in TEEs by a pure hardware approach using two modules named Key Management Unit (KMU) and Code Authentication Unit (CAU). Together, KMU and CAU provided a secure boot flow entirely out of the TEE processor's touch.

In the secure boot flow, CAU was used for verifying the CoT integrity based on the Elliptic Curve Digital Signature Algorithm (ECDSA) and SHA3 accelerators. The key generation was done in KMU using PUF and TRNG circuits. At reset, the boot sequencer, a Finite State Machine (FSM), activates the KMU to retrieve the root key, then passes the root key to the CAU to generate the subsequent asymmetric keys. Finally, if the BBL authentication successes, the boot sequencer will wake up the TEE cores. The drawback of this implementation is that the secure boot procedure is not flexible due to the fixed hardware solution. Furthermore, because all the cryptographic functions used in the boot process need to be realized in hardware, the cost of the resources will increase significantly together with the complexity of the chosen cryptographic functions.

III. RISC-V-BASED TEE HARDWARE SYSTEM

The proposed heterogeneous architecture is given in Figure 1. This design is a continuous version of the one presented in [32]. The isolated 32-bit MCU and TRNG are the two main differences compared to the previous architecture [32]. The system was created by using a hardware generator written in the Chisel library [36]. The integration was performed based on the Chipyard framework [37]. The Chisel-based generator was proven to be able to generate Register-Transfer Level (RTL) codes that are friendly with Very Large Scale Integration (VLSI) implementations [38], [39]. It can also

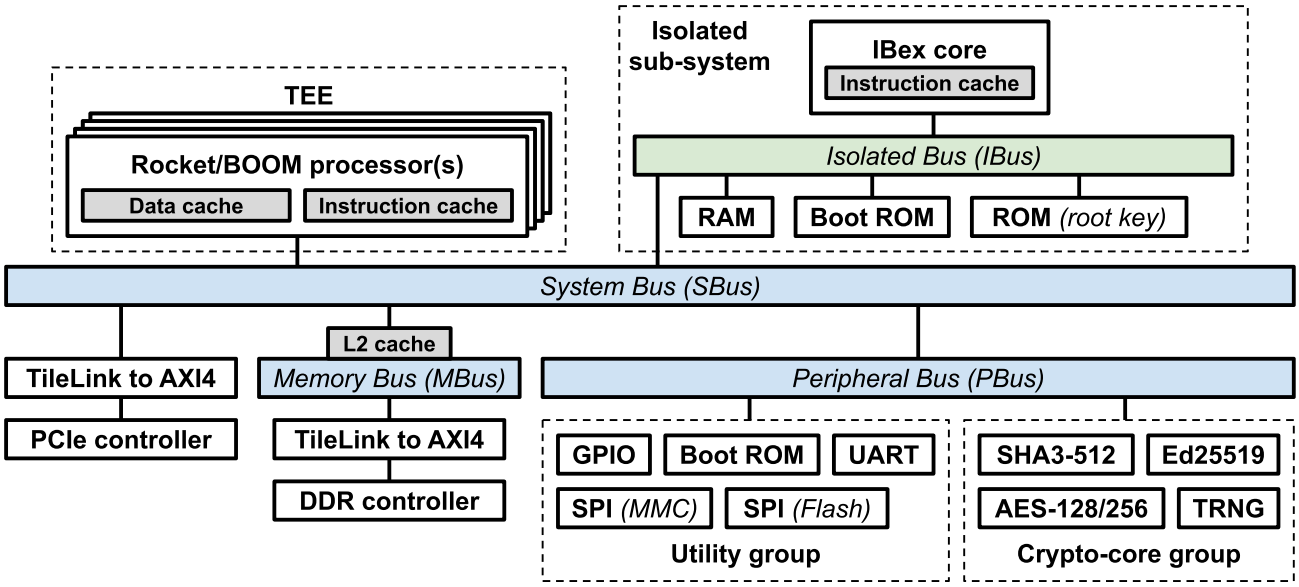

FIGURE 1. The proposed TEE hardware architecture.

create file artifacts such as testing scripts and Device Tree Source (DTS) files.

To increase the flexibility of the TEE hardware framework, several properties of the design were kept as optional and can be reconfigured easily depend on specific requirements. For example, the number of cores and the type of cores can be changed. Currently, there are two types of cores available, Rocket and BOOM [40], [41]. The ISA can be selected from these four options of RV64GC, RV64IMAC, RV32GC, and RV32IMAC [42]. The sizes of L1 and L2 caches can be changed. The Peripheral Component Interconnect express (PCIe) connection, the whole isolated sub-system, and each crypto-core can be included or excluded depending on needs. The default configuration used in this paper is a dual-core system with the Rocket core first and the BOOM core second; each core has a 16-KB instruction cache and 16-KB data cache. By default, the ISA is RV64GC, the size of L2 cache is 512-KB, the PCIe controller is excluded, the isolated architecture is included, and all of the peripherals shown in Figure 1 are included.

The processors are structured into the design by the SBus. From the SBus, the PBus and Memory Bus (MBus) are attached. All the SBus, MBus, and PBus are TileLink [33]. The SBus also includes the option of having a coherence cache manager, the L2 cache. As shown in Figure 1, the PBus contains several peripherals that can be categorized into two groups of utility group and crypto-core group. The utility group consists of a General-Purpose In-Out (GPIO), a Universal Asynchronous Receiver-Transmitter (UART), a ROM for the boot program, a Serial Peripheral Interface (SPI) for using SD-card, and an SPI for using a flash device. All cryptographic accelerators needed for TEE are included in the crypto-core group, such as SHA3-512, AES-128/256, Ed25519, and TRNG.

For the OS memory space, the TEE hardware system offers a 1-GB DDR controller. This controller is driven by an Advanced eXtensible Interface 4 (AXI4) [43] bus connected to the MBus via a TileLink-to-AXI4 bridge. The DDR IP could be Altera's or Xilinx's depends on which FPGA is being used. For the VLSI implementation, the MBus signals can be exported to the outside like GPIO digital buffers. Therefore, the fabricated chip can be mounted on an FPGA to use that FPGA's DDR IP as its primary OS memory.

IV. ISOLATED ARCHITECTURE

Figure 2 shows the isolated 32-bit architecture side-by-side with the conventional 64-bit TEE system. The isolated sub-system contains a RISC-V-based RV32IMC IBex core [44]. The IBex was chosen because it is a small 32-bit core with tamper awareness. The main bus in the isolated architecture is a TileLink bus named IBus, and its peripheral contains a boot ROM for the secure Boot-Loader (sBL), a 16-KB RAM for working memory, and another ROM for storing the root key. Additionally, this isolated sub-system contains its own Platform Level Interrupt Controller (PLIC) and Core Local INTerrupt (CLINT). The isolated CLINT handles internal core-level interrupts for scheduling. Through the PLIC, the outside TEE processors can issue commands to the isolated core for attestation. Then, the IBex core handles the PLIC's interrupts through programs stored in its boot ROM. The connection from the IBus to the SBus is master-only. That means all the sub-modules below the SBus are accessible from IBus but not the other way round. Therefore, all the information processing inside the isolated sub-system is inaccessible from the TEE side due to the hierarchy-bus architecture. Hence, the isolated sub-system is ideal for storing the root key and the initial bootloader program, the sBL. Due to the complete

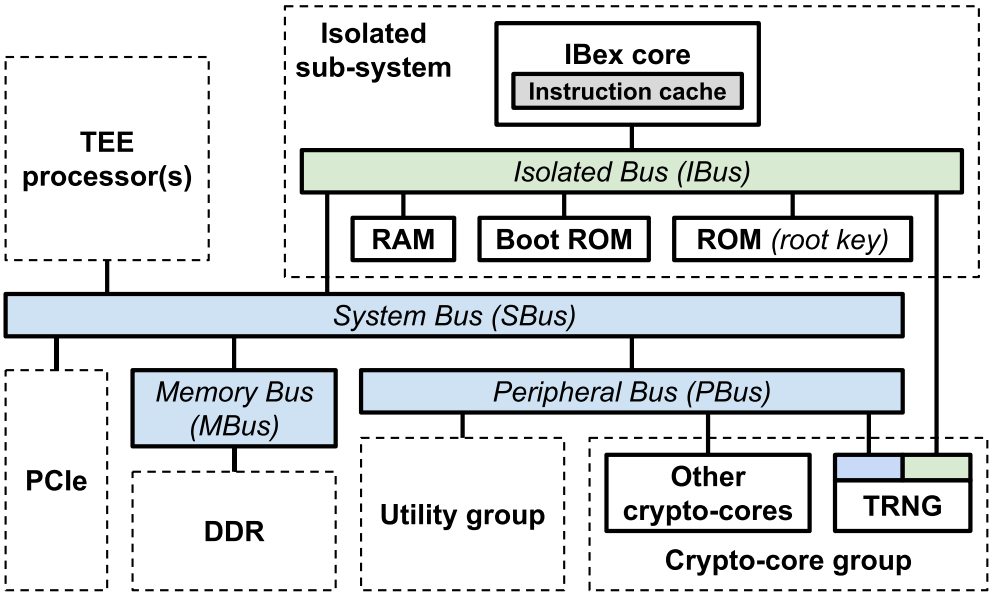

FIGURE 2. The isolated sub-system implementation.

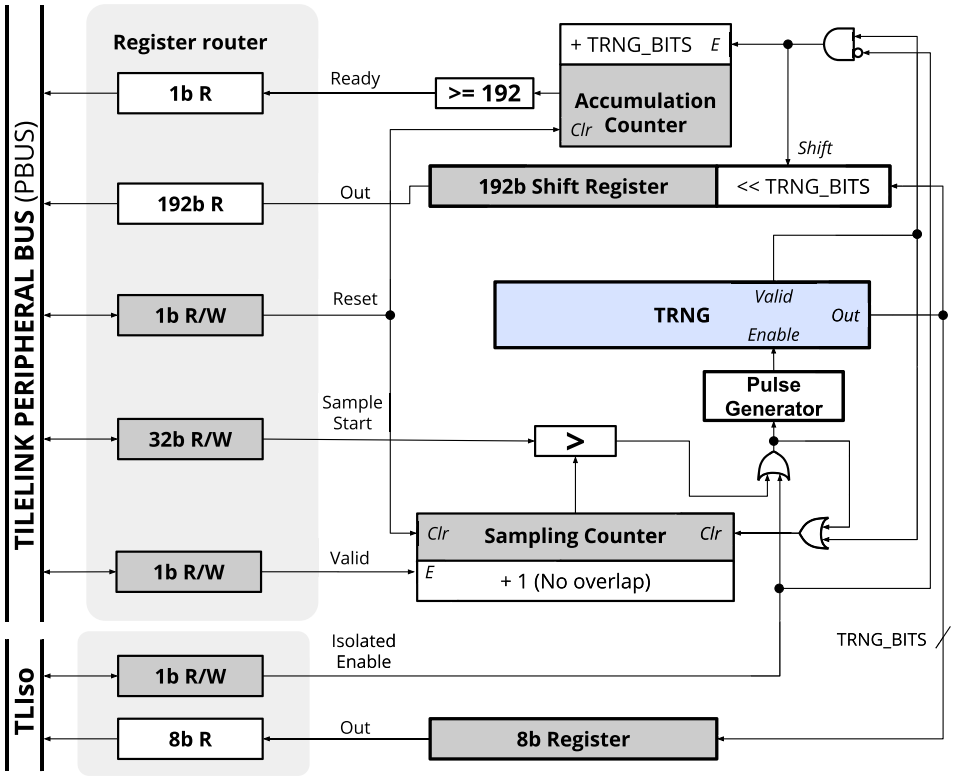

FIGURE 3. TRNG's peripheral with two interfaces.

isolation between the two domains, in the architectural point-of-view, the only potential threat from the public side is by having the TEE processors exploit the interrupt channel to request the IBex core for attestation. However, the IBex core only responds to the external interrupts based on its program written in the isolated boot ROM. Thus, we can modify the IBex's program to adapt with the attack vector if there is one.

Because the NIST standard requires the TRNG and keys attestation to communicate in the same environment [35], TRNG's peripheral was designed with two interfaces,

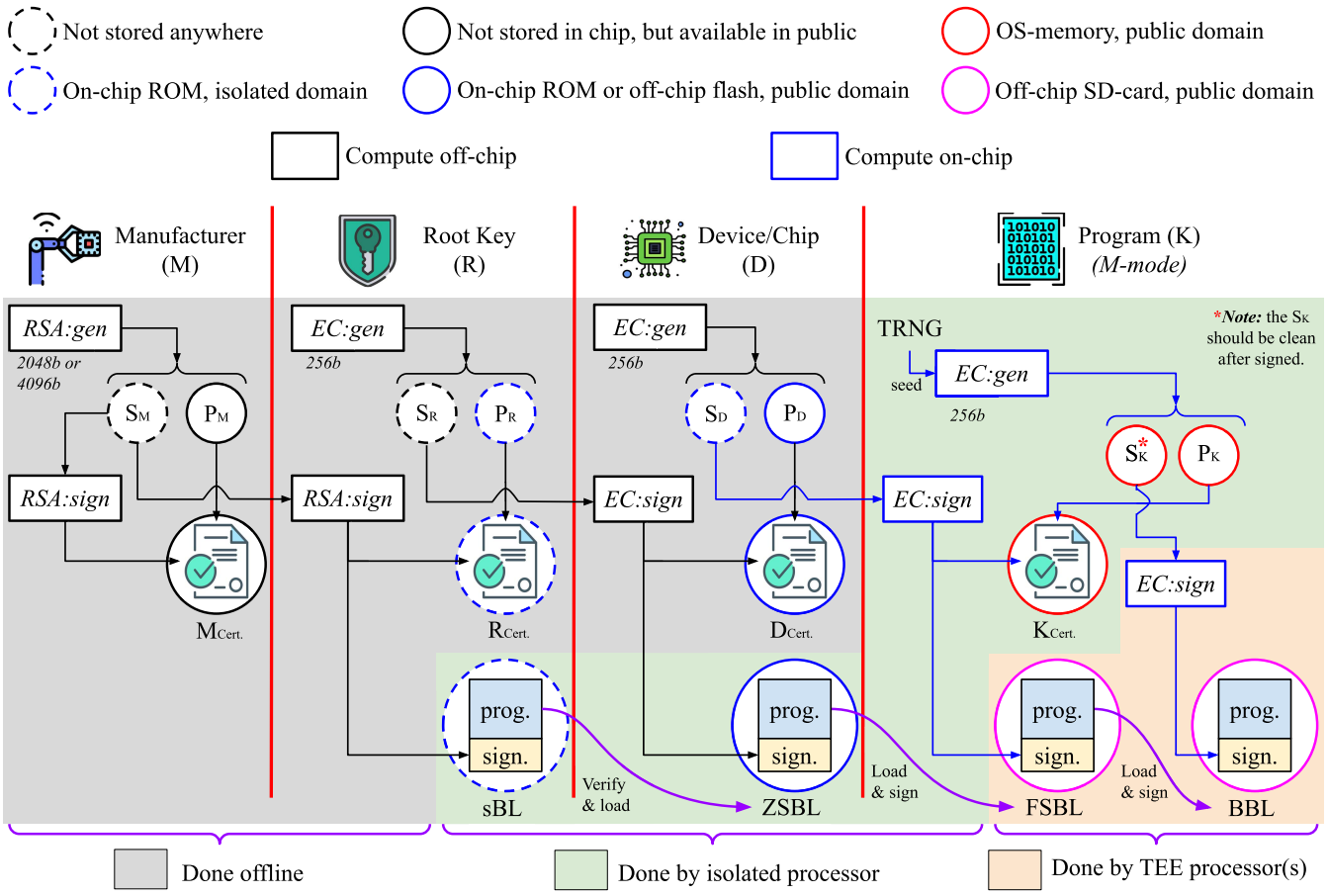

FIGURE 4. Proposed key management scheme with the secure boot process.

as shown in Figure 2, one with blue responding to the PBus, and one with green responding to the IBus. As a result, the IBex core has a direct connection to the TRNG module, and it could use the TRNG core without the risk of exposing data to the public domain. Figure 3 shows the TRNG module with two bus connections. In the figure, the TRNG core highlighted in blue comes from the previous work [34]. The generated random number can be extracted via any channel. However, the TRNG's peripheral is configured to respond with a higher priority for the IBus. As shown in Figure 3, if the *isolated enable* is activated, the accumulation process will halt and prevent the shift-register from accumulating the result. When the TRNG finishes its IBus transaction, it will self-reset and then resume the PBus transaction if there was one. Because the commands that come from the two channels are not treated as equals, the TRNG's outputs are classified as non-independence and Identically Distributed (non-IID) data in this case. The TRNG core was proven to pass the NIST non-IID restart test [34]. As a result, the two-channel TRNG's peripheral did not affect the quality of generated random numbers.

V. KEY MANAGEMENT AND SECURE BOOT FLOW

Figure 4 shows the key management scheme with the proposed secure boot process and TEE boot flow done by the isolated processor and TEE processors, respectively. As seen in the figure, there are four tiers of key pairs, including manufacturer, root, device or chip, and program. The pair of (S_M, P_M) identifies the trusted manufacturer. The pair of (S_R, P_R) stands for a series of products. The pair of (S_D, P_D) stands for one product. And the pair of (S_K, P_K) is generated during boot for the TEE at machine-mode (M-mode). Beyond the flow in Figure 4, there are subsequent keys generated by the TEE for the supervisor-mode (S-mode) and user-mode (U-mode), i.e., SM key pair (S_{SM}, P_{SM}) and enclave key pair (S_{Enc}, P_{Enc}), respectively. The idea behind Figure 4 is that the trusted manufacturer plays the role of a root CA; thus, its public key of P_M is well-known, and its certificate of $M_{Cert.}$ is self-signed. One manufacturer can have many key pairs, but each pair must be unique for each manufacturer. Because the (S_M, P_M) generation are done offline, they should be high-bit Rivest-Shamir-Adleman (RSA) [45] keys with many years of validity.

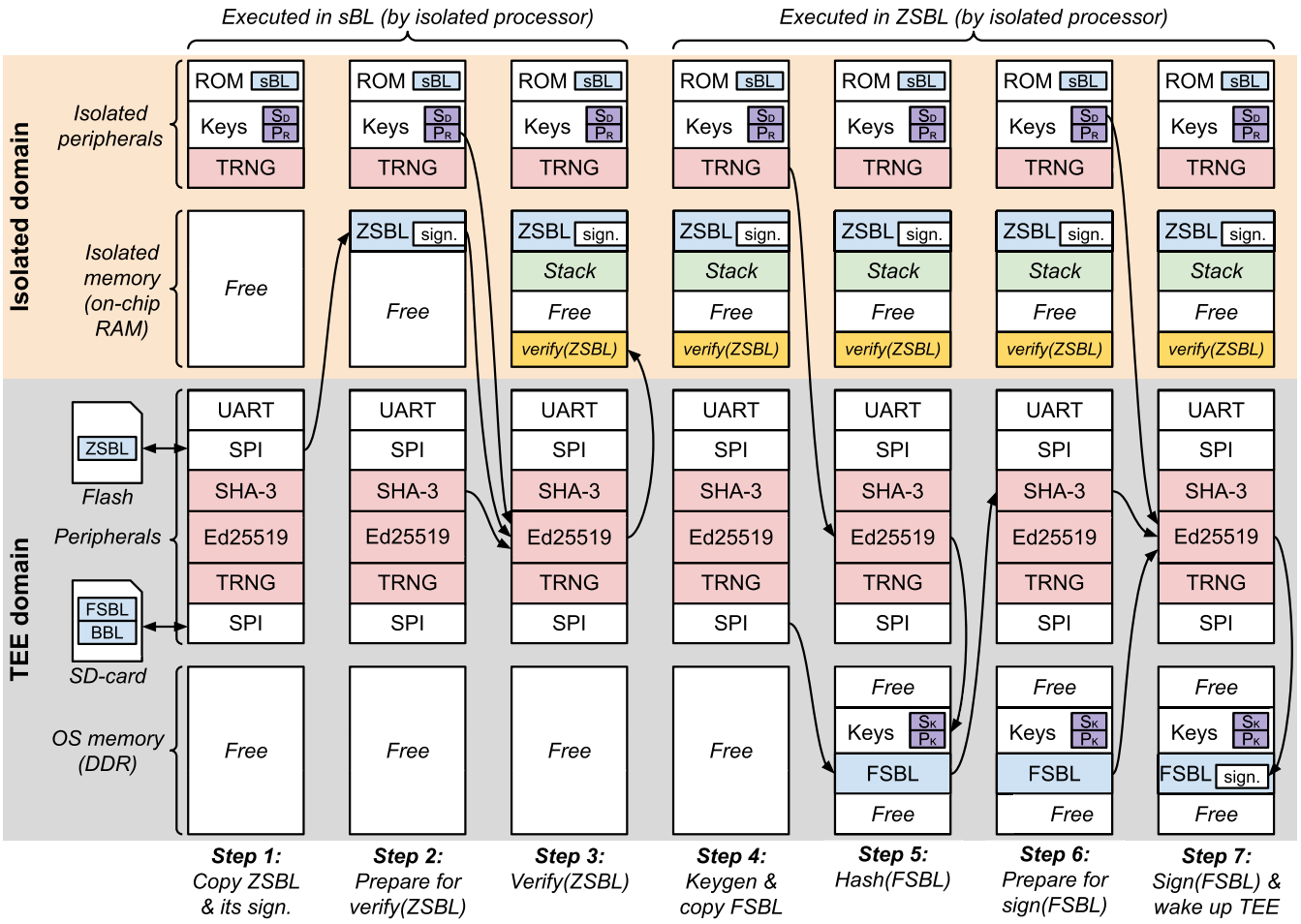

FIGURE 5. Boot sequence in the isolated environment.

The subsequent root's keys of (S_R, P_R) are Elliptic Curve (EC) keys, Ed25519 in this case, created by the manufacturer at the design time. The root certificate of $R_{Cert.}$ is signed offline by the manufacturer's secret key S_M. The root's secret key S_R is not stored in the chip, but the root's public key P_R is stored in the isolated ROM for the ZSBL authentication. The sBL is stored in the same place with the P_R, the isolated ROM. The content in sBL is signed previously by the S_M, as shown in Figure 4. At reset, the IBex core executes the sBL, and its very first task is to verify and load the ZSBL using the given P_R; this is the RoT establishment.

Next is the device or chip's key pair generation of (S_D, P_D). As shown in Figure 4, they are also generated offline by the manufacturer or the TEE/OS provider. The device's secret key S_D is stored in the isolated ROM, and the device's public key P_D is stored in the public domain (i.e., on-chip ROM or off-chip flash). The ZSBL is stored in the same place as the P_D, and it has a signature pre-signed by the root's secret key S_R. Because the first act of the isolated processor is to verify and load the ZSBL, this scheme allows the manufacturer or the TEE/OS provider to update the ZSBL securely, even if the ZSBL is stored in public as in an off-chip flash.

After verified and loaded, ZSBL then uses TRNG as seed for the on-chip key generation, as seen in Figure 4. The generated key pair of (S_K, P_K) is stored in a public RAM on the TEE side. Then, the device's secret key S_D is used to sign on the program's public key P_K, thus creating the first TEE certificate of $K_{Cert.}$. The First Stage BootLoader (FSBL) content is then loaded from the SD-card to the TEE's main memory, hashed, and signed by the S_D. After this step, both program's key pair and FSBL are in the TEE's working memory, ready to be executed by the TEE processors. Finally, the isolated core wakes the TEE processors to follow the conventional TEE boot flow. Note that the program's secret key of S_K should be clean after use, as also stated in [17]; this is for TEE's own sake of integrity and confidentiality.

Figure 5 shows the boot procedure in the isolated environment. This flow runs once the system is in the reset state. At reset, the TEE processors are halted in a wait state. The isolated processor will boot first and execute the sBL in the isolated boot ROM. The sBL then starts to copy the ZSBL and its signature (from an off-chip flash in this example) to the IBex core's working memory; this is Step 1 in the figure. Then, the root's public key P_R, the ZSBL's program, and

TABLE 1. Hardware overhead of the proposed TEE SoC in Virtex-7 XC7VX485T FPGA.

		BOOM	Rocket	IBex*	TRNG	Ed25519		SHA3	AES	Total
						mul	sign			
Slices	Logic LUT	66,525	24,817	7,465	198	2,305	5,344	8,881	2,710	149,765
	Flip-Flop	44,520	12,312	3,253	21	3,767	4,630	2,825	2,860	99,411
	Total LUT	111,045	37,129	9,793	219	2,465	5,344	9,013	2,842	249,176
BRAM		62	63	12	0	4	0	0	0	283
DSP block		36	15	4	0	16	0	0	0	71
FPGA utilization (%)		22.86	7.64	2.02	0.0451	0.51	1.1	1.86	0.59	51.3
Area overhead	Rocket-based (%)	299.08	100	26.38	0.59	6.64	14.39	24.28	7.65	671.11
	Total-based (%)	44.57	14.9	3.93	0.0879	0.99	2.15	3.62	1.14	100

Including the isolated sub-system.

TABLE 2. Synthesis results of the proposed TEE SoC with ROHM-180nm process.

	Cell-count (NAND2)	Cell-area		Power			
		μm^2	%	Leakage (nW)	Dynamic (μW)	Total (μW)	%
Total system	666,957	63,163,554	100.00	8,725.42	2,121,323	2,121,332	100.00
BOOM	362,038	18,745,262	29.68	4,890.36	1,152,855	1,152,860	54.35
core	142,090	3,615,820	5.72	1,668.91	532,203	532,205	25.09
dcache	20,722	4,171,936	6.61	327.11	60,191	60,191	2.84
icache	170,942	9,955,331	15.76	2,549.61	501,332	501,335	23.63
fpu	18,103	434,456	0.69	203.45	31,607	31,607	1.49
Rocket	94,663	6,673,530	10.57	1,138.11	332,103	332,104	15.66
core	23,290	593,378	0.94	288.90	67,958	67,959	3.20
dcache	10,584	2,773,067	4.39	138.99	30,684	30,684	1.45
icache	17,848	2,283,889	3.62	260.58	49,793	49,793	2.35
fpu	37,864	880,321	1.39	368.97	166,620	166,620	7.85
IBex*	25,508	2,641,567	4.18	265.06	45,291	45,291	2.14
core	22,113	1,234,939	1.96	203.98	32,507	32,508	1.53
L2 cache	41,072	30,663,467	48.55	648.99	123,889	123,889	5.84
Ed25519	58,559	2,336,819	3.70	783.50	240,425	240,425	11.33
sign	25,380	630,950	1.00	311.78	65,797	65,797	3.10
mul	26,464	1,488,695	2.36	341.72	154,598	154,598	7.29
SHA3	26,130	650,604	1.03	276.19	31,792	31,792	1.50
AES	16,325	412,932	0.65	205.29	37,515	37,515	1.77
BootROM	7,465	120,438	0.19	47.71	2,889	2,889	0.14
TRNG	268	3,984	0.0063	1.76	133	133	0.0063

Including the isolated sub-system.

the ZSBL's signature are fetched to Ed25519 crypto-core, as seen in Step 2. Together with the usage of SHA-3, Ed25519 verifies the ZSBL's signature and writes the result back to the isolated RAM, as shown in Step 3. Depending on the *verify*(*ZSBL*) result, the IBex core will jump to the ZSBL program if the verification is valid. In Step 4, the TRNG's private channel is used to get a random value. The generated random number is then fed to the Ed25519 as a seed to generate the next key pair of (S_K, P_K). In Step 4, the content of FSBL is also loaded to the OS main memory. In Step 5, the FSBL is hashed by the SHA-3 crypto-core, and its hash result is delivered to the Ed25519. In Step 6, the device's secret key of S_D is written to the write-only register in the Ed25519 crypto-core. This write-only register of Ed25519 is not exposed to the peripheral's register map. Therefore, it can not be read by any party, and it can be used only for the subsequent processing in the Ed25519 [32]. Finally, in Step 7, Ed25519 generates the FSBL's signature and wakes up the TEE processors to follow the conventional TEE boot flow. From this point forward, the procedure of loading BBL and setting up SM using crypto-cores is the same as described in [32].

VI. EXPERIMENTAL RESULTS

A. FPGA AND VLSI IMPLEMENTATIONS

The proposed TEE SoC in Section III is implemented in both FPGA and VLSI with the default configuration. That means the TEE processors are the dual-core of Rocket and BOOM with the ISA of RV64GC; each core has 16-KB for each instruction cache and data cache. The size of the L2

TABLE 3. Dhrystone tests comparison.

Core	ISA	Dhrystone/s	DMIPS/MHz
Rocket	RV64GC	150,511	1.713
	RV32IMC	138,197	1.573
IBex	RV32IMC	38,165	0.434

cache is 512-KB. The isolated sub-system is included, and the IBex core has a 4-KB instruction cache. The PCIe connection is excluded. And all modules in the utility and crypto-core groups, as shown in Figure 1, are included.

The chosen FPGA is the Virtex-7 XC7VX485T Xilinx FPGA, and Table 1 gives the resources utilization by total and by different parts of the system. The entire design occupies 51.3% of the FPGA resources, and nearly half of it is because of the BOOM core with 22.86%. The 7.64% FPGA resources are for the Rocket core, almost one-third compared to the BOOM core. The whole isolated sub-system costs 2.02% FPGA resources, nearly a quarter compared to the Rocket-core. The crypto-cores need a very few FPGA resources with 0.0451%, 0.51%, 1.1%, 1.86%, and 0.59% for the TRNG, Ed25519 base-point multiplication, Ed25519 sign, SHA3-512, and AES-128/256 modules, respectively.

For the VLSI design, the TEE SoC is also synthesized by a typical bulk process; the ROHM-180nm process library was chosen, and Table 2 gives the results of the system together with its submodules. The synthesis results are reported using Cadence's Genus tool version 18.13. According to the table, the L2 cache costs the most die area with 48.55%, nearly half of the chip, while consuming just 5.84% total power. For the most power consumption, the BOOM tile, including the core, two caches, and its floating-point unit, consumes 54.35%, more than half of the total power, while occupying only 29.68% of the chip area. The Rocket tile has a fair value of 10.57% and 15.66% for the total area and power consumption, respectively. The IBex tile is relatively small, with 4.18% and 2.14% for total size and power consumption. Compared to the Rocket tile, the area and power overheads of the whole isolated sub-system are 39.58% and 13.64%, respectively.

B. COMPARISON AND DISCUSSION

For comparing cores, the Drystone test is conducted for both IBex core and Rocket core, and Table 3 gives the experimental results. Because the ISA of the IBex is RV32IMC, the test on the Rocket core is repeated using the same executable. The Dhrystone test is run 500 times, and the average values are recorded. According to Table 3, the Rocket core achieves 1.573 to 1.713-DMIPS/MHz, a good result compared to an average processor [46]. The IBex core scores 0.434-DMIPS/MHz, which falls into the mid-range MCUs [47]. The DMIPS/MHz results of the Rocket core are about 3.62× to 3.95× compared to that of the IBex core.

Table 4 shows the FPGA results compared to recent similar implementations, and Table 5 gives the comparison regarding the security features. For more details on ITUS, WorldGuard, HECTOR-V, and CURE architectures, please refer to the Section II.

In ITUS [28], [29], they tried to solve the secure boot problem in TEE by a fixed hardware approach. The new hardware modules were introduced, including CAU and KMU. The KMU contains a PUF and a TRNG for key generation and distribution. The CAU handles program authentication by using an ECDSA and SHA-3 to sign and verify. Because their solution is based solely on hardware, their approach is not flexible. On the other hand, the IBex's program in our approach could do any cryptographic functions given in [29] and [28]. The crypto-cores in our system are not necessary for the secure boot; they just help accelerate the program. In contrast, due to the complete hardware solution, ITUS has to realize all cryptographic functions needed in the boot process, thus increasing the resources significantly when the complexity of those functions goes up. For the comparison in Table 4, if the IBex sub-system is compared to CAU or KMU, our work requires a smaller hardware cost. If the comparison includes the crypto-cores, our resources are approximately equal to ITUS with CAU+KMU.

In WorldGuard [30], a security scheme with IDs was implemented for solid isolation between secure and non-secure domains. The primary goal of the WorldGuard is to strengthen the existing TEEs, not to provide a secure boot process like our target. For the secure boot, WorldGuard utilizes the conventional boot flow with a root key and the first bootloader pre-stored in ROM at the time manufactured. The bootloader then verifies and loads the SM into the main working memory. Therefore, in WorldGuard, the RoT and the boot program are still in the TEE domain; hence, the attack surface still exists. Although WorldGuard opens its bootloader program, its hardware implementation is not opened or reported. Thus, we cannot include WorldGuard's size in the comparison table.

For HECTOR-V [31], the ultimate goal was to separate REE and TEE domains by introducing an SCA-resilience processor exclusively for TEE. The secure boot process is done using the TEE core. Therefore, HECTOR-V's boot program is flexible like our approach. However, similar to WorldGuard, the secure storage that contains the boot program and root key is still visible in the eyes of the TEE processor after boot. Thus, potential threats from malicious TEE software stacks could be realized in the future. Although HECTOR-V's set goal is different, we could still compare the secure boot's hardware requirements. Because HECTOR-V uses the exclusive TEE processor for the boot with no cryptographic accelerations, the cost of its resources will be compared with ours that includes only the isolated sub-system. The REE processor in HECTOR-V is a typical RV64GC Rocket core, while the 32-bit TEE processor could be a RI5CY, a REMUS, or a Frankenstein core. According to Table 4, our implementation consumes a lower cost compared to HECTOR-V with Remus or Frankenstein but a higher cost compared to HECTOR-V with RI5CY.

TABLE 4. Hardware overhead of the proposed TEE SoC in Virtex-7 XC7VX485T.

Design		Registers overhead (+%)		LUTs overhead (+%)	
This work (2021)	Baseline: Dual-Rocket	24,624		74,258	
	+ IBex[1]	+3,253	(13.21%)	+9,793	(13.19%)
	+ crypto-cores[2]	+14,103	(52.27%)	+19,883	(26.78%)
	+ IBex[1] + crypto-cores[2]	+17,356	(70.48%)	+29,676	(39.96%)
ITUS [28], [29] (2019)	Baseline: Dual-Rocket	24,624		74,258	
	+ CAU	+6,722	(27.30%)	+27,170	(36.59%)
	+ KMU	+3,344	(13.58%)	+29,529	(39.77%)
	+ CAU + KMU	+10,066	(40.88%)	+56,699	(76.35%)
HECTOR-V [31] (2021)	Baseline: Single-lowRISC	N/A		55,443	
	with RI5CY			+8,205	(14.80%)
	with Remus			+11,581	(20.89%)
	with Frankenstein			+13,303	(23.99%)

[1] *Including the isolated sub-system.*
[2] *Including SHA-3, AES, Ed25519, and TRNG.*

TABLE 5. Comparison with recent security-driven RISC-V-based TEE SoCs, regarding the security features. ●, ◐, and ○ rank the performance from best to worst, respectively.

	CURE [18]	HECTOR-V [31]	WorldGuard [30]	ITUS [28], [29]	This work
Secure boot with RoT	◐	●	◐	●	●
Flexible boot process	●	●	●	○	●
TEE & secure boot isolation	○	○	○	●	●
Exclusive TEE processor	◐	●	◐	○	○
Exclusive secure storage	○	●	○	●	●
Cryptographic accelerators	○	○	◐	●	●
SCA resilience	●	●	◐	○	○
Hardware cost	●	◐	●	○	◐
Low porting efforts	○	○	◐	●	●

In CURE [18], a new TEE model was proposed together with new hardware primitives for fine-tuning the security strength. The main goal of the CURE is a TEE model that can support multi-type of enclaves while maintaining strong isolation between them. To achieve that, new hardware modifications were added across the system at many architectural levels, such as registers in the core, access controller in the system bus, and way-based partitioning in the shared cache. Although CURE tailors the hardware for its proposed TEE, it considers the secure boot with RoT out of scope. In CURE, the RoT is assumed at reset, and the secure boot process is provided beforehand. As a result, besides security features, we cannot include the cost of the CURE in the area comparison Table 4.

Regarding the security-related features, Table 5 gives the comparison with recent security-driven RISC-V-based TEE SoCs. The comparison criteria are:

- **Secure boot with RoT.** Has a clear secure boot process with RoT.
- **Flexible boot process.** The boot process is flexible, and the boot program can be updated.
- **TEE and secure boot isolation.** The secure boot procedure is done by another party, not by the TEE processors. And the RoT is isolated from the TEE domain after boot.
- **Exclusive TEE processor.** TEE is executed by a dedicated processor, separated from the REE domain.
- **Exclusive secure storage.** The proposed system has an exclusive place to store the root key and the boot program.
- **Cryptographic accelerators.** Hardware accelerators for cryptographic functions are provided in the system.
- **SCA resilience.** The ability to prevent some SCAs.
- **Hardware cost.** The amount of resources the proposed architecture needs in terms of overhead ratio. Low is ∼5-10%, medium is ∼20%-25%, and high is ∼50%.
- **Low porting effort.** The development effort for adapting/porting the proposed design.

According to Table 5, the proposed architecture in this paper achieves a secure and flexible boot process with RoT that is completely isolated from the TEE domain. Furthermore, exclusive secure storage and cryptographic accelerators are also supported. Because the primary goal of this paper focuses on the secure boot procedure, the TEE-related security issues are considered out of scope. As a result, the proposed implementation lacks SCA resilience or exclusive TEE processor features, as shown in the table.

VII. LIMITATIONS AND FUTURE WORK

First of all, because the main focus of this paper is the secure boot process with isolated RoT, the TEE security-related issues such as SCAs are out of scope for this current version. Therefore, it is a natural development for this paper to combine with an SCA-focused implementation such as CURE or HECTOR-V. Then, the next version has not only a hidden MCU for the secure boot process but also SCA-resilence hardware primitives for strengthening TEEs.

The second update is also related to SCA. By using only TRNG for seeding the key generation, as shown in Figure 4, the system is subjected to certain TRNG fault attacks [48]. To overcome the issue, combining TRNG with a PUF could increase the security level [49]. Moreover, introducing a PUF to the system means adding more options for developers, thus making the overall architecture more versatile.

The next issue is about crypto-core options. Currently, the available crypto-cores are SHA3-512, AES-128/256, Ed25519, and TRNG. They are the minimum requirement for a crypto-system to work. Hence, more crypto-cores need to be developed in the near future to extend the options for developers, from the legacy crypto-functions, such as RSA [45] and Hashed Message Authentication Code (HMAC) [50], to the latest crypto-functions, such as ChaCha20 [51], Poly1305 [52], and Authenticated Encryption with Associated Data (AEAD) [53].

Finally, using an on-chip ROM to store the root key could limit the application's range. Therefore, if we could replace the on-chip ROM with an on-chip flash or One-Time Programmable (OTP) memory devices, the versatility of the proposed architecture would increase. For example, using OTP memory can program the S_D and P_R keys after taped-out, thus leading to unique keys for each chip. Similarly, using on-chip flash memory can re-program the keys for other purposes. The flash and OTP memories are just for adding options to the framework, not for security reasons.

VIII. CONCLUSION

In this paper, a TEE hardware implementation with an isolated sub-system was presented. The heterogeneous architecture was tested on a Virtex-7 XC7VX485T Xilinx FPGA and also synthesized with the ROHM-180nm process library. The chosen isolated core is the RV32IMC IBex with the anti-tampering feature. The isolated architecture contains a boot ROM, RAM, and another ROM for storing the root key. The connection from the isolated bus to the conventional systems bus is a master-only TileLink protocol. Therefore, the IBex core can use all of the modules in the system, but the TEE processors cannot access the isolated bus' peripherals. SHA3-512, AES-128/256, Ed25519, and TRNG modules are attached to the peripheral bus in the TEE domain. To satisfy the NIST standard, the TRNG's peripheral has two interfaces for the system and isolated buses. Via the isolated bus, the IBex core has direct access to the TRNG module with a higher priority. The FPGA implementation of the proposed SoC costs 249,176 slices with 9,793 slices for the isolated sub-system; the area overhead is 3.93%. For VLSI design, the ROHM-180nm synthesis results in 63.16-mm^2 and 2.12-W with 2.64-mm^2 and 45.29-μW for the isolated sub-system; the area and power overheads are 4.18% and 2.14%, respectively.

A complete key management scheme was also presented in this paper. The device or chip will have its RoT pre-signed by a trusted manufacturer at the time of fabrication, thus allowing the end-user to attest device integrity down to the silicon level. A complete boot flow was realized with the proposed keys scheduling scheme. At reset, the isolated processor goes first and does the first authentication to verify and load the ZSBL. Then, ZSBL loads the FSBL to the TEE's working memory, creates the FSBL's signature, and prepares the environment for the TEE processors. Finally, the isolated processor wakes up TEE processors to follow the conventional TEE boot flow. The proposed design achieved a secure boot process with isolated RoT. Furthermore, exclusive secure storage and cryptographic accelerators are available in the system, the boot flow is flexible, and the boot program can be updated for future attack models.

ACKNOWLEDGMENT

This work was supported through the activities of VDEC, The University of Tokyo, in collaboration with Cadence Design Systems and Mentor Graphics.

REFERENCES

[1] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 57–64.

[2] S. Sau, J. Haj-Yahya, M. M. Wong, K. Y. Lam, and A. Chattopadhyay, "Survey of secure processors," in *Proc. Int. Conf. Embedded Comput. Syst., Architectures, Modeling, Simulation (SAMOS)*, Jul. 2017, pp. 253–260.

[3] Intel Corporation. *Intel Software Guard Extensions (Intel SGX) Developer Guide*. Accessed: Apr. 17, 2022. [Online]. Available: https://download.01.org/intel-sgx/linux-1.7/docs/Intel_SGX_Developer_Guide.pdf

[4] V. Costan and S. Devadas, "Intel SGX explained," Cryptol. ePrint Arch., MIT Comput. Sci. Artif. Intell. Lab., 32 Vassar St, Cambridge, MA, USA, Tech. Rep., 2016/086, Jan. 2016. [Online]. Available: https://eprint.iacr.org/2016/086.pdf

[5] V. Costan, I. Lebedev, and S. Devadas, *Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture*. La Habra, CA, USA: Now Foundations and Trends, Jul. 2017.

[6] C. Victor, L. Ilia, and D. Srinivas, *Secure Processors Part II: Intel SGX Security Analysis and MIT Sanctum Architecture*. La Habra, CA, USA: Now Foundations and Trends, Jul. 2017. [Online]. Available: https://people.csail.mit.edu/devadas/pubs/part_2.pdf

[7] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *Proc. USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Broomfield, CO, USA, Oct. 2014, pp. 267–283.

[8] *ARM Security Technology: Building a Secure System using TrustZone Technology*, document PRD29-GENC-009492C, ARM, Apr. 2009. [Online]. Available: https://documentation-service.arm.com/static/5f212796500e883ab8e74531?token=

[9] S. Pinto and N. Santos, "Demystifying arm TrustZone: A comprehensive survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–36, Nov. 2019.

[10] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *Proc. Symp. Operating Syst. Princ. (SOSP)*, Shanghai, China, Oct. 2017, pp. 287–305.

[11] R. Buhren, C. Werling, and J.-P. Seifert, "Insecure until proven updated: Analyzing AMD SEV's remote attestation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, London, U.K., Nov. 2019, pp. 1087–1099.

[12] D. Kaplan. (Feb. 2017). *Protecting VM Register State With SEV-ES*. [Online]. Available: https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20%State%20with%20SEV-ES.pdf
[13] (Jan. 2020). *AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More*. [Online]. Available: https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf
[14] Hex Five Security. *MultiZone Hex-Five Security*. Accessed: Apr. 17, 2022. [Online]. Available: https://hex-five.com/
[15] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Secur. Symp. (USENIX Security)*, Aug. 2016, pp. 857–874.
[16] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "TIMBER-V: Tag-isolated memory bringing fine-grained enclaves to RISC-V," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, Feb. 2019, pp. 1–15.
[17] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proc. 15th Eur. Conf. Comput. Syst. (EuroSys)*, Apr. 2020, pp. 1–16.
[18] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "CURE: A security architecture with customizable and resilient enclaves," in *Proc. USENIX Secur. Symp. (USENIX Security)*, Aug. 2021, pp. 1073–1090.
[19] J. Noorman, J. V. Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling, "Sancus 2.0: A low-cost security architecture for IoT devices," *ACM Trans. Privacy Secur.*, vol. 20, no. 3, pp. 1–33, Aug. 2017.
[20] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, "Providing root of trust for ARM TrustZone using on-chip SRAM," in *Proc. 4th Int. Workshop Trustworthy Embedded Devices (TrustED)*, 2014, pp. 25–36.
[21] Y. Qin, J. Liu, S. Zhao, D. Feng, and W. Feng, "RIPTE: Runtime integrity protection based on trusted execution for IoT device," *Secur. Commun. Netw.*, vol. 2020, pp. 1–14, Sep. 2020.
[22] E. M. Benhani, L. Bossuet, and A. Aubert, "The security of ARM TrustZone in a FPGA-based SoC," *IEEE Trans. Comput.*, vol. 68, no. 8, pp. 1238–1248, Aug. 2019.
[23] Intel Corporation. *Intel Active Management Technology (AMT) Developers Guide*. Accessed: Apr. 17, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/develop/documentation/amt-developer-guide/top.html
[24] ARM. *ARM Security IP: CryptoCell-700 Family*. Accessed: Apr. 17, 2022. [Online]. Available: https://developer.arm.com/ip-products/security-ip/cryptocell-700-family
[25] *Inside a Deeply Embedded Security Processor*. Accessed: Apr. 17, 2022. [Online]. Available: https://i.blackhat.com/USA-20/Wednesday/us-20-Buhren-All-You-Ever-Wanted-To-Know-About-The-AMD-Platform-Security-Processor-And-Were-Afraid-To-Emulate.pdf
[26] Rambus. *Security CryptoManager Provisioning*. Accessed: Apr. 17, 2022. [Online]. Available: https://www.rambus.com/security/provisioning-and-key-management/cryptomanager-infrastructure/
[27] LowRISC CIC. (2020). *OpenTitan*. [Online]. Available: https://github.com/lowRISC/opentitan
[28] V. B. Y. Kumar, A. Chattopadhyay, J. Haj-Yahya, and A. Mendelson, "ITUS: A secure RISC-V system-on-chip," in *Proc. 32nd IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Sep. 2019, pp. 418–423.
[29] J. Haj-Yahya, M. M. Wong, V. Pudi, S. Bhasin, and A. Chattopadhyay, "Lightweight secure-boot architecture for RISC-V System-on-Chip," in *Proc. 20th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2019, pp. 216–223.
[30] SiFive. *Securing the RISC-V Revolution*. Accessed: Apr. 17, 2022. [Online]. Available: https://www.sifive.com/technology/shield-soc-security
[31] P. Nasahl, R. Schilling, M. Werner, and S. Mangard, "HECTOR-V: A heterogeneous CPU architecture for a secure RISC-V execution environment," in *Proc. ACM Asia Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, May 2021, pp. 187–199.
[32] T.-T. Hoang, C. Duran, D.-T. Nguyen-Hoang, D.-H. Le, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Quick boot of trusted execution environment with hardware accelerators," *IEEE Access*, vol. 8, pp. 74015–74023, 2020.
[33] SiFive. (Aug. 2019). *SiFive TileLink Specication*. [Online]. Available: https://www.sifive.com/documentation/tilelink/tilelink-spec/
[34] R. Serrano, C. Duran, T.-T. Hoang, M. Sarmiento, K.-D. Nguyen, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "A fully digital true random number generator with entropy source based in frequency collapse," *IEEE Access*, vol. 9, pp. 105748–105755, 2021.
[35] E. Barker, A. Roginsky, and R. Davis, "Recommendation for cryptographic key generation," Nat. Inst. Standards Technol. (NIST), Gaithersburg, MD, USA, Tech. Rep. NIST.SP.800-133r2, 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133r2.pdf, doi: 10.6028/NIST.SP.800-133r2.
[36] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2012, pp. 1212–1221.
[37] University of California at Berkeley. (2020). *Chipyard: An Agile RISC-V SoC Design Framework With in-Order Cores, Out-of-Order Cores, Accelerators, and More*. [Online]. Available: https://github.com/ucb-bar/chipyard
[38] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach, "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 209–216.
[39] P. S. Li, A. M. Izraelevitz, and J. Bachrach, "Specification for the FIRRTL language," Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-9, Feb. 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html
[40] RISC-V Foundation. (2019). *Rocket Chip Generator*. [Online]. Available: https://github.com/chipsalliance/rocket-chip
[41] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd generation Berkeley out-of-order machine," in *Proc. Workshop Comput. Arch. Res. RISC-V*, May 2020, pp. 1–7.
[42] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The RISC-V instruction set manual, volume I: User-level ISA, version 2.0," Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html
[43] *AMBA AXI and ACE Protocol Specification*, document ARM IHI 0022D, ARM, 2013. [Online]. Available: https://developer.arm.com/architectures/system-architectures/amba/specifications
[44] LowRISC CIC. (2020). *Ibex RISC-V Core*. [Online]. Available: https://github.com/lowRISC/ibex
[45] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
[46] R. Longbottom. (Oct. 2021). *Dhrystone Benchmark Results On PCs*. [Online]. Available: http://www.roylongbottom.org.uk/dhrystone%20results.htm
[47] Stratify Labs. (May 20, 2019). *Dhrystone Benchmarking on MCUs*. [Online]. Available: https://blog.stratifylabs.co/device/2019-05-20-Dhrystone-Benchmarking-on-MCUs/
[48] H. Martín, T. Korak, E. S. Millán, and M. Hutter, "Fault attacks on STRNGs: Impact of glitches, temperature, and underpowering on randomness," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 2, pp. 266–277, Feb. 2015.
[49] S. Larimian, M. R. Mahmoodi, and D. B. Strukov, "Lightweight integrated design of PUF and TRNG security primitives based on eFlash memory in 55-nm CMOS," *IEEE Trans. Electron Devices*, vol. 67, no. 4, pp. 1586–1592, Apr. 2020.
[50] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, document RFC2104, Feb. 1997. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2104
[51] D. J. Bernstein. (Jan. 2008). *ChaCha, a Variant of Salsa20*. [Online]. Available: https://cr.yp.to/chacha/chacha-20080128.pdf
[52] D. J. Bernstein, "The Poly1305-AES message-authentication code," in *Proc. Int. Conf. Fast Softw. Encryption (FSE)*, Feb. 2005, pp. 32–49.
[53] Y. Nir and A. Langley, *ChaCha20 and Poly1305 for IETF Protocols*, document RFC8439, Jun. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8439

TRONG-THUC HOANG (Member, IEEE) received the B.Sc. and M.S. degrees in electronic engineering from the Ho Chi Minh University of Science (HCMUS), Ho Chi Minh City, Vietnam, in 2012 and 2017, respectively, and the Ph.D. degree in engineering from The University of Electro-Communications (UEC), Tokyo, Japan, in 2022. From 2012 to 2017, he was a Lecturer Assistant with HCMUS. From 2019 to 2020, he was a Research Assistant with UEC. From 2019 to 2022, he was a Research Assistant with the Cyber-Physical Security Research Center (CPSEC), National Institute of Advanced Industrial Science and Technology (AIST), Tokyo. Since April 2022, he has been an Assistant Professor with the Department of Computer and Network Engineering, UEC. His research interests include digital signal processing, computer architecture, cyber-security, and ultra-low power system-on-a-chip.

CKRISTIAN DURAN (Graduate Student Member, IEEE) received the B.Sc. degree in electronics and the M.S. degree in telecommunications from the Universidad Industrial de Santander (UIS), Bucaramanga, Colombia, in 2014 and 2017, respectively, where he is currently the Ph.D. degree in electronics engineering. He is also a Research Assistant at The University of Electro-Communications (UEC), Tokyo, Japan.

RONALDO SERRANO (Student Member, IEEE) received the B.Sc. degree in electronics from the Universidad Industrial de Santander (UIS), Bucaramanga, Colombia, in 2020. He is currently a Research Assistant with The University of Electro-Communications (UEC), Tokyo, Japan. His research interests include computer architecture, high-speed digital interfaces, and hardware for security.

MARCO SARMIENTO (Graduate Student Member, IEEE) received the B.Sc. degree in electronics from the Universidad Industrial de Santander (UIS), Bucaramanga, Colombia, in 2020. He is currently a Research Assistant with The University of Electro-Communications (UEC), Tokyo, Japan. His research interests include debugging and security for integrated systems.

KHAI-DUY NGUYEN (Student Member, IEEE) received the B.Sc. degree in electronics from the University of Science and Technology, The University of Danang, Danang, Vietnam. He is currently a Research Assistant with The University of Electro-Communications (UEC), Tokyo, Japan. His research interests include computer architecture and digital design.

AKIRA TSUKAMOTO received the M.S. degree in computer science from Columbia University, New York, USA. He currently works at the National Institute of Advanced Industrial Science and Technology (AIST). He has worked on products based on cell/B.E. and ARM. His main areas of interest include software engineering on a networks, operating systems, and system security, and he is enthusiastic regarding any technical development.

KUNIYASU SUZAKI (Member, IEEE) received the B.E. and M.E. degrees in computer science from the Tokyo University of Agriculture and Technology and the Ph.D. degree in computer science from The University of Tokyo, Tokyo, Japan. He is currently a Senior Researcher with the National Institute of Advanced Industrial Science and Technology (AIST) and a Researcher with the Technology Research Association of Secure IoT Edge Application Based on RISC-V Open Architecture (TRASIO). His research interests include security on CPU, operating systems, and hypervisor.

CONG-KHA PHAM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Sophia University, Tokyo, Japan. He is currently a Professor with the Department of Information and Network Engineering, The University of Electro-Communications (UEC), Tokyo. His research interests include the design of analog and digital systems using integrated circuits.