Towards Secure Runtime Customizable Trusted Execution Environment on FPGA-SoC

Yanling Wang, Xiaolin Chang, *Senior Member, IEEE,* Haoran Zhu, *Graduate Student Member, IEEE,* Jianhua Wang, Yanwei Gong, and Lin Li

***Abstract*—Processing sensitive data and deploying well-designed Intellectual Property (IP) cores on remote Field Programmable Gate Array (FPGA) are prone to private data leakage and IP theft. One effective solution is constructing Trusted Execution Environment (TEE) and its secure boot process on FPGA-SoC (FPGA System on Chip). This paper aims to establish Secure Runtime Customizable TEE (SrcTEE) on FPGA-SoC through the design of a novel secure boot scheme and the design of the following three components: 1) CrloadIP, which enforces access control on TEE applications deploying IP at runtime such that SrcTEE can alleviate threats from unauthorized TEE applications and then SrcTEE can be adjusted dynamically and securely; 2) CexecIP, which not only enables the execution of newly-installed IP cores without modifying the operating system of FPGA-SoC TEE, but also prevents insider attacks from executing IPs in SrcTEE; 3) CremoAT, which can provide the newly-measured SrcTEE state and establish a secure communication path between remote verifiers and SrcTEE. Our secure boot scheme supports refreshable root trust key, and assures the authenticity and integrity of boot codes during the SrcTEE booting process. We conduct a security analysis of SrcTEE and its performance evaluation on Xilinx Zynq UltraScale+ XCZU15EG 2FFVB1156 MPSoC.**

***Index Terms*—Field programmable gate array, intellectual property, secure boot, system on chip, trusted execution environment.**

I. Introduction

LARGE-SCALE data processing in cloud data centers advances the widespread deployment of hardware accelerators in Field Programmable Gate Array (FPGA) [1], Application Specific Integrated Circuit (ASIC) [2] and Graphic Processor Unit (GPU) [3]. Benefiting from its dynamic re-configurability, FPGA performs more flexibly and gains huge interest as an acceleration solution [1], [4]. Cloud Service Providers (CSP), such as Microsoft, Alibaba, Amazon and Huawei, are adopting FPGA with unmatched capabilities and intrinsic advantages into their infrastructures [5]. However, sensitive data and well-designed Intellectual Properties (IPs) of users are prone to privacy data leakage and IP theft [6] in clouds. For data privacy and IP protection, users prefer to develop IPs locally and deploy IPs in secure processing environments of cloud FPGA resistant to untrusted CSPs. Currently, there is no CSP providing cloud FPGA equipped with a secure processing environment, undermining remote users' confidence of current cloud FPGA services.

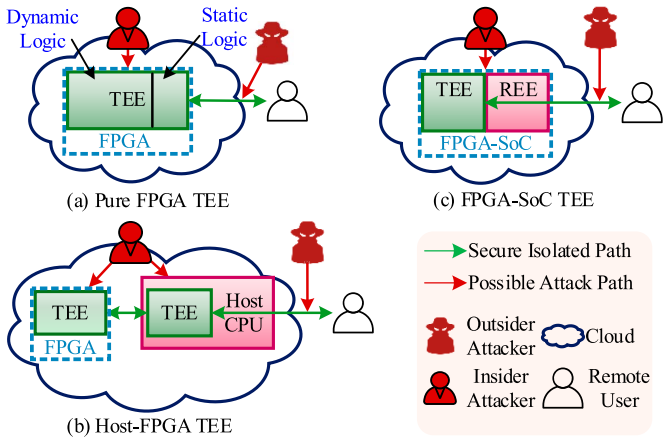

Fig. 1. FPGA-TEE solutions.

Trusted Execution Environment (TEE) addresses these privacy and security issues by providing a secure processing environment. That is, isolating data and IPs of users from that in a non-secure processing environment (referred to as Rich Execution Environment, REE) [7], [8]. TEE is originally designed for Central Processing Unit (CPU) but this purely CPU-based TEE has efficiency issues [9]. Researchers have explored provision of TEE on FPGA (denoted as FPGA-TEE) with rich hardware acceleration resources for accelerators developed by cloud users. Existing solutions can be classified into the following three categories, illustrated in Fig. 1.

(Solution_a) **Pure FPGA TEE**. Use a type of FPGA with no processing system integrated as independent TEE [10], [11], [12], [13], shown in Fig. 1(a). This type of solution divides TEE into static logic and dynamic logic on FPGA. Remote users communicate with the static logic, which is responsible for deploying IPs in the region of dynamic logic.

(Solution_b) **Host-FPGA TEE**. Use the whole FPGA as an extension of CPU-TEE [14], shown in Fig. 1(b). The Host-FPGA TEE system is formed by extending CPU-based TEE on a host to FPGA. The authors in [14], [15] explored extending

Manuscript received 5 July 2023; revised 23 December 2023; accepted 14 January 2024. Date of publication 22 January 2024; date of current version 13 March 2024. This work was supported by the National Natural Science Foundation of China under Grant 62272028. Recommended for acceptance by G. Di Natale. *(Corresponding author: Xiaolin Chang.)*

The authors are with Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, P.R. China (e-mail: 21120488@bjtu.edu.cn; xlchang@bjtu.edu.cn; 21112051@bjtu.edu.cn; jianhuawang@bjtu.edu.cn; 22110136@bjtu.edu.cn; lilin@bjtu.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2024.3355772, provided by the authors.

Digital Object Identifier 10.1109/TC.2024.3355772

CPU-TEE based on ARM TrustZone and Intel SGX for Host-FPGA TEE, respectively.

(Solution_c) **FPGA-SoC TEE**. Use part of FPGA-SoC (FPGA System on Chip) as FPGA-TEE and the left as REE [17], shown in Fig. 1(c). ARM TrustZone as processor-built-in security technology has been implemented on FPGA-SoC [16], [18], [19] for improving FPGA-SoC security. According to whether processor-built-in security technology is used, this type of solutions are further divided into two sub-categories: (c.1) no processor-built-in security technology is used [21], [22], [23] and (c.2) processor-built-in security technology is used [17], [20]. Note that both these two sub-categories have the same structure shown in Fig. 1(c).

There are at least three issues with Solution_a: (P1) A network server must run inside TEE for handling remote user requests, which requires an open service port and then increases the attack surface of FPGA-TEE. (P2) Static logic occupies part of hardware acceleration resources at all times even when it is not used, which reduces the effective utilization of FPGA resources. (P3) Since static logic and dynamic logic are not isolated from each other, a compromised or malicious static logic can easily cause threats to dynamic logic.

There are functionality and security issues with existing Solution_b: (P1) FPGA-TEE proposed in [15] only provided the IPs deployed at boot time for remote cloud users, but did not replace or add new IPs at runtime. (P2) FPGA-TEE proposed in [14] executed a virus scanning on decrypted IPs in CPU-TEE, which required remote cloud users to trust both CSP and FPGA vendors. For Solution_c.1, the execution space of FPGA-TEE is not separated from that of REE due to the lack of hardware isolation based on processor-built-in security technology, and then is susceptible to all security attacks from REE.

Solution_c.2 can address the above-mentioned problems. The authors in [17] and [20] utilized ARM TrustZone based Open Portable Trusted Execution Environment (OP-TEE) to construct FPGA-SoC TEE, combined with secure boot. Secure boot in this paper represents the boot process which involves establishing a chain of trust initiated from hardware root of trust to ensure TEE boot time integrity [45]. But there are at least two security issues with [17] and [20].

1) Use non-updatable root trust key. They both realized secure boot relying on non-updatable root trust key, which caused security risk and lack of maintainability (defined at the end of Section III.A).
2) Miss flexible, secure and trusted deployment and invocation of IPs. Gross et al. [20] failed to achieve runtime customizing of TEE with sole encryption protection of IPs. Khan et al. [17] allowed deploying IPs in TEE but caused increased design complexity and attack surface. Meanwhile, in [17], no access control measures were enforced on the TEE application when this application installs a new IP at runtime, which may cause threats to the TEE system. Section III.B details the difference of our work from [17].

The above discussions motivate our work on FPGA-SoC. In this paper, we propose a SrcTEE approach to construct *S*ecure *R*untime *C*ustomizable FPGA-SoC *TEE* (**SrcTEE**), which is characterized by the trust from boot time state to any runtime instant and also characterized by securely installing and invoking a new IP at runtime. This approach includes a secure boot scheme and three components working at runtime. To the best of our knowledge, it is the first time to investigate the construction of maintainable and customizable FPGA-SoC TEE. Note that the proposed **SrcTEE** is only applicable to FPGA-SoC devices. The security requirements which SrcTEE can meet and the attacks which SrcTEE can resist are detailed in Section II.D and Section VI.A, respectively.

We summarize the main contributions as follows.

1) We propose a novel secure boot scheme for SrcTEE, which satisfies the following two requirements. The first is the security requirement of boot codes during the booting process. Through authentication based on Physical Unclonable Function and public key infrastructure, our scheme can assure that the FPGA-SoC boots up with untampered and authorized codes provided by a trusted party. The second is the support of infinite updates of root trust key in order to alleviate the threat of key leakage. This is achieved by programming it in battery-backed RAM of FPGA-SoC, allowing periodic key refresh. To the best of our knowledge, this is the first design of secure boot scheme for FPGA-SoC which can achieve both these security requirements.
2) We make an extension to OP-TEE for installing and invoking a new IP securely at runtime, without constraints of placement and routing on this IP. The extension consists of three major components, CrloadIP, CexecIP and CremoAT. CrloadIP enforces access control on TEE applications deploying IP at runtime such that SrcTEE can alleviate threats from unauthorized TEE applications and then can be adjusted dynamically and securely. CexecIP not only enables the execution of newly-installed IP cores without modifying the operating system of FPGA-SoC TEE, but also prevents insider attacks from executing IPs in SrcTEE. CremoAT can provide the newly-measured SrcTEE state and establish a secure communication path between remote verifiers and SrcTEE. To the best of our knowledge, we are the first to enforce access control on TEE applications when they deploy IPs at runtime, and the first to design a secure and unified IP invocation interface for all newly-installed IPs to reduce design complexity and attack surface.

Security of SrcTEE is analyzed on the resistance to cryptographic attacks, network attacks, and insider attackers. Our experiments and performance evaluation are conducted on Xilinx Zynq UltraScale+ XCZU15EG 2FFVB1156 MPSoC, indicating our approach's practicability.

The remainder of the paper is set up as follows: Section II and Section III introduce background and related works, respectively. Section IV describes the proposed SrcTEE approach. We detail the implementation in Section V followed by security and performance analysis about this work in Section VI. Section VII concludes this paper.

II. Background

This section presents the background for understanding the related components and concepts in this paper.

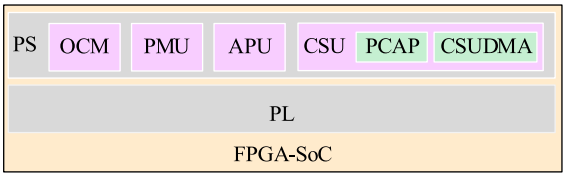

Fig. 2. FPGA-SoC Of ZU+ architecture.

A. FPGA-SoC of ZYNQ Ultrascale+ Platform

The Xilinx Zynq Ultrascale+ MPSoC family is based on the Zynq Ultrascale+ (ZU+) architecture. As presented in Fig. 2, this family of FPGA-SoC integrates programmable logic (PL) and a feature-rich processing system (PS) on a single device. The PL consists of configurable FPGA fabric, of which the functionality is determined by configuration memory. Data stored in the configuration memory is referred to as the partial bitstream or full bitstream. Full bitstream is generated from a complete hardware design with several IPs integrated while partial bitstream is usually generated from an IP of a hardware design. The units of PS used in our work include an ARM Cortex-A53 Application Processor Unit (APU), a Platform Management Unit (PMU) and a Configuration Security Unit (CSU) [18]. In this paper, we leverage CSU to achieve secure boot of the entire FPGA-SoC, including PS and PL. CSU incorporates hardware cryptographic accelerators (AES256, SHA3-384 and RSA4096), key management unit indispensable to secure boot, a built-in Direct Memory Access (DMA), dubbed as CSUDMA and Processor Configuration Access Port (PCAP) used for loading bitstream to PL via software. In short, CSU is responsible for booting PS in a secure or non-secure mode and for configuring PL. Besides a large external Double Data Rate (DDR) memory, FPGA-SoC contains 256 KB on-chip memory (OCM) which can be used for storing sensitive data or codes [24].

B. ARM TrustZone and OP-TEE

ARM TrustZone can be utilized to build TEE in PS and also can extend the security of TEE in PS to PL, which achieves TEE across PS and PL.

TEE in PS. ARM TrustZone for Cortex-A processors inside FPGA PS provides system-wide isolation between TEE and REE, which is achieved by partitioning both hardware and software resources. ARM Trusted Firmware (ATF) contains a Secure Monitor for managing the switching and communication between two TEE and REE. OP-TEE is a free-software framework to implement secure processing environment designed by Linaro [28], as a companion to a non-secure Linux kernel running on ARM Cortex-A cores using the ARM TrustZone technology. OP-TEE provides TEE Internal Core API and TEE Client API for developing applications. Normal applications (CAs) in REE can interact with trusted applications (TAs) in TEE via Secure Monitor. TA is identified by a unique UUID and is accessible to a CA developed with TEE Client API [28], [29].

Security extension to PL. The security of TEE can be extended to FPGA PL by setting the security bit in an AXI Interconnect IP. A master interface of AXI Interconnect can declare the AXI peripheral connected to its slave interface as a secure or non-secure IP. Two internal bits of the AXI peripheral, namely AWPROT for write transaction and ARPROT for read transaction, inform the full system that the software requesting read or write is running in TEE or REE. The MMIO (Memory-Mapped I/O) address area for input and output of an IP is defined during IP design, and cannot be changed except redesign the IP. Only TEE can access the MMIO address of a secure IP, while the area of a non-secure IP is accessible to TEE and REE [24], [30], [31].

C. Physical Unclonable Function

Physical Unclonable Function (PUF) is a hardware security technology which uses the manufacturing variation of a silicon nanocircuit to identify devices and derive entropy sources [27]. It is always used for FPGA authentication to prevent FPGA impersonation attack [11], and provides truly random seeds to generate key. Our paper adopts PUF to obtain truly random seeds for generating key pairs and authenticate FPGA-SoC [37].

D. Threat Model and Security Requirements

The most valuable assets in a cloud FPGA-SoC contain the private data and IPs developed by the cloud service user. Appropriate security measures must be taken to ensure that those assets are protected against attackers [11]. In this work, we focus on preventing attacks carried out through software since physical attacks utilizing hardware cannot be effectively defended by a device usage scheme alone. Explicitly, malicious attackers who have access to REE and the transmission channel between REE and the user are considered. There are at least the following three types of threats to FPGA-SoC TEE.

1) Obtain sensitive data or IPs of remote users in transmission between FPGA-SoC TEE and the user. Attackers may attempt to obtain sensitive data or bitstream of IP by deceiving the user with an impersonated device or breaking the cryptographic security of session key shared between FPGA-SoC TEE and the user.
2) Inject malicious IPs. Attackers inside or outside the cloud attempt to inject a malicious IP into PL, trigger it, and then access or tamper with the IPs or data created by users. Note that users are assumed to be trusted and then they do not inject malicious IPs in this paper.
3) Make unauthorized access to resources in TEE. Insider attackers can read out IPs in bitstream form via reconfiguration interfaces even when the bitstream is encrypted before the device being reconfigured with the bitstream [17]. They can also perform unauthorized and arbitrary invocation of IPs created by users. It is also possible that insider attackers attempt to obtain the device key (defined in Section III.A) and reverse engineering the encrypted bootable image.

Section VI.A details the attacks related to the threats above. To defend against these threats, SrcTEE at least needs to meet the following security requirements (**SR**).

1) **SR1: Verifiability and Maintainability of Secure Boot**. The cloud device is booted securely with encrypted bootable image involving integrity-checking

and authentication, which ensures untampered and authorized codes are executed at boot time. Note that the measurements of components in bootable image should be stored securely for later remote attestation and root trust key for secure boot can be refreshed infinitely.

2) **SR2: FPGA-SoC Authentication.** FPGA-SoC authentication ensures that the device in the cloud is a specific physical FPGA-SoC instead of an emulated hardware or software module.
3) **SR3: Secure and Trusted TEE Runtime Customizing.** If there is no access control enforced on the rights of deploying IPs, there exist unauthorized operations to IPs and then TEE security may be degraded. If the authenticity and integrity of bitstream used to customize TEE cannot be guaranteed, TEE may easily be compromised.
4) **SR4: Secure IP Invocation.** A secure and unified IP invocation interface should be developed and integrated in OP-TEE for any new IPs that may be deployed at runtime. Only authorized TA in TEE can invoke IPs and cloud employees cannot invoke IPs via REE exposed to them.
5) **SR5: Secure Transmission.** No plain text data or bitstream is accessible to entities other than FPGA-SoC TEE and the user who creates these assets. If any private keys for remote attestation of FPGA-SoC TEE and the user are exposed to any third party involved, the party can easily decrypt IPs and data encrypted with a session key shared between FPGA-SoC TEE and the cloud user.

III. Related Work

A. Secure Boot Schemes

This section presents four existing types of secure boot schemes. The first two are provided in the Xilinx ZU+ devices to protect boot files for PS and bitstream for PL: hardware root of trust mode (denoted as **mode_1**) and encrypt only mode (denoted as **mode_2**) [25]. **mode_1** uses a primary RSA key pair with the hash of primary public key programmed in eFuse to provide the authentication of each component in bootable image. **mode_2** utilizes an AES key programmed in eFuse to boot from an encrypted bootable image [25]. eFuse is a fixed and non-updatable memory while a physical battery is required for battery backed RAM (BBRAM) [26]. If a key or its hash is programmed in eFuse or BBRAM for secure boot, the key is the root trust key for a secure boot process, dubbed as device key in this paper. The ability of **mode_1** and **mode_2** to refresh device key is determined by resource-limited eFuse in ZU+ device. Therefore, both **mode_1** and **mode_2** are unable to update root trust key when valuable and extremely resource-limited eFuses in the device are exhausted. However, considering security and maintainability, it is generally preferable to periodically update the root trust key for secure boot.

The authors in [43] and [44] considered maintainability in secure boot of FPGA-SoC by applying advanced techniques such as Trusted Platform Modules (TPM) and PUF. The authors in [43] proposed a self-authenticating secure boot process to protect the encrypted operating system (OS) and applications. They first programmed bitstream of a PUF, read out configuration information of the bitstream, generated a digest for the information, and then used the digest as a challenge to obtain the response from the PUF. The response is the key to decrypt OS and applications, and thus malicious tampering with the bitstream changes the challenges and the corresponding decryption key, resulting in key regeneration failure. The authors in [44] adopted PUF and TPM to design a secure boot scheme. First, PUF was used to generate decryption keys for bitstream of hardware applications. These keys were stored in TPM to construct security features that can protect hardware application bitstream during the secure boot process and at runtime. However, malicious tampering with OS and applications [43] or application bitstream [44], cannot be detected effectively because they failed to ensure the integrity and authenticity of these boot codes. Moreover, additional hardware modules like TPM caused increased complexity and cost.

TABLE I
Comparison of Existing Secure Boot Solutions

Features	mode_1 [17], [20]	mode_2 [22], [23]	[43]	[44]	Ours
(F1) Integrity	√	√			√
(F2) Authenticity	√				√
(F3) Confidentiality	√	√	√	√	√
(F4) Easy-to-impl	√	√	√	√	√
(F5) Maintainability			√	√	√

It is noticed that it is hard, if not impossible, for any combination of **mode_1**, **mode_2**, and the secure boot schemes proposed in [43] and [44] to satisfy all the following features. However, our secure boot scheme achieves these features. Table I summarizes the comparison.

Integrity. The integrity of boot codes is validated before execution, avoiding the execution of malicious codes.

Authenticity. The boot codes are authenticated using RSA algorithm before decryption, which ensures that only authorized codes are executed.

Confidentiality. The boot codes in bootable image are protected with encryption, preventing plaintext codes and private information leakage.

Easy-to-impl. There are no additional hardware modules required in the secure boot process, which reduces complexity and cost.

Maintainability. There is no restriction on the number of times of refreshing the root trust key (device key) for secure boot of FPGA-SoC, defending against device key leakage.

B. FPGA-SoC TEE

There are researches on developing TEE for protecting user data and computation on FPGA. Pure FPGA TEE was developed by authors in [10], [11], [12], [13], the authors in [14], [15] explored the establishment of Host-FPGA TEE, and authors in [21], [22], [23] explored FPGA-SoC TEE without using ARM TrustZone. The rest of this section focuses on investigating the approaches to build FPGA-SoC TEE using ARM TrustZone.

Note that ARM TrustZone is the only processor-built-in security technology used in FPGA-SoC for building FPGA-SoC TEE so far. ARM TrustZone prevents insider attackers from compromising FPGA-TEE through the software and/or hardware on REE exposed to them. Therefore, approaches using TrustZone are more secure than those not using TrustZone.

There are case studies regarding utilizing ARM TrustZone technology to construct TEE on FPGA-SoC. Khan et al. [17] and Gross et al. [20] proposed Open Portable Trusted Execution Environment (OP-TEE) [28], which is ARM TrustZone based FPGA-SoC TEE. However, Gross et al. [20] did not consider extending trust from PS to PL, where IPs are deployed. That means IPs are not in the chain of trust and thus non-secure and untrusted. Security Framework (SFW) proposed in [17] is close to our work. Both SFW and our work construct FPGA-SoC TEE across PS and PL but with the following major differences:

Diff_1) Secure boot. SFW applies **mode_1** to realize a secure boot process for FPGA-SoC TEE. Although using a non-updatable root trust key can simplify the security boot process, it comes with potential security risks and lacks maintainability, increasing attack surface of FPGA-SoC TEE. Differing from SFW, we design a novel secure boot scheme allowing updatable root trust key. The scheme is designed in Section IV.C.1 and implemented in Section V.F. Note that our secure boot scheme can work with SFW.

Diff_2) Secure and trusted deployment of IPs at runtime. SFW allowed any TAs to deploy IPs at runtime, vulnerable to unauthorized IP injecting through an unauthorized TEE application. But in SrcTEE, only the authorized TA provided by Trusted Third Party (TTP) has the right of deploying IPs at runtime. This access control is enforced by **P3** module in Section IV.C.2, detailed in Section V.C. In addition, an IP under SFW cannot be deployed at runtime unless this IP meets specific constraints of placement and routing. SrcTEE puts no such constraints on an IP to be deployed at runtime, which is achieved by clearing bitstream configured at boot time before deploying IPs developed by USER at runtime (See **step 13** of Fig. 5).

Diff_3) Unified IP invocation. SFW did not discuss how to securely invoke IPs at runtime. Usually, the invocation of an IP is controlled directly by its specific IP driver in the kernel of OP-TEE. That is, a new driver should be developed for a new IP, increasing design complexity and attack surface. In our work, an address-based IP invocation API is designed in OP-TEE, through which each IP can be invoked. That is, a newly-deployed IP can be invocated without developing a new driver for this IP in OP-TEE. It is detailed in Section V.D.

Diff_4) Privacy of user's data and bitstream. The private key of SFW for protecting bitstream encryption key is generated by TTP, and then plaintext bitstream of IP developed by the user can be easily obtained by TTP. In our work, private keys of FPGA-SoC TEE and the user, and bitstream encryption key are never exposed to TTP, detailed in Section IV.C.4 and Section IV.C.5.

TABLE II
COMPARISON OF EXISTING WORKS AND OUR SRCTEE

Security Requirements	Not Use ARM TrustZone			Use ARM TrustZone		
	[21] 2019	[23] 2021	[22] 2022	[17] 2021	[20] 2022	Our SrcTEE
SR1	○	◖	◖	○	◖	●
SR2	●	●	●	○	○	●
SR3	○	○	●	◖	○	●
SR4	○	○	○	○	○	●
SR5	●	○	●	○	-	●

● full support ◖ partial support ○ no support - not applicable.

Note that our secure boot scheme can be adjusted to be compatible with SFW, but SFW remains improvement in secure provision of runtime deployment and invocation for IPs and bitstream protection. To this end, our work achieves secure customizable FPGA-SoC TEE with refreshable root trust key, smaller attack surface, lower design complexity, reduced privacy concerns and secure IP invocation. Table II summarizes these schemes in terms of security requirements defined in Section II.D.

IV. SRCTEE APPROACH

This section presents the system which uses SrcTEE in Section IV.A. Then Sections IV.B and IV.C, respectively, describe the SrcTEE architecture and its new components which enable the establishment of SrcTEE based on our proposed secure boot scheme.

A. System Description

The system includes three participants.

1) Cloud Service Provider (CSP). They earn profit by providing network-attached devices to remote users. They offer a webpage of online store for idle cloud devices with varying PL sizes and hard-wired functionalities.
2) Cloud User (USER). These users are the consumers of the CSP. They develop IPs locally and deploy these IPs on a leased cloud device for use.
3) Trusted Third Party (TTP). It plays the role of a trusted authority or is played by the FPGA vendor, which supports the process of verifying TEE on cloud devices as a neutral and trustworthy entity. Their services mainly comprise user enrollment, device enrollment, checking measurements sent by FPGA-SoC TEE, and providing users with an FPGA-SoC authentication credential when users request to authenticate the cloud FPGA-SoC.

Based on the proposed secure boot scheme, SrcTEE is able to support remote attestation, secure IP deployment and IP invocation at runtime. Fig. 3 illustrates the phases of customizing SrcTEE at runtime (See details of the phases in Section IV.C.5).

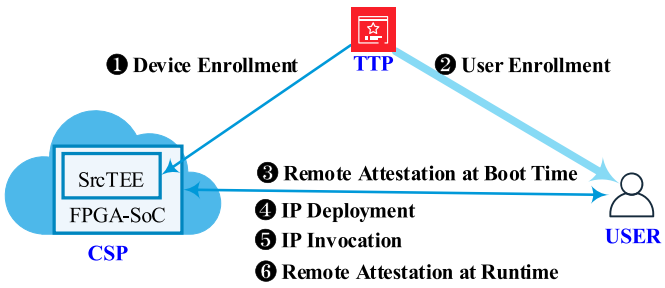

Fig. 3. Illustration of the phases of customizing SrcTEE at runtime.

With the help of TTP, CSP first registers device information and obtains a device-specific bootable image from TTP, and USERs register their information. FPGA-SoC can be booted from the device-specific bootable image to run OP-TEE in PS. USER can deploy multiple IPs at one time using full bitstream and also can update only an IP using partial bitstream. By interacting with SrcTEE and TTP, USER can verify the trustworthiness of SrcTEE, establish a secure communication path with it, deploy IPs, invoke IPs, and check the integrity of IPs at runtime.

The basic requirements and prerequisites for cloud devices, TTP and USER are described as follows:

1) **Assumption 1** for cloud devices. We assume a unique device identifier (*#DI*) is programmed in the device by FPGA vendors during manufacturing. This *#DI* is non-volatile, unchangeable and readily accessed. However, it is not adequate for uniquely identifying an FPGA-SoC in the cloud [11], [32].
2) **Assumption 2** for trust on TTP. The device key specific to secure boot is programmed in BBRAM of FPGA-SoC by TTP during device enrollment. The challenge-response pairs (CRPs) of PUF are pre-stored in a secure database of TTP for later remote attestation. The public key of TTP (PK_{TTP}) is stored in bootable image for secure boot. An RSA key pair (PK_{TA} and SK_{TA}) for authenticating TAs is generated by the TTP. And PK_{TA} is stored in OP-TEE for authenticating TAs. Therefore, only TAs signed by TTP can run on OP-TEE, which is referred to as TA authentication. Cloud employees can execute any TAs on the OP-TEE. Differing from SFW, we take measures to reduce the degree of trust placed in TTP, whose access to the encrypted bitstream and data is restricted at runtime.
3) **Assumption 3** for USER. The work of this paper focuses on ensuring the confidentiality and integrity of users' private assets on remote cloud devices. Therefore, we assume that the user host is equipped with secure execution environment where all critical applications and data are properly protected and the user's hardware design conforms to security setting. By security setting, all the IPs designed by users are secure IPs with two TrustZone specific control ports (Section II.A), and the master interface of AXI Interconnect in user's hardware design declares the AXI peripheral connected to its slave interface as a secure IP. Assumption 3 is the premise that private assets of the user can be protected by SrcTEE.
4) **Assumption 4** for communication channel. A secure communication channel between the user and TTP is available.

B. SrcTEE Architecture

SrcTEE aims to secure the private data and IPs developed by USER, suggesting that it must meet **SR1**-**SR5** given in Section II.D. The following lists the challenges for achieving **SR1**, **SR3**, and **SR4**:

1) **Challenge to SR1.** Although **mode_1** achieves authenticity and integrity by programming hash of root trust key in eFuse, it lacks maintainability. The secure boot schemes proposed in [43] and [44] achieved maintainability by adopting PUF or/and TPM but they cannot satisfy authenticity and integrity. Moreover, it is hard, if not impossible, for any combination of **mode_1**, **mode_2**, and the secure boot schemes proposed in [43] and [44] to meet SR1. How to achieve authenticity and integrity without damaging maintainability is a challenge.
2) **Challenge to SR3.** It is important to keep REE from deploying IPs in TEE while enforcing access control on TAs deploying IP. SFW proposed in [17] can only control REE but there is no access control on TEE applications, which may cause unauthorized operations on IPs and then cause security damage. How to enforce security access control on both REE and TAs for secure and trusted TEE runtime customizing is a challenge.
3) **Challenge to SR4**. In OP-TEE and the existing works on FPGA-SoC TEE, one driver for one underlying IP is adopted to support IP invocation of OP-TEE kernel. Such invocation method requires the development of a new driver for the corresponding newly-installed IP in order to enable the IP invocation. How to avoid the necessity of developing a new driver whenever installing a new IP in TEE at runtime is a challenge.

SrcTEE approach makes SrcTEE satisfy **SR1**-**SR5** by designing a novel secure boot scheme and designing new components in OP-TEE. Fig. 4 illustrates a simplified SrcTEE architecture (See Fig. 6 in Section V for detailed SrcTEE implementation), consisting of CSU, PL and OP-TEE running in APU extended with three components: CrloadIP, CexecIP and CremoAT. These components all need to make modifications or implementations in both user space and kernel space of OP-TEE. OP-TEE of SrcTEE has full access to all components of the device. REE is exposed to both users and cloud employees. We use a TA, Secure Management Application (SMA) to denote all the implementation in user space. The modifications to kernel space of OP-TEE include the definition and implementation of new Internal Core APIs (detailed in Section V). Such design aims to maintain a smaller Trusted Computing Base (TCB) size and avoid the enlarging of attack surface [33], [34]. A proxy server is deployed in REE for forwarding data between SMA (that is, CremoAT) and the remote user. Therefore, the user can verify the trustworthiness of SrcTEE, establish a secure communication path with it, deploy IPs, and invoke and check these IPs at runtime.

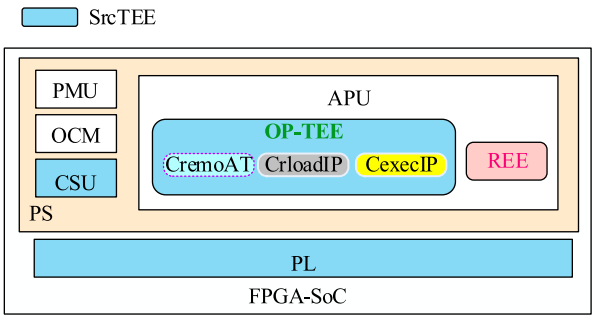

Fig. 4. A simplified SrcTEE architecture.

We now detail how SrcTEE meets **SR1-SR5**. For **SR1**, we realize a secure boot process (Section V.F), integrate a bitstream protection scheme (Section IV.C.1) and develop CremoAT to verify SrcTEE. CremoAT also provides physically unique and irreproducible fingerprint from PUF to authenticate FPGA-SoC for **SR2** and secure communication protocol for **SR5**. For **SR3**, CrloadIP is developed to ensure only SMA can deploy IPs in PL when integrity and authentication of bitstream are both ensured. For **SR4**, CexecIP provides the secure execution of IPs deployed in SrcTEE.

C. SrcTEE Approach

This section first describes a secure boot scheme. Then CrloadIP, CexecIP, and CremoAT are detailed. In the end, the phases of customizing SrcTEE at runtime are presented.

1) Secure Boot Scheme: Before presenting our secure boot scheme, we first introduce a bitstream protection scheme proposed in [42], which is used in our scheme. The authors in [42] proposed two secure bitstream protection schemes, one of which is realized by combining six security elements to counter checksum attacks and down grade attacks, dubbed as GCM-based scheme in this paper. These six elements include AES-GCM, root trust key stored in BBRAM, X-GHASH, a configuration counter, eFuse control register setting and un-reused initialization vectors (IVs).

We now describe our designed secure boot scheme. Through a custom First Stage Boot Loader (FSBL) detailed in Section V.F, we use a GCM-based scheme to authenticate the bitstream of a PUF IP before configuring it in PL, and then use PUF to authenticate PK_{TTP}. At last, we check the authenticity and integrity of boot codes (e.g., OP-TEE and ATF) via PK_{TTP} (**F1, F2**). Any malicious modification to boot codes will change the hash values and cause failure during authentication. In our proposed secure boot scheme, a device key is the sole root trust key used to decrypt AES keys for encrypted components in bootable image (**F3**). Moreover, the root trust key is updatable because it is stored in BBRAM instead of eFuse (**F5**), and no extra hardware module like TPM are used (**F4**). Differing from [43] and [44], the maintainability of our scheme lies in an unlimited number of times to refresh the root trust key. CSP must take the device to TTP for refreshing the key and TTP maintains the latest key of each device. Therefore, when the physical battery required for BBRAM runs out its power and then the loss of the root trust key occurs, the root key can be recovered with the help of TTP. Such loss seldom occurs because the life of a button battery usually used for BBRAM storage is 10 years [25].

2) CrloadIP: CrloadIP is comprised of (i) a reconfiguration Internal Core API in kernel space of SrcTEE (denoted as **P3** in Fig. 6 of Section V) and (ii) a functional module named as IP deployment in SMA. In CrloadIP, IPs in bitstream form can be configured in PL via a software interface (PCAP) or a hardware reconfiguration interface, Internal Configuration Access Port (ICAP). PCAP cannot be isolated from REE using isolation flow, but this can be achieved by generating custom PMU firmware to restrict both REE and TEE from accessing PCAP. Consequently, a secure interface for ICAP or PCAP should be integrated in OP-TEE, which only allows OP-TEE to configure PL and restricts insider attacks [17], [20]. Since the user has to take account the constraints on IPs caused by the design in static logic with an instance of ICAP integrated, we choose PCAP to configure PL. The access control mechanism in CrloadIP is dubbed as UUID-checking. The UUID-checking is achieved by comparing UUID of current TA which requests the kernel with the pre-stored UUID of SMA in kernel space of OP-TEE. Then we develop a syscall with UUID-checking and an Internal Core API (Section V.C) to request the syscall, which is a secure API for PCAP. CrloadIP can not only prevent insider attacks from deploying IPs via REE but restrict the rights of deploying IPs to one authorized TA, namely SMA. IP deployment in SMA is responsible for bitstream checking before deployment (Section V.E).

3) CexecIP: CexecIP consists of two parts: an IP Invocation Internal Core API in kernel space of SrcTEE (denoted as **P4** in Fig. 6 of Section V) and a functional module in SMA named as IP invocation (See Section V.D for implementation). **P4** is a secure interface to execute IPs deployed in SrcTEE. IP invocation is responsible for processing user requests and forwarding input to **P4** for invoking IP. We design an address-based input/output format for remote user requests, therefore any IP deployed at runtime can be used without modifying operating system of FPGA-SoC TEE. IP execution needs secure memory to store plaintext input/output data, which is achieved by a custom FSBL reserving a secure memory region when booting FPGA-SoC (Section V).

4) CremoAT: CremoAT can handle remote attestation at boot time and any runtime instant. There are three parts involved:

(i) A secure PUF IP in PL and an Internal Core API in kernel space of OP-TEE to obtain response from the PUF. We implement this secure PUF IP in initial hardware design and develop an Internal Core API in OP-TEE to obtain FPGA-SoC authentication credential and random seed from it. ARM TrustZone specific control ports are utilized to prevent arbitrary access from REE to the PUF (See Section V.A for implementation).

(ii) An Internal Core API in kernel space of OP-TEE to read measurements. We modify FSBL and OP-TEE to store measurements computed during secure boot and implement an Internal Core API in OP-TEE to read measurements computed at boot time (See Section V.B for implementation).

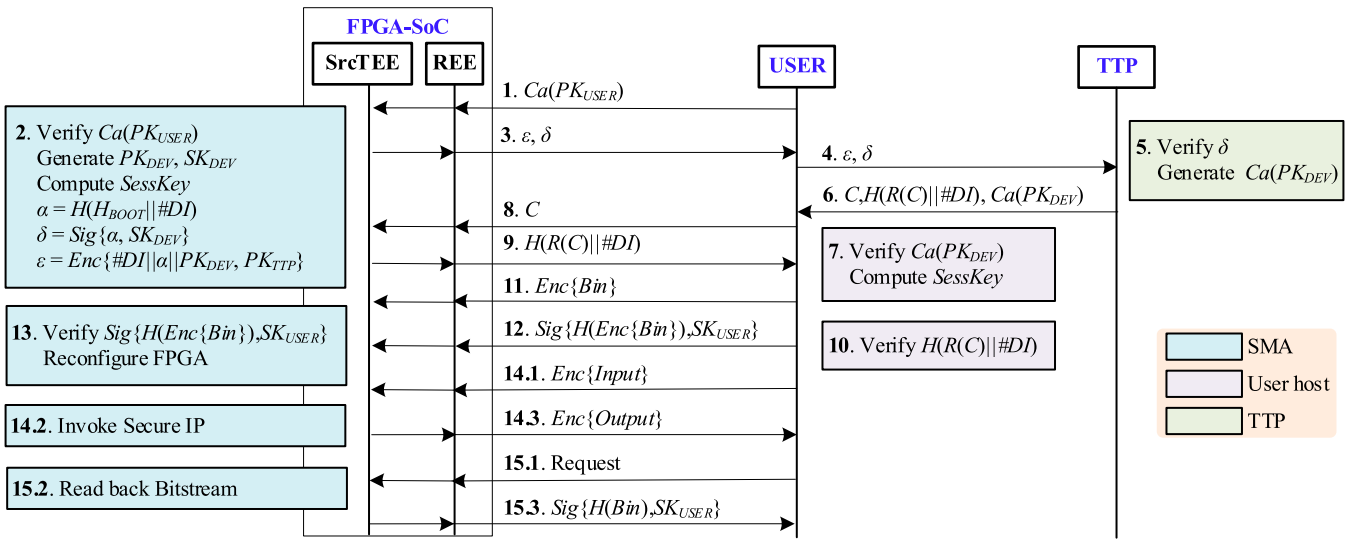

Fig. 5. Main steps in remote attestation at boot time, IP deployment, IP invocation, and remote attestation at runtime.

(iii) A function module in SMA to perform remote attestation at boot time and runtime. The remote attestation at boot time is a process when SMA and the user authenticate each other through TTP to establish a session key (*SessKey*). Remote attestation at runtime is a process when the user requests SMA to check the integrity of all the IPs deployed before.

5) Phases of Customizing SrcTEE: This section details the phases of Fig. 3 as follows. Fig. 5 illustrates the main steps of Phase ❸-❻.

Phase ❶: Device Enrollment

CSP rents out FPGA-SoC in order to gain profit. The first step is to enroll device information in the database of TTP, and request a device-specific bootable image to support the runtime TEE customizing for FPGA-SoC. TTP reads and stores *#DI* generated by device vendors. SK_{TA} and PK_{TA} are generated by TTP for authenticating TAs. Then SMA is signed with SK_{TA} and the default public key in OP-TEE is replaced with PK_{TA}. Afterwards, PK_{TTP} is stored in SMA for authenticating any certificate of public key signed by TTP and in FSBL for secure boot. A PUF IP is programmed on the device to collect CRPs. Then, a root trust key is generated and programmed into the device. It is used to encrypt a device-specific bootable image including the following components: REE operating system with SMA stored in its file system, OP-TEE, firmware and bitstream of an initial hardware design with the PUF IP integrated.

For each device, TTP creates an entry in a secure device database including the following items: (i) *#DI*, (ii) CSP's identity (*#CSP*), (iii) concrete board version of the device, (iv) the device key for secure boot, and (v) a list of CRPs of PUF. Then, the device and bootable image are delivered to the CSP. In our scheme, a unique device key must be generated for each device, which means a unique bootable image needs to be created for each device. When a device is enrolled, CSP publishes the details of all enrolled devices in its online store, and then users can choose devices according to their budget. When CSP receives the device and bootable image, the device is booted from the bootable image and SMA is started via the Linux terminal by cloud employees. Then TEE in the cloud FPGA-SoC supporting runtime customizing is ready to receive request from the user.

Phase ❷: User Enrollment

For every remote attestation process, the user updates the key pair (PK_{USER} and SK_{USER}) for attestation and registers PK_{USER}. The user first shares PK_{USER} with TTP and then an entry in the database of TTP is created to record user account (*#UID*) and PK_{USER}. When a user is enrolled, user can receive a certificate of PK_{USER}(i.e., $Ca(PK_{USER})$) and PK_{TTP}, and use $Ca(PK_{USER})$ to request SMA for remote attestation.

Phase ❸: Remote Attestation at Boot Time

Remote attestation at boot time starts when requested by the cloud user such that a communication path is established for later secure transmission. User utilizes TTP to verify measurements during secure boot and authenticates the FPGA-SoC. We design a protocol combining PUF-based authentication with public key infrastructure. The protocol flow is illustrated by **step 1-10** in Fig. 5. The details are given as follows.

The user makes a request to SMA for remote attestation with $Ca(PK_{USER})$, **step 1**. SMA first verifies the received $Ca(PK_{USER})$ with pre-stored PK_{TTP}. Only successful verification indicates the user has been registered by TTP and SMA can proceed with the remote attestation process started by the user. Then SMA invokes the PUF IP with pseudo-random input to obtain a random seed and generates an asymmetric key pair for attestation using the seed, a private (SK_{DEV}) and public key (PK_{DEV}). Then it reads the H_{BOOT} computed in the process of secure boot and obtains $R(H_{BOOT})$ by invoking the PUF IP with H_{BOOT} as challenge. SMA concatenates $R(H_{BOOT})$ with device identifier (*#DI*) and then computes a hash value to obtain $H(H_{BOOT}||\#DI)$, abbreviated as α. Next, SMA signs α to obtain a secure boot report δ. Using PK_{USER} and SK_{DEV}, SMA performs key exchange to compute *SessKey*. *#DI*, α and PK_{DEV} are encrypted with PK_{TTP} to obtain a private ε, **step 2**. The secure boot report δ and private ε are sent to the user, **step 3**.

Once receiving the message from SMA, the user forwards the message to TTP, **step 4**. Combining *#DI* and PK_{DEV} obtained by decrypting ε with pre-stored CRPs and H_{BOOT}, TTP verifies the secure boot report δ of the device, **step 5**. Only when the requested device is successfully verified, the TTP generates a certificate of PK_{DEV} (i.e., $Ca(PK_{DEV})$) and a FPGA-SoC authentication credential. The credential contains a session challenge C, and hash of corresponding response value of PUF concatenated with *#DI*. TTP sends $Ca(PK_{DEV})$ and the credential to the user, **step 6**. After receiving the message from TTP, the user verifies $Ca(PK_{DEV})$ with PK_{TTP}. Only after a successful validation, the user utilizes PK_{DEV} to compute a *SessKey*, stores the FPGA-SoC authentication credential, and then forwards C to the SMA, **step 7-8**. The SMA invokes the PUF IP to obtain the response of C, computes a $H(R(C)\|\#DI)$, and finally sends it to the user, **step 9**. The user can now authenticate the FPGA-SoC by comparing the value of $H(R(C)\|\#DI)$ received from the TTP and SMA. If the two values are the same, the user validates the state of the cloud device and can use *SessKey* to communicate with SMA, **step 10**.

Phase ❹: IP Deployment

IP deployment involves **step 11-13** presented in Fig. 5. The user develops a hardware design and generates bitstream, which can be used to deploy IPs in the cloud device. The bitstream is encrypted using *SessKey*, and sent to SMA in SrcTEE via the proxy server in REE [36], **step 11**. The user signs the hash of the encrypted bitstream and obtains a hash signature, $Sig\{H(Enc\{Bin\}), SK_{USER}\}$. Then the user requests the SMA for reconfiguring PL with the hash signature, **step 12**. SMA generates a hash of the encrypted bitstream and authenticates it with the hash signature received from the user, ensuring that untampered and authorized bitstream is received. Then SMA clears PL and reconfigures PL with decrypted bitstream via a secure reconfiguration (i.e., **P3**) interface, **step 13.**

Phase ❺: IP Invocation

The main steps of IP invocation are **step 14.1-14.3** presented in Fig. 5. The user makes a request to SMA for invoking a secure IP deployed in cloud device with encrypted input data. SMA decrypts the input data and passes the plaintext data to a secure IP invocation interface (i.e., **P4**). Finally, SMA reads output data via the interface, and sends encrypted output data to the user, **step 14**.

Phase ❻: Remote Attestation at Runtime

The main steps of remote attestation at runtime are **step 15.1-15.2** presented in Fig. 5. Remote attestation at runtime starts when the user requests SMA to check whether the hardware design including all the IPs deployed in PL is untampered, **step 15.1**. The SMA reads back bitstream of all the IPs deployed before, and signs the hash of the bitstream using SK_{DEV} to obtain $Sig\{H(Bin), SK_{USER}\}$, **step 15.2**. SMA sends $Sig\{H(Bin), SK_{USER}\}$ to the user, and the user can detect malicious tampering with the IPs deployed before, **step 15.3**.

V. Implementation

In this work, a Xilinx Zynq UltraScale+ XCZU15EG 2FFVB1156 MPSoC is chosen as the target device. We use Xilinx Vitis Design Suite 2020.1, Xilinx Vivado Design Suite 2020.1 and corresponding PetaLinux Tools for the final implementation. Fig. 6 illustrates SrcTEE implementation in FPGA-SoC, composed of CSU, extended OP-TEE and PL. Sky-blue blocks, grey blocks and yellow blocks represent CrloadIP, CexecIP and CremoAT, respectively. The implementation starts with creating a secure IP of Ring Oscillator (RO) PUF [37] (See Section V.A). Then a project of the initial hardware design integrated with several initial IPs is exported as an XSA file. The initial IPs include an instance of Zynq Ultrascale+ MPSoC, an AXI interconnect IP, and the secure RO-PUF. The XSA file is used in Vivado to create custom FSBL and PMU firmware (See Section V.B and Section V.C). Then we implement four new Internal Core APIs (**P1**: TEE_Sec_GetHwPufResponse, **P2**: TEE_Sec_GetBootHash, **P3**: TEE_Sec_ProgramUserHw, **P4**: TEE_Sec_UsrDefIP) and their corresponding syscalls called by these APIs, and then apply these APIs to develop SMA (See Section V.E). TCP stack provided by Linux is applied to realize a proxy server in REE.

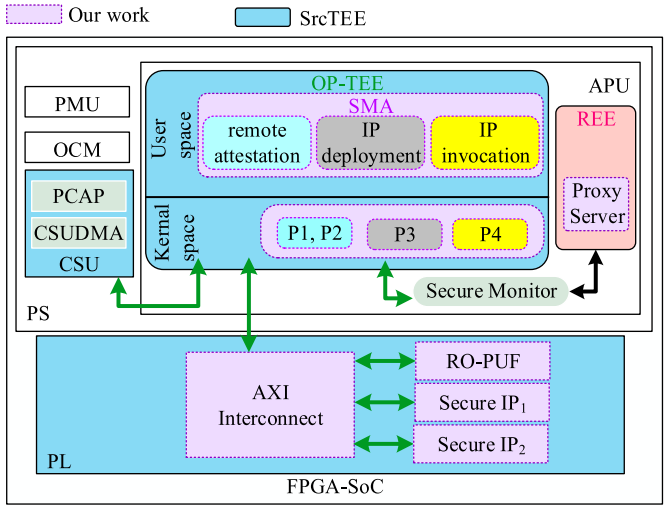

Fig. 6. Illustration of SrcTEE implementation on PS and PL in FPGA-SoC.

In comparison to the standard FSBL that is generated from Xilinx tools, we propose a custom FSBL. Although the method in [38] can measure and check other components in bootable image, it cannot provide secure storage for sensitive data and codes. The proposed custom FSBL supports access control of sensitive data stored in memory by configuring secure memory region (Section V.A), and supports the proposed secure boot scheme (Section V.F).

A. PUF-Based Random Seed and FPGA-SoC Authentication

In this work, RO-PUF proposed in [37] is adopted to provide a truly random seed for generating the key pair and unique response to authenticate FPGA-SoC for a remote attestation process. By doing this, the response produced must be hard to predict for attackers. Although FPGA-SoC contains a built-in PUF, the noise contained in the PUF response cannot be exploited for obtaining hardware entropy, because this information is not accessible to the developers. Therefore, we develop and package

the RO-PUF as an AXI peripheral with TrustZone specific signals including AWPROT and ARPROT, and an AXI Interconnect IP is used to connect PS and RO-PUF. Then, we set the AXI master of AXI Interconnect IP to declare RO-PUF as a secure IP, which is only accessible to OP-TEE. In addition, an Internal Core API (**P1**) and a corresponding syscall are implemented in the OP-TEE to read and write the input/output physical address spaces of AXI registers for the RO-PUF. The syscall only allows the request from SMA to invoke the secure RO-PUF, which is achieved by UUID-checking.

B. Storage and Reading of Measurement

In this work, access control of sensitive data stored in DDR memory and OCM is supported by custom FSBL. Decrypted bitstream computed before programming PL and plaintext input/output data of IPs obtained before/after writing/reading the MMIO address of IPs is stored in DDR memory. For secure storage of these data, two secure DDR memory regions are reserved by configuring XMPU (Xilinx Memory Protection Unit) in custom FSBL [40]. Moreover, a secure memory region is reserved for OP-TEE and TAs, and a secure OCM region is configured to store measurements computed by custom FSBL. The limitations of configuring static secure memory regions in FSBL executed during the early stages of the device boot process are as follows.

(i) Once the secure memory regions are defined during boot using XMPU, changes to the region layout or attributes require a restart of the device.
(ii) Any security issues or vulnerabilities identified later in the boot process or at runtime cannot be addressed.

We design a unified address-based input/output format for IPs and develop an address-based IP invocation Internal Core API (**P4**) in OP-TEE.

We design an Internal Core API (**P2**) in OP-TEE to read measurements computed by the custom FSBL and stored in secure memory region. These measurements cannot be directly used for remote attestation, since the FSBL is responsible for loading OP-TEE, which contains the SMA responsible for remote attestation. Therefore, the measurements of decrypted components are written in the last chunk of OCM by custom FSBL before OP-TEE is executed. We modify the boot file (boot.c) in OP-TEE to read measurements from the OCM and write these values in secure memory, which can only be accessed by TEE. Once the measurements are extracted from OCM, the measurements stored in OCM are cleared. When SMA starts to perform remote attestation requested by the user, it requests a TEE Internal Core API (**P2**) to execute corresponding syscall to read measurements stored in secure memory. Similar to **P1**, the access to corresponding syscall is controlled with UUID-checking.

C. Reconfiguration Internal Core API

We design a secure IP deployment Internal Core API (**P3**) in OP-TEE. Note that **P3** can be used to deploy IPs and read back bitstream of IPs. Xilinx considers PCAP as trusted under secure boot conditions. However, this interface allows the REE to load malicious IPs and to readback IPs in bitstream form [24]. To restrict access to the interface from REE, we create a custom PMU firmware by removing XilFPGA library from it, and thus PCAP is disabled for both TEE and REE. To support secure IP deployment through OP-TEE, we modify the OP-TEE to implement a syscall for PCAP enhanced by UUID-checking. By comparing UUID of current TA which requests deploying IP and the correct UUID of SMA pre-stored in the secure memory of OP-TEE kernel, UUID-checking is done. Therefore, only request from SMA is allowed to deploy IPs. The syscall transfers bitstream from secure memory into the PCAP using CSUDMA and invokes the PCAP to program bitstream in PL. An Internal Core API (**P3**) is designed for TAs to request the syscall in OP-TEE. In order to reconfigure PL with full bitstream, a bitstream setting (i.e., -bin_file) is added to write the bitstream as a binary bit without header (.bin) in Vivado.

D. IP Invocation Internal Core API

We design a unified address-based input/output format for IPs and develop an address-based IP invocation Internal Core API (**P4**) in OP-TEE. For input format, every input data and corresponding input physical address is specified as an input record. Then, a total number of input records, a physical address for indicating execution state, and a list of output physical addresses for output data should be specified as additional input data. For output format, every output data and corresponding output physical address is bound together as an output record. We reserve secure memory to store plaintext data before writing it into the input physical addresses of the IP and after reading it from the output physical addresses. A syscall enhanced by UUID-checking for invoking IPs via SMA is implemented. The syscall first writes the input data of an IP in the corresponding input physical addresses, then waits for the IP to finish computing by monitoring the execution state, and finally reads output data from the output physical addresses of the IP. **P4** is designed for SMA to request the syscall in OP-TEE.

E. Secure Management Application

SMA is implemented as a User-Mode TA [28] encompassing the following functional modules below.

1) Remote attestation module. FPGA-SoC authentication and verifiable TEE of remote attestation at boot time is achieved respectively using **P1** and **P2**. Then an ECDH lib [35] which can generate attestation key pair using the entropy source provided by the RO-PUF is integrated in the SMA. As thus the session key generated through key exchange is sourced from a truly random seed. Remote attestation at runtime is achieved using **P3** to read back bitstream of IPs deployed in PL and generate signature of the bitstream hash value.
2) IP deployment module. Bitstream checking is enforced before configuring PL with decrypted bitstream via **P3**. Bitstream checking consists of generation of a hash of the encrypted bitstream and verifying it against the provided signed one. Bitstream checking is achieved by

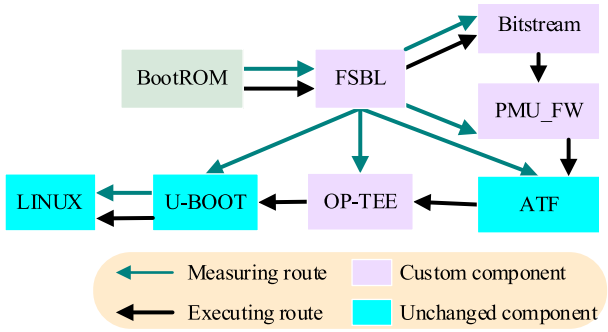

Fig. 7. Secure boot process.

utilizing the built-in SHA384 Internal Core APIs of OP-TEE and the integrated ECDH lib.

3) IP invocation module. **P4** is used to invocate the secure IPs deployed in PL.

F. Secure Boot

We now describe the process of packing an encrypted bootable image. Seven pairs of 256-bit AES encryption keys and seven 128-bit IVs are first generated. The key in one pair is programmed in BBRAM, used as the device key, and the remaining six pairs are used to encrypt six components in bootable image respectively. Using bootgen tool [39], seven keys and corresponding IVs are used to pack a bootable image with the following components encrypted using different key and IV:

1) Custom FSBL of 149 KB generated by Vitis. The custom FSBL contains PK_{TTP}, response of $H(PK_{TTP})$ and signatures of hash values for PMU_FW, ATF, OP-TEE, and U-BOOT.
2) PMU firmware (PMU_FW) of 133KB generated by Vitis.
3) Full bitstream (Bitstream) of 28029 KB generated from initial hardware design by Vivado.
4) ATF of 129 KB, REE bootloader and operating system (U-BOOT and LINUX) of 969KB and OP-TEE of 383KB generated by Petalinux tools.

We illustrate the boot process of our proposed secure boot scheme, as shown in Fig. 7. BootROM is fixed and stored on FPGA-SoC in an immutable way, which is a hardware root of trust. BootROM is first executed upon power on and then it decrypts FSBL before loading it into OCM. The FSBL then performs bitstream authentication using AES-GCM and X-GHASH [42] before configuring decrypted bitstream of PUF in PL. The FSBL then verifies the integrity of PK_{TTP}, which is essential to authenticate other components in bootable image. FSBL computes a hash value of PK_{TTP}, dubbed as $H(PK_{TTP})$, and invokes PUF with $H(PK_{TTP})$ as challenge to obtain a response, dubbed as $R(H(PK_{TTP}))$. Now FSBL can verify the integrity of PK_{TTP} by comparing$R(H(PK_{TTP}))$ with the response of $H(PK_{TTP})$ prestored in bootable image. Then FSBL computes a hash value of PMU_FW, dubbed as $H(PMU\_FW)$. With $H(PMU\_FW)$, PK_{TTP} and the signature of $H(PMU\_FW)$, PMU_FW can be authenticated using RSA algorithm [25]. Only If FSBL succeeds in authenticating PMU_FW, FSBL decrypts PMU_FW using its AES key, which is obtained by decrypting with device key. Measurement of decrypted PMU_FW is stored in secure OCM and PMU_FW is loaded for execution. Then FSBL continues to authenticate ATF, OP-TEE, and U-BOOT, which authenticates and loads a Linux Kernel and its root file system.

VI. Security and Performance Analysis

We provide security analysis and show the results of our experimental evaluation in Section VI.A and VI.B, respectively.

A. Security Analysis

Table III describes the attacks related to security requirements given in Section II.D. The capability of SrcTEE against these attacks is detailed in the following.

1) Cryptographic Attack: Cryptographic attackers are adversaries who attempt to break cryptographic security or obtain *SessKey*. Encrypted with *SessKey*, security-critical data and bitstream are protected in transmission. *Known-plaintext attack* (KPA) is prevented since only ciphertext is known to the attackers. Generated from RO-PUF, SK_{DEV} is never available to any parties including TTP other than SMA. Similarly, the user never shares SK_{USER} with other parties. Therefore, attacks based on known private keys are prevented and thus *SessKey* computed by ECDH is only shared between SrcTEE and remote USER.

2) Network Attack: Network attacks are conducted by malicious attackers attempting to impersonate the FPGA-SoC and then obtain private assets. The following three types are investigated:

Replay attack (RA) is prevented because the session challenge C sent by TTP and *SessKey* is never repeated. For every remote attestation process, SK_{DEV}, PK_{DEV}, PK_{USER} and SK_{USER} are updated to compute *SessKey*, which is ephemeral and is not derived from long-term private keys. Therefore, even if *SessKey* is compromised at a later time, attackers cannot retroactively decrypt the exchanged messages in the secure communication established by previous remote attestation process.

Impersonation attack (IMA) and *Man-in-the-middle attack* (MITM) can be thwarted due to use of RO-PUF for authenticating FPGA-SoC. Since the only possible way to access the secure PUF is restricted to SMA, attackers cannot access the PUF through REE. Responses generated from RO-PUF are difficult to predict when C is public, for the RO-PUF is resistant to machine learning and side channel based attacks [37]. Therefore, by matching $H(R(C)\|\#DI)$ sent by TTP and SMA, the user can detect FPGA-SoC impersonated by attackers. Furthermore, SK_{DEV} and PK_{DEV} cannot be replaced by attackers. Attackers cannot obtain and sign$H(H_{BOOT}\|\#DI)$ due to secure storage of $H(H_{BOOT}\|\#DI)$ based on secure boot and secure access to $H(H_{BOOT}\|\#DI)$ at runtime. As for communication channel between the user and TTP, the secure communication channel prevents attackers from impersonating TTP.

3) Insider Attack: Insider attackers are assumed to be malicious cloud employees who have the device-specific bootable image and physical access to the cloud device.

Invasive attack does not pose any threat to users but to CSP. *Semi-invasive attack* can divulge the device key without damaging the chip. However, attackers have to do multiple reverse

TABLE III
RELATIONSHIP BETWEEN SECURITY REQUIREMENTS AND SECURITY ANALYSIS

SR	Cryptographic Attack	Network Attack			Insider Attack		
	KPA	RA	IMA	MITM	RBA	FIA	UAFR
SR1					√	√	√
SR2		√	√	√			
SR3					√	√	
SR4							√
SR5	√	√					

TABLE IV
RESOURCE USAGE AND UTILIZATION RATIO BY THE INITIAL HARDWARE DESIGN AND TWO ACCELERATORS

Resources	Available	Initial Hardware design	IP1	IP2
LUT	341280	53989 (15.82%)	13691 (4.01%)	58626 (17.18%)
FF	682560	801 (0.12%)	3065 (0.45%)	49242 (7.21%)
BRAM	744	0 (0.0%)	112.5 (15.12)	282.5 (37.97%)
DSP	3528	0 (0.0%)	12 (0.34%)	0 (0.0%)

TABLE V
TIME OVERHEAD OF CRITICAL OPERATIONS

Operation	SrcTEE	Standalone Application
Keypair generation	50ms	-
Key exchange	48ms	-
TEE_Sec_GetHwPufResponse	53ms	-
TEE_Sec_GetBootHash	51ms	-
Checking of encrypted bitstream	7460KB/s	-
Program IP1(620KB)	69ms	7ms
Program IP2(2172KB)	102ms	36ms
Program full bitstream(28029KB)	204ms	52ms

engineer attacks on components in the bootable image, modify them and repackage a new bootable image. It is difficult to launch an attack through an unauthorized TA when OP-TEE kernel is not compromised since TA authentication and UUID-checking are enforced by OP-TEE kernel.

There exist several possible *Non-invasive attacks*. *Readback attack* (RBA) aims to read out the bitstream from PL via a reconfiguration interface even if the bitstream is encrypted before configuration. This attack is prevented via a custom PMU firmware and secure reconfiguration of Internal Core API, which is restricted to SMA. *Fault-injection attack* (FIA) refers to inject malicious IPs to damage the IPs designed by the user, or obtain private assets. It's difficult to program a malicious IP in PL because bitstream checking is enforced in SMA and UUID-checking is performed in OP-TEE kernel. The attack of *Unauthorized access from REE* (UAFR) to secure IPs deployed in PL is blocked because MMIO addresses of these IPs are protected by ARM TrustZone. Other *Non-invasive attacks* cannot be completely prevented but can be mitigated via proposed secure boot scheme. Relying on bitstream protection of PUF resistant to checksum attacks and down grade attacks, PMU_FW, ATF, OP-TEE, and U-BOOT are authenticated using RSA algorithm before decryption. Attackers are prevented from acquiring additional data per key by substituting their own data for the data contained in bootable image. For additional security, key rolling supported by ZU+ devices should be configured to limit the amount of data encrypted on any given key.

B. Performance Evaluation

In order to evaluate SrcTEE performance, we implement two accelerators, as the design of remote users to customizing FPGA-TEE at runtime. Our experiment is designed and tested on a Xilinx Zynq UltraScale+ XCZU15EG 2FFVB1156 MPSoC. There are mainly four types of available reconfigurable resources on the device:

- Lookup Tables (LUTs) and Flip-flops (FFs), used for general logic designs.
- Block RAMs (BRAMs), used as memory elements.
- Digital Signal Processing units (DSPs), used to translate signal processing algorithms with multiply-accumulation operations.

We evaluate SrcTEE performance in terms of two metrics: *Utilization Ratio* of the above reconfigurable resources and *Time Overhead* of three secure management operations, key exchange, keypair generation and checking of encrypted bitstream.

$$Utilization\ \ Ratio = Used/Available$$

Here, *Used* denotes the amount of reconfigurable resources used by a hardware design and*Available* denotes the amount of the whole reconfigurable resources on a Xilinx Zynq UltraScale+ XCZU15EG 2FFVB1156 MPSoC. Table IV and Table V show the results.

Table IV shows the resources used by the initial hardware design. Overall, the initial hardware design occupies 15.82% of LUTs and 0.12% of FFs. Since the hardware design developed by the user will be programmed in PL and overwrite the initial hardware design, the reconfigurable resources occupied by the initial hardware design do not affect the usage of users.

Resource utilization ratio of two convolutional neural network (CNN) accelerators [41], dubbed as IP1 and IP2 is summarized in Table IV. Different from initial hardware design, BRAMs are used to store parameters such as weights and bias that are exported from trained CNN models. In fact, users can develop personalized hardware designs according to their own needs, as long as the usage of various resources does not exceed the amount of available resources. Here we use two CNN accelerators to test the feasibility of runtime IP deployment in SrcTEE.

We evaluate *Time Overhead* of critical operations of SMA using a CA running in REE. Two partial bitstream files of 602KB and 2172KB are respectively generated from two CNN accelerators and used to test *Time Overhead*. Table V shows the results, from which we observe:

1) **Keypair generation**, **Key exchange**, **TEE_Sec_GetBootHash** and **TEE_Sec_GetHwPufResponse**. They

take 50ms, 48ms, 51ms and 53ms respectively. They are performed once during the protocol flow presented in Fig. 5, and the execution time of the operations is reasonable for SrcTEE.

2) **Checking of encrypted bitstream**. The checking of two encrypted bitstream files, including hashing step, bitstream decryption, and verifying signature, take 81ms and 290ms respectively with a throughput of 7460KB/s.
3) **TEE_Sec_ProgramUserHw**. We measure the time of calling TEE_Sec_ProgramUserHw to program IP1, IP2, and full bitstream. The time of programming IP1, IP2 and full bitstream on a standalone application without running any operating system is also measured and presented in Table V. Compared with the time of programming bitstream via a standalone application, SrcTEE consumes more time due to the introduction of two operating systems. A full bitstream file without header (.bin) is used in our experiment. It takes 204ms to program the file (.bin), 287ms faster than that with header (.bit), which is a good tradeoff for the additional security guarantees of SrcTEE. Note that partial bitstream files with header (.bit) are necessary for successful partial configuration.

In our experiment, the application to measure time overhead runs in REE while the critical operations run in TEE. Therefore, the measured time overhead is larger than the actual time overhead, which results from the time consumption caused by world switch between TEE and REE, taking about 45ms. Moreover, the results and progress of these operations are printed out during the execution of operations, which takes about 5ms to 10ms. In summary, SrcTEE is feasible in terms of overhead on the Xilinx Zynq UltraScale+ XCZU15EG 2FFVB1156 MPSoC.

VII. Conclusion and Future Work

This paper presents a scheme to provide remote users a secure runtime customizable TEE (SrcTEE) on cloud FPGA-SoC. The focus of this work is maintainability, practicality, security and verifiability. The existing secure boot solutions for FPGA-SoC either cannot achieve refreshable root trust key, or cannot achieve authenticity and integrity of boot codes. The existing works on runtime deployment of IPs in FPGA-SoC TEE imposed constraints of placement and routing, did not provide adequate access control on the deployment, and needed modification to the runtime system for each newly-installed IP core. SrcTEE addresses all of these issues. Maintainability is achieved by a secure boot process based on refreshable root trust key. We provide the random seed for key generation, device-specific values, and measurements of boot codes for remote attestation. For insider threats, SrcTEE can enforce access control on IP deployment and IP invocation from insider attackers. For IP protection, SrcTEE ensures the confidentiality of plaintext bitstream, and only authorized and untampered IPs are allowed to be deployed. Our security analysis indicates that SrcTEE can resist cryptographic attacks, network attacks, and insider attackers through software.

This paper has focused on the feasibility of a TrustZone-assisted runtime customizable TEE construction scheme for FPGA-SoC. Our next research is to explore support for multi-tenancy in FPGA-SoC by partitioning PL into regions and placing independent IPs developed by multi-tenants in different regions.

Acknowledgment

We would like to thank the editors and anonymous reviewers for their valuable comments and suggestions.

References

[1] N. Suda et al., "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, 2016, pp. 16–25.
[2] M. Armanuzzaman and Z. Zhao, "BYOTee: Towards building your own trusted execution environments using FPGA," 2022, *arXiv:2203.04214*.
[3] T. Hunt et al., "Telekine: Secure computing with cloud GPUs," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation (NSDI)*, 2020, pp. 817–833.
[4] F. Restuccia and A. Biondi, "Time-predictable acceleration of deep neural networks on FPGA SoC platforms," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2021, pp. 441–454.
[5] C. Bobda et al., "The future of FPGA acceleration in datacenters and the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, pp. 34:1–34:42, 2022.
[6] P. Sun, "Security and privacy protection in cloud computing: Discussions and challenges," *J. Netw. Comput. Appl*, vol. 160, 2020, Art. no. 102642.
[7] F. Brasser, P. Jauernig, F. Pustelnik, A.-R. Sadeghi, and E. Stapf, "Trusted container extensions for container-based confidential computing," 2022, *arXiv:*2205.05747.
[8] O. Demigha and R. Larguet, "Hardware-based solutions for trusted cloud computing," *Comput. Secur.*, vol. 103, 2021, Art. no. 102117.
[9] A. Dhar, S. Sridhara, S. Shinde, S. Capkun, and R. Andri, "Empowering data centers for next generation trusted computing," 2022, *arXiv:2211.00306*.
[10] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, "A novel FPGA architecture and protocol for the self-attestation of configurable hardware," *IACR Cryptology ePrint Archive, Paper 2019/405*, 2019.
[11] M. E. S. Elrabaa, M. Alasli, and M. Abu-Amara, "Secure computing enclaves using FPGAs," *IEEE Trans. Dependable Secur. Comput*, vol. 18, no. 2, pp. 593–604, Mar./Apr. 2021.
[12] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, "SACHa: Self-attestation of configurable hardware," in *Proc. Des., Automat. Test Eur. Conf. Exhib. (DATE)*, 2019, pp. 746–751.
[13] H. Oh, K. Nam, S. Jeon, Y. Cho, and Y. Paek, "MeetGo: A trusted execution environment for remote applications on FPGA," *IEEE Access*, vol. 9, pp. 51313–51324, 2021.
[14] S. Zeitouni, J. Vliegen, T. Frassetto, D. Koch, A.-R. Sadeghi, and N. MeNTens, "Trusted configuration in cloud FPGAs," in *Proc. IEEE 29th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2021, pp. 233–241.
[15] K. Xia, Y. Luo, X. Xu, and S. Wei, "SGX-FPGA: Trusted execution environment for CPU-FPGA heterogeneous architecture," *in Proc. 58th ACM/IEEE Des. Automat. Conf. (DAC)*, 2021, pp. 301–306.
[16] 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.
[17] N. Khan, S. Nitzsche, A. Garciandia López, and J. Becker, "Utilizing and extending trusted execution environment in heterogeneous SoCs for a pay-per-device IP licensing scheme," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2548–2563, 2021.
[18] Xilinx, "Zynq UltraScale+ MPSoC data sheet: Overview," Xilinx, San Jose, CA, USA, May 26, 2021. Accessed: 2021. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf
[19] Intel. Accessed: 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/ecosystems-fpga-soc-devices.html#tab-blade-1-1
[20] M. Gross, K. Hohentanner, S. Wiehler, and G. Sigl, "Enhancing the security of FPGA-SoCs via the usage of ARM TrustZone and a hybrid-TPM," *ACM Trans. Reconfigurable Technol. Syst*, vol. 15, no. 1, pp. 5:1–5:26, 2022.
[21] H.-Y. Kim et al., "SafeDB: Spark acceleration on FPGA clouds with enclaved data processing and bitstream protection," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD),* 2019, pp. 107–114.

[22] M. Zhao, M. Gao, and C. Kozyrakis, "ShEF: Shielded enclaves for cloud FPGAs," in *Proc. 27th ACM Int. Conf. Archit. Support Programming Lang. Operating Syst. (ASPLOS)*, 2022, pp. 1070–1085.
[23] W. Ren, J. Pan, and D. Chen, "AccGuard: Secure and trusted computation on remote FPGA accelerators," *in Proc. IEEE Int. Symp. Smart Electron. Syst. (iSES)*, 2021, pp. 378–383.
[24] Xilinx, "Zynq UltraScale+ MPSoC software developer guide," Xilinx, San Jose, CA, USA, Jul. 1, 2020. Accessed: 2022. [Online]. Available: https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_2/ug1137-zynq-ultrascale-mpsoc-swdev.pdf
[25] Xilinx, "Zynq UltraScale+ device technical reference manual," Xilinx, San Jose, CA, USA, Jan. 4, 2023. Accessed: 2023. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/user_guides/ug1085-zynq-ultrascale-trm.pdf
[26] M. K. Ahmed, S. K. Saha, and C. Bobda, "Trusted IP solution in multi-tenant cloud FPGA platform," 2022, *arXiv:2209.11274*.
[27] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. 44th Annu. Des. Autom. Conf. (DAC)*, 2007, pp. 9–14.
[28] OP-TEE Documentation. Accessed: 2023. [Online]. Available: https://optee.readthedocs.io/en/latest/
[29] Globalplatform. Accessed: 2023. [Online]. Available: https://globalplatform.org/
[30] S. A. G. Pereira, "A TrustZone-assisted secure silicon on a co-design framework," Doctoral dissertation, Universidade do Minho, Braga, Portugal, 2018.
[31] E. M. Benhani, C. Marchand, A. Aubert, and L. Bossuet, "On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC," in *Proc. 30th IEEE Int. Syst.-on-Chip Conf. (SOCC)*, 2017, pp. 108–113.
[32] S. Gören, O. Ozkurt, A. Yildiz, and H. F. Ugurdag, "FPGA bitstream protection with PUFs, obfuscation, and multi-boot," in *Proc. 6th Int. Workshop Reconfigurable Commun.-Centric Syst.-on-Chip (ReCoSoC)*, 2011, pp. 1–2.
[33] F. Schwarz, "TrustedGateway: TEE-assisted routing and firewall enforcement using ARM TrustZone," in *Proc. 25th Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2022, pp. 56–71.
[34] J. Sebastian, U. Agrawal, A. Tamimi, and A. Hahn, "DER-TEE: Secure distributed energy resource operations through trusted execution environments," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6476–6486, Aug. 2019.
[35] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *J. Cryptogr. Eng*, vol. 2, no. 2, pp. 77–89, 2012.
[36] A. Tellabi, A. Sabri, C. Ruland, and K. Waedt, "Security aspects of FPGA and virtualization case studies," *GI-Jahrestagung*, 2021, pp. 1771–1780.
[37] N. N. Anandakumar, M. S. Hashmi, and S. K. Sanadhya, "Design and analysis of FPGA-based PUFs with enhanced performance for hardware-oriented security," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 4, pp. 72:1–72:26, 2022.
[38] Xilinx, "Measured boot of Zynq UltraScale+ devices," Xilinx, San Jose, CA, USA, Apr. 18, 2019. Accessed: 2023. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp1342-measured-boot.pdf
[39] Xilinx, "Bootgen user guide," Xilinx, San Jose, CA, USA, Dec. 14, 2022. Accessed: 2022. [Online]. Available: https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_2/ug1283-bootgen-user-guide.pdf
[40] Xilinx, "Isolation methods in Zynq UltraScale+ MPSoCs," Xilinx, San Jose, CA, USA, Jul. 21, 2021. Accessed: 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/xapp1320-isolation-methods
[41] L. Alzubaidi et al., "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, 2021, Art. no. 53.
[42] M. Ender, G. Leander, A. Moradi, and C. Paar, "A cautionary note on protecting Xilinx' UltraScale(+) bitstream encryption and authentication engine," in *Proc. IEEE 30th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2022, pp. 1–9.
[43] D. Owen Jr. et al., "An autonomous, self-authenticating, and self-contained secure boot process for field-programmable gate arrays," *Cryptography*, vol. 2, no. 3, 2018, Art. no. 15.
[44] G. S. Nicholas, A. S. Siddiqui, S. R. Joseph, G. Williams, and F. Saqib, "A secure boot framework with multi-security features and logic-locking applications for reconfigurable logic," *J. Hardware Syst. Secur.*, vol. 5, no. 3, pp. 260–268, 2021.
[45] Z. Ling et al., "Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes," *J. Syst. Archit.*, vol. 119, 2021, Art. no. 102240.

Yanling Wang is currently working toward the master's degree with the School of Computer and Information Technology, Beijing Jiaotong University, China. Her research interests include trusted computing and trusted execution environment.

Xiaolin Chang (Senior Member, IEEE) is a Professor with the School of Computer and Information Technology, Beijing Jiaotong University. Her research interests include cloud-edge computing, network security, secure and dependable machine learning, and stochastic modeling.

Haoran Zhu (Graduate Student Member, IEEE) is currently working toward the Ph.D. degree with the School of Computer and Information Technology, Beijing Jiaotong University, China. His research interests include blockchain security and data transmission security.

Jianhua Wang is currently working toward the Ph.D. degree with the School of Computer and Information Technology of Beijing Jiaotong University, majoring in cyberspace security. His research interests include adversarial machine learning and federated learning.

Yanwei Gong is currently working toward the Ph.D. degree with the School of Computer and Information Technology, Beijing Jiaotong University, China. His research interests include identity authentication protocol related to MEC and fully homomorphic encryption acceleration.

Lin Li is a Professor with the Cyberspace Security, Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, China. Her research interests include cryptography.