POLITECNICO DI TORINO
Repository ISTITUZIONALE

An OpenSSL Engine for Secure and Transparent Offloading of Cryptographic Operations to OP-TEE in Embedded IoT Platforms

Original
An OpenSSL Engine for Secure and Transparent Offloading of Cryptographic Operations to OP-TEE in Embedded IoT Platforms / Sayarmoafi, T., Barchi, F., Bottaccioli, L., Acquaviva, A., Patti, E., Montuschi, P., Barbierato, L.. - In: IEEE INTERNET OF THINGS JOURNAL. - ISSN 2327-4662. - 13:16(2026), pp. 37591-37600. [10.1109/JIOT.2026.3701432]

Availability:
This version is available at: 11583/3011911 since: 2026-06-11T15:02:42Z

Publisher:
IEEE

Published
DOI:10.1109/JIOT.2026.3701432

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 November 2026

An OpenSSL Engine for Secure and Transparent Offloading of Cryptographic Operations to OP-TEE in Embedded IoT Platforms

Tina Sayarmoafi, Francesco Barchi, Lorenzo Bottaccioli, *Member, IEEE*, Andrea Acquaviva, Edoardo Patti, Paolo Montuschi, *Fellow, IEEE*, and Luca Barbierato, *Member, IEEE*

***Abstract*—Embedded IoT platforms are frequently deployed in hostile or physically exposed environments, where compromise of the operating system (OS) is a realistic threat. In conventional deployments, open secure sockets layer (OpenSSL) executes entirely in user space, leaving cryptographic keys and intermediate material potentially exposed in the presence of a compromised OS or privileged malware. This work presents a portable OpenSSL engine that enables secure and transparent offloading of cryptographic operations to open portable TEE (OP-TEE), a software stack leveraging a trusted execution environment (TEE) to confine cryptographic key material and security-critical computations within secure-world memory without modifying existing applications or altering the OpenSSL EVP interface. The proposed engine uses a GlobalPlatform (GP) client application programming interface (API) implementation as an underlying communication layer and supports both copy-based (CB) data transfer and a shared-memory (SM) mode, enabling performance tuning under the resource constraints typical of embedded IoT platforms. An experimental evaluation on an NXP i.MX7 industrial gateway shows that secure-world execution introduces bounded overhead dominated by rich execution environment (REE)–TEE transitions and memory-management operations. For bandwidth-intensive primitives such as SHA-256, SM mode improves throughput by approximately 10%–15% over CB transfers. For symmetric encryption algorithms such as AES-256-CBC, SM communication provides a substantial throughput improvement of approximately 40%, indicating that data-movement overhead plays a significant role on embedded platforms. In contrast, asymmetric primitives (e.g., RSA-1024) remain largely insensitive to transfer optimizations because they operate on small, fixed-size operands, making the overall execution time dominated by invocation overheads and internal big-number computations rather than data movement.**

***Index Terms*—Cryptographic offloading, edge computing, embedded systems security, Internet of Things (IoT), open portable TEE (OP-TEE), open secure sockets layer (OpenSSL), trusted execution environment (TEE).**

Received 21 January 2026; revised 23 March 2026 and 11 May 2026; accepted 4 June 2026. Date of publication 9 June 2026; date of current version 10 August 2026. This study was carried out within the project "Circular Tracing per l'Industria 5.0" (PNO F/310164/01-04/X56 – CUP B19J23000600005) funded by Ministero delle Imprese e del Made in Italy within the program "Accordi per l'Innovazione" (D.M. 31/12/2021, D.D. 18/03/2022, D.M. 25/05/2022). This article reflects only the authors' views and opinions, and the Ministry cannot be considered responsible for them. *(Corresponding author: Tina Sayarmoafi.)*

Tina Sayarmoafi, Lorenzo Bottaccioli, Edoardo Patti, Paolo Montuschi, and Luca Barbierato are with Politecnico di Torino, 10129 Turin, Italy (e-mail: tina.sayarmoafi@polito.it; lorenzo.bottaccioli@polito.it; edoardo.patti@polito.it; paolo.montuschi@polito.it; luca.barbierato@polito.it).

Francesco Barchi and Andrea Acquaviva are with the Universitádi Bologna, 40126 Bologna, Italy (e-mail: francesco.barchi@unibo.it; andrea.acquaviva@unibo.it).

Digital Object Identifier 10.1109/JIOT.2026.3701432

NOMENCLATURE

Abbreviation	Expansion
AES	Advanced encryption standard.
ARM	Advanced RISC machine.
API	Application programming interface.
CB	Copy-based.
CBC	Cipher block chaining.
GP	GlobalPlatform.
IoT	Internet of Things.
MMU	Memory management unit.
OP-TEE	Open portable TEE.
OpenSSL	Open secure sockets layer.
OS	Operating system.
RSA	Rivest–Shamir–Adleman.
REE	Rich execution environment.
SM	Shared-memory.
SoM	System on Module.
SMC	Secure monitor call.
SHA	Secure hash algorithm.
TLS	Transport layer security.
TEE	Trusted execution environment.
TA	Trusted application.
UUID	Universally unique identifier.

I. INTRODUCTION

THE exponential growth of the Internet of Things (IoT) has led to the widespread deployment of interconnected devices that continuously collect, process, and transmit sensitive data. In industrial and edge-computing scenarios, these devices often operate in physically accessible or otherwise untrusted environments, significantly enlarging the attack surface. Software stacks executing on such platforms are exposed to a variety of cyber threats, including memory disclosure through speculative-execution attacks [1], kernel compromise [2], and microarchitectural leakage affecting shared hardware resources [3]. These attack vectors demonstrate that cryptographic algorithm strength alone is insufficient; robust protection requires guarantees on the execution environment itself to ensure confidentiality and integrity of execution [4], [5].

OpenSSL remains the de facto cryptographic library for embedded, server, and cloud deployments, underpinning

© 2026 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

widely adopted secure communication protocols such as TLS [6], [7]. Through its EVP (Envelope) interface, which represents the high-level OpenSSL API that abstracts cryptographic primitives, OpenSSL provides a comprehensive set of services for encryption, hashing, and digital signatures, making it the primary entry point for application-level cryptography. However, in standard deployments, all cryptographic computations and key management occur entirely in the untrusted REE, which corresponds to the main OS domain. As a consequence, secret keys and intermediate values are exposed to software probing, privilege-escalation attacks, and memory-disclosure vulnerabilities such as Heartbleed [8]. Even when cryptographic implementations are correct, attacks such as Spectre [1] and Meltdown [2] demonstrate that sensitive data can be extracted through side-channel attacks exploiting speculative execution. System-level analyses of TrustZone-based platforms further confirm that insecure interactions between untrusted software and protected components remain a recurring cause of data leakage [3]. These observations motivate the need to strengthen the execution context in which cryptographic operations are performed.

Beyond device-level attacks, recent surveys on edge and IoT security emphasize that modern edge platforms increasingly operate under zero-trust assumptions, where OSs, middleware, and third-party software components cannot be fully trusted [9], [10]. In such environments, hardware-assisted isolation mechanisms are identified as a key enabler for protecting security-critical computation even in the presence of a compromised software stack.

A widely adopted mitigation strategy involves relocating security-critical operations into isolated, hardware-enforced execution domains. Hardware-rooted solutions such as trusted platform modules (TPMs), hardware security modules (HSMs), and Secure Elements provide strong protection guarantees, but often require dedicated hardware, specialized integration, and incur nonnegligible deployment overhead, particularly in resource-constrained embedded systems [11]. An alternative approach is offered by TEE, which provides a hardware-enforced secure domain that operates alongside the main OS, ensuring the confidentiality and integrity of code and data even in the presence of a compromised REE. In embedded platforms, the most widespread TEE technology is ARM TrustZone, which partitions the processor into a Normal World hosting the untrusted Rich Execution Environment (REE) and a Secure World dedicated to trusted services [4], [5]. This split-world execution model mitigates a broad class of attacks by confining cryptographic keys within the Secure World and executing sensitive cryptographic operations in the isolated secure environment.

Among TrustZone-based implementations, OP-TEE represents a widely adopted open-source TEE for embedded systems. OP-TEE provides a secure runtime environment in the Secure World and exposes the GP client API, a standardized communication interface that enables applications running in the REE to establish sessions and invoke trusted services. Within Open Portable TEE (OP-TEE), security-critical functionality is implemented as TA, which are isolated software components responsible for performing specific services, such as symmetric encryption, hashing, or digital signature generation, while ensuring that cryptographic keys remain inaccessible to untrusted software.

Several works have evaluated the use of OP-TEE to enhance the security of IoT and embedded systems. Nehal and Ahlawat [12] demonstrated that OP-TEE can effectively confine cryptographic keys and enforce hardware-level isolation. Broader studies highlight both the benefits and limitations of TrustZone-assisted execution, identifying performance overheads introduced by world switching, shared memory management, and Secure Monitor mediation [3], [4].

In the specific context of OpenSSL integration, Volante et al. demonstrated the feasibility of redirecting RSA signature operations to OP-TEE on the NXP i.MX7 platform [13]. Their work focused primarily on a single primitive and platform-specific integration. In contrast, this work generalizes the approach to multiple cryptographic primitives and introduces a backend-agnostic integration layer based primarily on the GP client API. These results underline the need for solutions that preserve strong isolation guarantees while carefully managing communication and execution overheads. Similar zero-trust assumptions are also adopted in recent edge-security architectures that combine software mechanisms with hardware roots of trust to confine sensitive operations [14].

Building on this background, two research challenges emerge. **RC1** concerns the absence of a reusable abstraction layer that cleanly decouples the OpenSSL EVP interface from specific TEE implementations. While the GP client API standardizes REE–TEE communication, existing integrations often bind OpenSSL directly to a particular TEE configuration or primitive, limiting extensibility and backend substitution. This work introduces a modular engine layer that preserves EVP transparency while isolating backend-specific logic behind a GP-compliant interface.

RC2 addresses the overhead introduced by frequent transitions and data transfers between the REE and the TEE, which can significantly impact performance if not carefully optimized [3], [4]. Despite multiple experimental integrations, existing solutions remain limited in scope, backend generality, or performance-awareness, leaving room for a more modular and EVP-transparent integration framework.

To address these challenges, this work introduces the **Agnostic Engine**, a portable OpenSSL engine that transparently redirects cryptographic workloads from the standard OpenSSL EVP interface to a trusted backend (e.g., OP-TEE). The proposed design leverages the GP client API to provide a standardized and portable communication path between unmodified applications and TAs running in the Secure World.

To address communication overhead across the TEE boundary, the proposed architecture supports both CB and SM data-transfer strategies and enables their direct comparison under identical conditions. This design allows us to isolate and quantify the impact of data-transfer mechanisms on REE–TEE transition overhead. In particular, Copy-Based (CB) relies on temporary buffer transfers managed by the OP-TEE driver, introducing additional copying overhead,

whereas Shared-Memory (SM) leverages GP buffer registration and reuse across multiple cryptographic invocations, thereby reducing data-movement costs. This dual-mode design enables performance-aware configuration while preserving the isolation guarantees provided by the underlying TrustZone hardware model.

The overall architecture combines the Agnostic Engine with a backend implementation (in this work, the CircularTracing backend running inside OP-TEE), enabling existing OpenSSL-based applications to benefit from TEE-assisted execution without code modifications. This design contributes to both **RC1**, by decoupling the OpenSSL integration layer from backend-specific implementations, and **RC2**, by supporting configurable REE–TEE data-transfer mechanisms for performance evaluation. Functional correctness of the offloaded primitives is verified across all evaluated cryptographic classes by cross-validating TEE-generated outputs against native OpenSSL results at the EVP interface level.

Although OpenSSL 3.x introduces the Provider model as the long-term pluggability mechanism, our design intentionally targets OpenSSL 1.1.1 to preserve backward compatibility with a RISC-V platform currently supported by the Agnostic Engine implementation [15]. This choice minimizes integration risk in long-lived embedded products where upgrading the crypto stack is constrained by certification and supply-chain requirements. We therefore focus on a robust ENGINE-based integration, while outlining a clear migration path to Providers as future work.

The security assumptions adopted in this work follow the standard ARM TrustZone threat model commonly used in TEE-based IoT platforms [4], [5]. In this model, the REE is considered untrusted and may host malicious or compromised components, including user-space applications, device drivers, and the OS kernel.

Communication between the REE and the TEE is mediated by the OS kernel through the GP client interface, which forward requests and associates them with a client identity. Since the kernel is part of the untrusted REE, a compromised kernel may arbitrarily manipulate forwarded requests. Consequently, a TA cannot reliably distinguish between legitimate and impersonated callers. The standard GP communication model does not inherently provide mechanisms for secure caller identification under these conditions.

Accordingly, this work defines a minimal and explicit trust boundary: the kernel-mediated request forwarding and identification mechanism is assumed to behave as specified by the platform. If this mechanism is compromised, impersonation attacks become possible and are considered out of scope.

The Secure World is protected by hardware-enforced isolation provided by ARM TrustZone. Secure-world memory is inaccessible from the REE, and all interactions are mediated by SMC, ensuring the confidentiality and integrity of trusted code, data, and cryptographic material.

All inputs originating from the REE are treated as untrusted. TAs perform parameter-type and bounds checking to ensure safe memory usage, without providing authentication or semantic validation of the request source.

Cryptographic keys and intermediate states are never exposed to the REE. They remain confined within Trusted Application (TA)s in secure-world memory managed by OP-TEE.

As a result, manipulation of shared buffers or TOCTOU attacks can affect only computation correctness (e.g., incorrect outputs or denial-of-service), but cannot compromise the confidentiality or integrity of cryptographic secrets, which remain protected within the Secure World.

This work does not address protection against physical or microarchitectural side-channel attacks, which are typically handled at the hardware and TEE platform level. Prior studies have demonstrated that such attacks can exploit shared microarchitectural structures even in TrustZone-based systems [16], [17]. The proposed design does not introduce additional shared resources or modify the underlying execution environment; therefore, its side-channel security properties are equivalent to those of the baseline OP-TEE deployment. Instead, the focus of this work is on reducing the software attack surface by confining cryptographic execution and key handling within the trusted domain while treating REE-originated inputs as untrusted.

The remainder of this manuscript is organized as follows. Section II reviews related work and positions this contribution within the existing literature. Section III describes the proposed methodology and system architecture. Section IV presents the experimental platform and configuration. Section V reports and discusses the experimental results, and Section VI concludes this article and outlines future research directions.

II. Literature Review

In this article, we use the term Secure Element to denote a trusted cryptographic backend that can be instantiated either: 1) as a software-isolated secure world (e.g., OP-TEE on TrustZone) or 2) as a discrete Root-of-Trust physically separated from the main application processor (e.g., OpenTitan-class designs). This broader definition reflects the design intent of the proposed architecture: the same OpenSSL interception layer can target heterogeneous trusted backends, enabling a unified integration strategy across ARM TEE deployments and emerging open RoT ecosystems, subject to backend-specific implementation constraints [18].

Recent work on open Root-of-Trust ecosystems has demonstrated that offloading OpenSSL primitives to a discrete trusted controller can provide substantial acceleration and isolation guarantees. For example, TitanSSL [19] proposes an OpenSSL-centric stack that offloads cryptographic tasks to OpenTitan, studying the tradeoff between communication overhead and the speed-ups enabled by RoT accelerators. However, in TitanSSL, the OpenSSL engine is tightly coupled to the OpenTitan backend and its specific security controller interface. In contrast, the architecture proposed in this work generalizes the engine design by introducing a backend-agnostic interception layer at the OpenSSL EVP level. This design decouples the OpenSSL integration from

backend-specific implementations, allowing the same engine architecture to interface with heterogeneous trusted backends without modification. Consequently, the proposed approach can support both TrustZone-based TEEs, such as OP-TEE and discrete Root-of-Trust platforms similar to OpenTitan, without requiring a dedicated OpenSSL engine for each backend.

TEE has become a central building block for securing cryptographic operations on ARM TrustZone-enabled platforms. Among available implementations, OP-TEE has emerged as the de facto open-source reference TEE due to its compliance with GP standards and its broad adoption across ARM-based embedded systems [5], [20], [21]. Despite these advantages, existing efforts to integrate OpenSLL with OP-TEE frequently rely on platform-specific adaptations or nonstandard communication paths, which limit portability, reuse, and long-term maintainability across heterogeneous hardware targets.

Early work by Nehal and Ahlawat [12] demonstrated the feasibility of using OP-TEE to protect cryptographic keys in IoT devices by isolating sensitive material from the Linux kernel. While their approach successfully achieved strong confidentiality guarantees, communication between the REE and the TEE was implemented using custom routines rather than the standard GP client API. As a consequence, the solution was tightly coupled to a specific platform, reducing portability and extensibility. This limitation highlights the persistent challenge of providing a unified and portable OpenSSL integration layer capable of leveraging standardized REE–TEE communication interfaces (e.g., the GP client API) across heterogeneous trusted backends, as formalized in **RC1**.

Recent contributions in the *IEEE Internet of Things Journal* further emphasize the role of TEEs as practical trust anchors for IoT systems operating in hostile or physically exposed environments. Mao et al. [22] proposed a blockchain- and TEE-assisted authentication framework that confines credentials and cryptographic operations within a secure execution environment, thereby protecting them from a potentially compromised OS. Their experimental results demonstrate that TEE-based isolation significantly improves system trustworthiness while introducing acceptable, bounded performance overhead. Although their work focuses on authentication protocols rather than general-purpose cryptographic libraries, it reinforces the broader applicability of TEE-based designs for protecting security-critical operations in embedded and edge IoT platforms.

Similarly, Jang and Kang [23] introduced 3rdParTEE, a framework that leverages TrustZone-based TEEs to secure third-party IoT services even in the presence of an untrusted kernel. Their study shows that isolating sensitive logic inside the TEE effectively mitigates software-level attacks while incurring moderate overhead dominated by world switches and runtime mediation. These findings corroborate the threat model and performance tradeoffs considered in this work, further motivating the use of TEEs as secure execution domains for cryptographic services.

Beyond application-specific integrations, several system-level studies have analyzed the architectural properties and performance implications of ARM TrustZone. Jauernig et al. [4] provided a comparative analysis of major TEE frameworks, identifying context switching, cache maintenance, and shared memory management as dominant contributors to secure-world latency. Cerdeira et al. [3] conducted a comprehensive security assessment of TrustZone-assisted TEEs, highlighting world-switch transitions and shared microarchitectural resources as critical attack surfaces. Prior work on TrustZone side-channel attacks further supports these observations [24], [25].

At the hardware level, Lesjak et al. [26] and Benhani et al. [27] explored TrustZone-assisted isolation and auxiliary co-processors for industrial IoT security, showing that hardware-enforced separation significantly improves trustworthiness, albeit often at the cost of flexibility or throughput. Parallel efforts in RISC-V ecosystems, such as OP-TEE [28] and OpenTitan Roots of Trust [19], reflect a growing interest in open and portable trusted execution platforms, yet they similarly lack standardized interfaces for OpenSSL integration.

Previous works have demonstrated the feasibility of redirecting selected OpenSSL primitives to OP-TEE. For example, Volante et al. [13] implemented an RSA-signature engine for OP-TEE on the NXP i.MX7 platform, showing that OpenSSL operations can be offloaded to the TEE to improve key isolation. However, such implementations typically target a specific algorithm class and are tightly coupled to particular backend configurations or platform adaptations. As a result, extending them to additional algorithms or alternative secure backends often requires redesigning or significantly modifying the engine implementation. In contrast, the proposed architecture introduces an *Agnostic Engine* within a broader *software stack*, designed to operate at the OpenSSL EVP abstraction layer while remaining independent of backend-specific implementations. This design allows cryptographic operations to be transparently redirected to trusted backends without modifying the OpenSSL-side integration. Consequently, support for additional cryptographic primitives can be incorporated without redesigning the engine itself; backend implementations can be extended independently while preserving the same OpenSSL interception layer. Similarly, alternative trusted backends can be integrated without requiring a dedicated engine for each platform. In the current implementation, the secure backend is provided by the *CircularTracing backend* running inside OP-TEE via the standard GP client API.

Furthermore, while SM support is already defined in the GP specification, the contribution of this work does not lie in the mechanism itself. Instead, the proposed software stack exposes both CB and SM transfers as interchangeable configurations within the same architecture, enabling controlled and systematic evaluation of REE–TEE communication tradeoffs across multiple cryptographic primitive classes. This work, therefore, extends previous OpenSSL–TEE integrations by introducing a portable interception layer that supports multiple cryptographic primitive classes while remaining independent of the underlying trusted backend.

Overall, the reviewed literature demonstrates substantial progress in leveraging TEEs for secure cryptographic processing, while revealing persistent fragmentation in interoperability and performance optimization. The proposed **software stack**, centered around the *Agnostic Engine*, addresses these

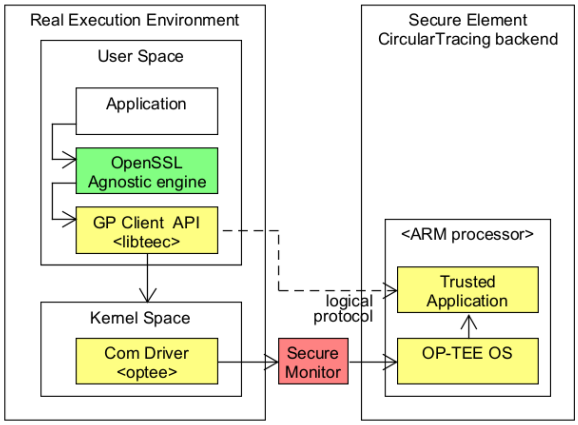

Fig. 1. Architectural overview of the proposed software stack, which redirects OpenSSL cryptographic operations to OP-TEE through the GP client Application Programming Interface (API).

limitations by enabling portable and performance-aware OpenSSL integration across heterogeneous trusted execution backends.

III. Methodology

The proposed methodology defines a comprehensive framework that enables user-space Open Secure Sockets Layer (OpenSSL) applications to offload cryptographic workloads to a secure execution domain implemented through OP-TEE. The solution is realized in the REE as an OpenSSL interception layer implemented by the **Agnostic Engine**, while the trusted domain hosts a set of OP-TEE TAs forming the CircularTracing backend. The proposed software stack addresses **RC1** by introducing an OpenSSL-side interception and abstraction layer that decouples EVP-level cryptographic operations from backend-specific implementations, while leveraging the GlobalPlatform (GP) client API as a standard transport mechanism across heterogeneous TEEs backends. In addition, it addresses **RC2** by optimizing data-transfer efficiency across the *Normal World* (i.e., the untrusted execution domain hosting the main OS) and the *Secure World* (i.e., the trusted execution domain enforced by Arm TrustZone) [29].

The overall architecture, shown in Fig. 1, is organized into three functional layers: 1) real execution environment; 2) secure monitor; and 3) secure element (CircularTracing backend implementation). This layered organization promotes portability across different hardware targets while preserving the isolation properties required by TEEs [4].

1) Real Execution Environment:

The REE represents the untrusted domain in which the host OS and user applications execute. It is composed of two sublayers: *User Space* and *Kernel Space*. Communication between these components follows the GP client API standard, ensuring interoperability with a compliant TEE backend.

2) User Space:

The User Space contains three primary components: the *Application*, the OpenSSL Engine (implemented as `Agnostic Engine`), and the *GP client API* (`libteec`). These modules cooperate to intercept cryptographic operations and transparently redirect them to OP-TEE.

Applications remain unmodified and continue to invoke cryptographic primitives through the OpenSSL `EVP` interface. For instance, when a TLS handshake or digital-signature request is initiated, the existing application logic is preserved while the underlying operation is automatically redirected to the trusted backend. This transparent interception ensures backward compatibility.

The OpenSSL Engine, `Agnostic Engine`, acts as a lightweight middleware layer that dynamically reroutes cryptographic primitives such as SHA-256, AES-256-CBC, and RSA-1024 to the secure execution domain. Each invocation is encapsulated in a metadata descriptor that specifies the algorithm type, key size, operation mode, and buffer references. These descriptors are translated into GP-compliant parameter structures and forwarded via the GP client API, providing a portable, backend-agnostic integration pathway across multiple TEEs. This abstraction effectively decouples OpenSSL from device-specific implementations, addressing the core requirement of **RC1**.

The GP client API (`libteec`) provides the communication bridge between the user-space engine and the kernel-space OP-TEE driver. It serializes metadata into GP-compliant command structures and invokes the secure backend using primitives such as `TEEC_InvokeCommand` and `TEEC_OpenSession`. These functions establish a secure session with the target TA and mediate all data transfers across the TEE boundary.

This software stack supports two explicitly configured data-transfer mechanisms between the REE and the TEE: CB and SM. In the CB configuration, data buffers are transferred using GP temporary memory references, which are marshaled by the OP-TEE client stack and copied across the TEE boundary on each invocation. This mode requires no persistent buffer state and represents the baseline communication mechanism. In the SM configuration, application buffers are explicitly registered through the GP SM mechanism using `TEEC_RegisterSharedMemory`. Registered buffers are subsequently referenced via partial memory references (`TEEC_MEMREF_PARTIAL_`$\star$), allowing the same SM regions to be reused across multiple cryptographic invocations. This reduces repeated data movement across the REE–TEE boundary while reducing data movement overhead, with access control and isolation guarantees enforced within the TEE. It is important to emphasize that in both CB and SM configurations, only nonsecure memory regions are used as communication buffers between the REE and the TEE. Secure-world memory remains inaccessible to the REE under the hardware-enforced TrustZone isolation model. Sensitive data is handled within secure-world memory and never exposed through SM buffers.

For streaming primitives such as Secure Hash Algorithm (SHA)-256, input buffers may be registered once and reused across successive update invocations when their address and capacity remain unchanged, amortizing SM registration costs over the lifetime of the hashing session. Short-lived buffers,

such as final padding tails, are instead registered on demand and released immediately after use. For block ciphers such as Advanced Encryption Standard (AES)-256-Cipher Block Chaining (CBC), input and output buffers may be registered and cached independently, enabling reuse across multiple encryption or decryption calls within the same cipher context.

In all cases, the lifetime of SM regions is explicitly bound to the cryptographic session or stream scope. Upon session teardown or algorithm cleanup, all registered SM regions are released. If SM registration is unavailable at initialization time or fails during operation, the engine operates exclusively in CB mode, preserving correctness and robustness. This design enables a direct and reproducible comparison between CB and SM configurations while accounting for SM management overheads and fulfilling the optimization objective defined in **RC2**.

In the SM configuration, SM buffers remain owned by the REE and are therefore considered attacker-controlled. Consistent with the threat model in Section I, the entire REE, including user-space applications and the OS kernel, is treated as untrusted.

Communication with the TEE is mediated by the kernel via the GP interface, which forward requests and associates them with a client identity. If the kernel is compromised, it may arbitrarily manipulate forwarded requests, and the TA cannot reliably distinguish legitimate from impersonated callers.

Cryptographic keys and intermediate states remain confined to secure-world memory within the TA and are never exposed to the REE. Nonsensitive parameters (e.g., IVs and metadata) and data buffers are exchanged through shared memory as required by the GP model.

For streaming inputs, the TA processes data directly from shared memory without retaining REE pointers beyond the scope of each invocation. Consequently, buffer manipulation can affect only computation correctness (e.g., incorrect outputs or denial-of-service), but cannot compromise the confidentiality or integrity of cryptographic material confined to the Secure World.

3) Kernel Space:

The Kernel Space hosts the `optee` communication driver (as shown in Fig. 1), which mediates interaction between the Linux kernel and the secure OP-TEE OS. This driver translates GP client requests into Secure Monitor Call (SMC) instructions and forward them to the Secure Monitor. It manages SM registration, session lifecycle, and command dispatching on behalf of user-space applications. The driver provides the underlying mechanisms required by both CB and SM, enabling the user-space engine to evaluate their performance impact while preserving isolation boundaries, thereby supporting **RC2**.

4) Secure Monitor:

The Secure Monitor regulates transitions between the *Normal World* and the *Secure World*, enforcing processor-state integrity and maintaining isolation [27]. Upon receiving an SMC request, the Secure Monitor suspends REE execution, preserves CPU context, manages cache and MMU coherence, and transfers control to the OP-TEE Operating System (OS). This ensures that sensitive data, once resident in secure-world memory, remains confined to the secure-world address space under the TrustZone isolation model.

5) Secure Element (CircularTracing Backend):

The Secure Element in this instantiation corresponds to the TEE provided by OP-TEE within the Secure World of the ARM processor. It comprises: 1) the OP-TEE OS and 2) the collection of TAs responsible for individual cryptographic operations. Cryptographic operations inside each TA are implemented exclusively using the native OP-TEE Internal Core API. No portion of the OpenSSL source code is ported or executed inside the Secure World. OpenSSL remains entirely in the REE and serves only as the application-facing cryptographic interface, while the TEE implementation relies solely on OP-TEE's native cryptographic framework.

6) ARM Processor:

The Advanced RISC Machine (ARM) Cortex-A7 processor used in this work implements Arm TrustZone extensions that provide hardware-enforced separation between secure and nonsecure execution states [30]. The OP-TEE OS provides the secure runtime environment responsible for service registration, session management, secure storage, and command handling [31]. It exposes GP-compliant entry points that accept driver requests, execute associated TAs, and return results to the REE. Hardware-backed memory-protection mechanisms ensure that keys and intermediate cryptographic data remain inaccessible to untrusted software. TA are implemented as isolated software modules registered within the OP-TEE OS. Each TA performs algorithm-specific operations—such as AES encryption, RSA signing, or SHA hashing—using buffers exchanged through the GP client API. Persistent TA-side state combined with SM buffer reuse reduces repeated initialization and buffer-registration overhead while preserving secure-world memory isolation guarantees, thereby fulfilling the performance and security objectives embodied in **RC2**.

IV. Scenario

The proposed software stack was evaluated on an industrial IoT gateway that integrates both the Linux-based REE and the TEE within a single embedded processor. This setup reflects typical deployment conditions in which secure and nonsecure workloads coexist on resource-constrained hardware.

1) Hardware Gateway:

Experiments were conducted on a gateway built around the NXP i.MX7 processor family, specifically the VAR-SOM-MX7 SoM developed by Variscite. The module integrates a dual-core ARM Cortex-A7 operating at up to 1 GHz. Both cores support Arm TrustZone extensions, enabling execution in either Normal World (REE) or Secure World (TEE). World transitions are mediated by the Secure Monitor, allowing Linux and OP-TEE to execute in their respective worlds on the

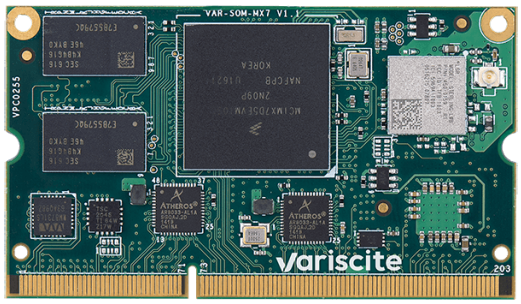

Fig. 2. VAR-SOM-MX7 System-on-Module used as the processing core of the MI-IoT-GW industrial gateway [30]. The module integrates a dual-core ARM Cortex-A7 and supports concurrent REE and TEE execution.

same CPU cores in a time-multiplexed manner rather than being statically assigned to separate cores. This hardware-level partitioning enforces memory isolation between secure and nonsecure execution states, confining cryptographic operations executed inside OP-TEE to secure-world memory [30].

The MI-IoT-GW gateway used in this study includes 2 GB of DDR3 RAM, on-board eMMC storage, and connectivity interfaces such as Ethernet, USB, serial, SDIO, Wi-Fi, and Bluetooth. The ARMv7-A architecture provides NEON SIMD, LPAE, and hardware integer division, enabling efficient cryptographic processing on embedded platforms.

Fig. 2 illustrates the VAR-SOM-MX7 module employed as the processing core of the gateway.

2) *Software Stack:*

The software configuration mirrors a realistic embedded deployment. A Debian-based Linux distribution running kernel version 5.15.71-imx7 operates in the Normal World and provides the execution environment for OpenSSL-based applications. In the Secure World, OP-TEE OS version 3.19.0 implements a GP-compliant TEE for isolated execution [29].

The proposed *Agnostic Engine* extends OpenSSL version 1.1.1 and redirects cryptographic operations to trusted backends through a wrapper layer compatible with the GP client API. This abstraction layer decouples the OpenSSL integration layer from backend-specific TEE client implementations while preserving a standardized invocation interface for TAs. In the prototype implementation evaluated in this work, the wrapper connects to the OP-TEE client library (`libteec`), enabling seamless invocation of TAs without requiring modifications to existing applications. Communication between the REE and the TEE is handled by the kernel-level `optee_driver`, which manages SMCs, SM registration, and session synchronization via the device node `/dev/tee0`.

3) *Experimental Context:*

The evaluation focuses on quantifying the performance and reliability of the proposed software stack integration under realistic embedded constraints. Three representative cryptographic primitives were analyzed: SHA-256 for hashing, AES-256-CBC for symmetric encryption, and Rivest–Shamir–Adleman (RSA)-1024 for asymmetric operations. Each primitive was implemented as a dedicated TA within OP-TEE, uniquely identified by its UUID.

Data transfers between the Normal and Secure Worlds were tested in two communication configurations: CB and SM. In the CB configuration, each invocation transfers data through temporary memory references managed by the OP-TEE driver; in SM mode, application buffers are registered through the GP SM mechanism and reused across multiple calls to reduce repeated data movement. While SM communication is expected to reduce latency and improve throughput by enabling buffer reuse, it requires more careful buffer-lifetime management—directly addressing the optimization objective captured in **RC2**.

Throughput was measured using wall-clock time measurements across payload sizes ranging from 16 kB to 128 MiB. Each experiment was repeated ten times, and the median value was reported to reduce sensitivity to transient fluctuations; the standard deviation remained below 5% across runs. Measurements were collected under three operational setups: native OpenSSL execution, OP-TEE in CB mode, and OP-TEE in SM mode. In addition to throughput measurements, functional correctness was validated for all evaluated primitives. For hashing, TEE-generated digests were byte-wise compared against native OpenSSL outputs. For AES-256-CBC, ciphertext produced inside the TEE was successfully decrypted in the REE (and vice versa), confirming interoperability. For RSA-1024, encryption–decryption and signature–verification pairs were cross-validated between REE and TEE implementations. All tested operations produced identical results across configurations.

All benchmarks were conducted in a single-threaded configuration. A single user-space process running in the REE invokes cryptographic operations sequentially through the Agnostic Engine. Each request triggers a synchronous REE–TEE invocation of the corresponding TA, and the backend software stack processes requests in a strictly sequential manner. Although the i.MX7 platform integrates two Cortex-A7 cores; the current implementation does not parallelise cryptographic operations across multiple cores. The TA execution model follows a sequential invocation pattern, meaning that each operation completes before the next begins. As a result, the experiments effectively utilize a single CPU core.

This controlled configuration allows us to isolate the per-invocation overhead introduced by REE–TEE transitions, secure-world entry, and data-transfer mechanisms. This setup also ensures deterministic measurements by eliminating variability introduced by thread scheduling and concurrent execution. The impact of multithreaded workloads, concurrent secure sessions, and multicore scaling is outside the scope of this work and is left for future investigation.

This experimental scenario provides a controlled testbed representative of typical industrial gateway deployments. It enables a systematic analysis of the tradeoffs between the isolation guarantees provided by OP-TEE and the performance implications of world switching, shared memory management, and TEE boundary communication optimization. Section V presents and discusses the resulting experimental observations. Results are reported for a single hardware platform; while the architectural design is backend-agnostic, quantitative performance characteristics may vary

TABLE I
SHA-256 THROUGHPUT ON THE PROPOSED SOFTWARE STACK: NATIVE OPENSSL VERSUS OP-TEE CB COMMUNICATION AND SM COMMUNICATION. ALL THROUGHPUTS ARE EXPRESSED IN MB/S

Payload [kB]	OpenSSL [MB/s]	CB [MB/s]	SM [MB/s]
16.4	0.99	0.23	0.23
82.9	4.05	0.99	1.03
417.8	10.70	3.49	3.76
913.4	11.70	5.15	5.73
2970.0	12.00	6.68	7.49
134217.7	12.25	6.82	7.34

across SoCs with different secure-monitor implementations or hardware accelerators.

V. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed software stack, throughput benchmarks were conducted on representative cryptographic primitives. Native OpenSSL execution in user space is compared with two OP-TEE-based configurations: CB communication and SM communication. These measurements quantify the overhead introduced by secure-world execution and assess the extent to which SM buffer registration mechanisms mitigate this overhead. All tests were performed on the industrial IoT gateway described in Section IV, using the GP client API as the communication path between the REE and the TEE [29]. Throughput is reported in megabytes per second (MB/s) for payload sizes ranging from 16 kB to 128 MiB. All configurations were executed on the same hardware platform under identical system conditions, ensuring a fair comparison between native OpenSSL and TEE-based execution. All reported measurements reflect end-to-end throughput and include the complete software stack, encompassing the Agnostic Engine in the REE, the GP client API, the OP-TEE kernel driver, Secure Monitor transitions, and TA execution in the Secure World. Each datapoint represents the median of ten repeated measurements. Functional correctness was validated prior to benchmarking as described in Section IV.

A. SHA-256

Table I reports the measured throughput of the SHA-256 hashing primitive for increasing payload sizes, comparing native OpenSSL execution with OP-TEE CB and SM configurations. For small payloads (16–83 kB), both secure-world modes achieve throughputs below 1 MB/s, reflecting the dominance of fixed overheads related to world switching, Secure Monitor transitions, and cache maintenance. In this regime, performance is approximately one-fifth of native execution.

As payload size increases, throughput improves steadily for all configurations. At intermediate payload sizes (417–913 kB), SM communication consistently outperforms CB communication, reaching 3.76–5.73 MB/s compared to 3.49–5.15 MB/s. This corresponds to a relative throughput improvement of approximately 10%–15%, confirming the benefits of buffer reuse for bandwidth-intensive hashing workloads. For large payloads (above 3 MB), throughput stabilizes: native OpenSSL sustains around 12 MB/s, while CB and SM converge to approximately 6.7–7.5 MB/s.

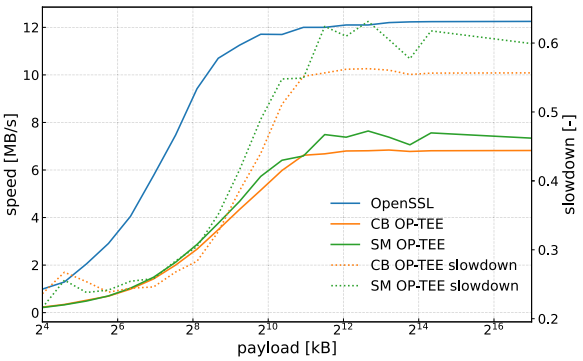

Fig. 3. Throughput of SHA-256 hashing for native OpenSSL, CB communication, and SM communication. Solid curves indicate throughput [MB/s]; dotted lines denote slowdown relative to native execution.

TABLE II
AES-256-CBC THROUGHPUT ON THE PROPOSED SOFTWARE STACK: NATIVE OPENSSL VERSUS OP-TEE CB COMMUNICATION AND SM COMMUNICATION. ALL THROUGHPUTS ARE EXPRESSED IN MB/S

Payload [kB]	OpenSSL [MB/s]	CB [MB/s]	SM [MB/s]
16.4	1.15	0.14	0.14
82.9	3.59	0.61	0.65
417.8	8.22	2.02	2.3
913.4	8.39	2.92	3.66
2970.0	8.53	3.92	5.50
134217.7	8.65	4.01	5.74

Fig. 3 visualizes these trends across the full payload range. Solid curves represent absolute throughput (left axis), while dotted curves show the slowdown relative to native execution (right axis). The figure highlights that SM operation achieves approximately 0.60–0.65 of native throughput in steady state, compared to approximately 0.55–0.60 for the CB configuration. These results confirm that SM buffer registration mitigates a meaningful fraction of REE–TEE data-transfer overhead for data-intensive hashing workloads.

B. AES-256-CBC

Table II summarizes the throughput of the AES-256-CBC cipher under the same execution modes. For small payloads (below 83 kB), both OP-TEE configurations remain below 1 MB/s due to the combined cost of world switching and cipher-context initialization. As payload size increases, throughput rises and stabilizes beyond approximately 2 MB.

In the steady-state region, native OpenSSL achieves approximately 8.5 MB/s, while the CB configuration stabilizes around 4.0 MB/s. In contrast, SM communication reaches sustained throughputs of approximately 5.6–5.8 MB/s, corresponding to a relative improvement of about 40%–45% compared to CB transfers.

Fig. 4 confirms this behavior graphically. Solid curves depict absolute throughput, while dotted curves indicate slowdown relative to native execution. For AES-256-CBC, SM communication provides a clear throughput improvement over the CB configuration. Although symmetric encryption is primarily computation-oriented, repeated block-based processing and frequent buffer exchanges cause data-movement overhead to

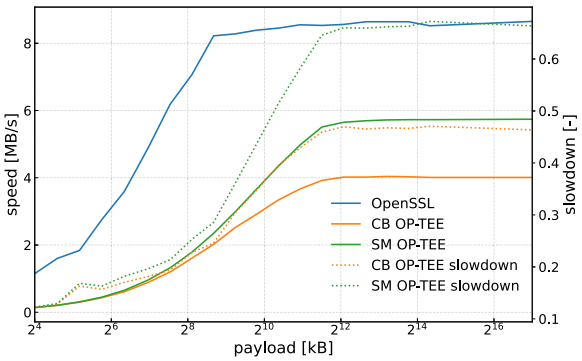

Fig. 4. Throughput of AES-256-CBC encryption for native OpenSSL, CB communication and SM communication. Solid curves indicate throughput [MB/s]; dotted lines denote slowdown relative to native execution.

contribute nonnegligibly to overall execution time on embedded platforms. By reusing registered buffers across successive encryption and decryption calls, SM communication reduces repeated memory copying and cache-maintenance operations, resulting in the observed performance gain.

C. RSA-1024

The results for RSA-1024 exhibit the expected compute-bound behavior [13]. Native OpenSSL achieves the highest throughput, while both CB and SM configurations executed in OP-TEE introduce additional latency dominated by the fixed overhead of the secure invocation path required to execute the operation inside the TEE.

Unlike hashing or symmetric encryption, where large payloads make data movement a dominant factor, RSA operations involve very small fixed-size operands (e.g., 128 bytes for RSA-1024). As a consequence, the per-invocation overhead introduced by the secure execution path cannot be amortized across larger data transfers. The observed latency, therefore, reflects the fixed per-invocation overhead of the complete software stack, including engine interception, GP invocation, driver interaction, world switching, and TA execution.

As a result, throughput remains effectively constant across all tested payloads, and SM communication provides no measurable benefit. A graphical representation is therefore omitted. In practice, secure-world execution incurs a slowdown of approximately 1.6× relative to native OpenSSL, reflecting the tradeoff between performance and the stronger key-isolation guarantees enforced by the TEE.

D. Discussion

The experimental results highlight the expected overheads of secure-world execution in OP-TEE. Slowdowns arise primarily from Secure Monitor transitions, cache synchronization, and shared memory management. For data-intensive primitives such as SHA-256, SM communication measurably mitigates TEE boundary transfer overhead. For symmetric encryption workloads such as AES-256-CBC, a substantial throughput improvement of approximately 40% is observed, demonstrating that data-movement overhead remains relevant even for computation-oriented primitives on embedded platforms. In contrast, asymmetric algorithms such as RSA-1024 remain largely insensitive to transfer optimizations, as their execution time is dominated by the fixed per-invocation overhead of the secure execution path rather than by data-transfer costs.

Overall, the **software stack** enables transparent integration for unmodified OpenSSL applications while executing cryptographic operations inside the Secure World. Although secure-world execution introduces a measurable and bounded performance penalty, the tradeoff reflects the cost of enforced isolation and controlled REE–TEE interaction. On industrial IoT gateways, where key isolation and trusted execution are primary design requirements, this overhead may be acceptable within typical deployment constraints.

VI. Conclusion

This work presented a portable OpenSSL engine that enables secure and transparent offloading of cryptographic operations to OP-TEE, enabling transparent EVP-level redirection of cryptographic operations to the CircularTracing backend without requiring modifications to existing OpenSSL applications. The proposed architecture addresses the research challenges **RC1** and **RC2** by providing a standardized and portable communication pathway based on the GP client API, while supporting both CB and SM data-transfer mechanisms across the TEE boundary.

Experimental evaluation on an industrial IoT gateway demonstrated that the proposed software stack preserves full application compatibility while ensuring that cryptographic key material and secure-world cryptographic state are confined to secure-world memory during protected execution. The results indicate that the SM configuration reduces data-transfer overheads across the TEE boundary for data-intensive primitives such as SHA-256. For symmetric encryption workloads such as AES-256-CBC,SM buffer reuse further provides a substantial throughput improvement by mitigating repeated data movement and cache-maintenance overhead. In contrast, asymmetric algorithms such as RSA-1024 are primarily affected by the fixed per-invocation overhead of the secure execution path, rather than data-transfer inefficiencies, due to their small operand size and nonstreaming nature.

These performance characteristics align with the intended deployment scenarios considered in this work. In industrial and edge IoT environments, cryptographic operations typically protect authentication credentials, device identity, firmware integrity, and secure communication channels, where strong isolation and predictable execution are prioritized over peak throughput. In this context, the bounded performance overhead introduced by secure-world execution represents a practical tradeoff for improved key isolation under the TrustZone execution model and reduced exposure window of key material in the REE.

Overall, the proposed architecture, composed of the Agnostic Engine and the CircularTracing backend, provides a balanced compromise between performance and security, delivering a modular, portable, and standards-compliant solution suitable for industrial systems where determinism, trustworthiness, and interoperability take precedence over peak cryptographic throughput.

While this work evaluates a representative set of primitives (SHA-256, AES-256-CBC, and RSA-1024) spanning streaming, symmetric, and asymmetric computation classes, the measured overhead trends are primarily influenced by structural properties of the REE–TEE interaction model, in addition to algorithm-specific characteristics. As such, the observed transition and data-movement costs are expected to extend to additional cryptographic primitives implemented through the same invocation pathway, subject to their computational profile.

Future work will focus on further optimizing shared memory management and secure-world interactions, and on extending the experimental coverage to additional primitives already supported by the OP-TEE crypto API, including AES-GCM (AEAD), ECDH/ECDSA, and RSA-2048, to further validate the generality of the observed overhead patterns under both CB and SM transfer modes. Future work will also investigate complementary mechanisms for strengthening the trustworthiness of the calling environment, such as remote attestation and integrity measurement techniques, in order to enable stronger guarantees on request origin identification even in the presence of a compromised REE kernel. We also plan to extend the framework to support asymmetric key-exchange protocols and dynamic key provisioning, and to explore tighter integration with hardware accelerators in order to improve scalability and reduce latency in next-generation trusted gateway architectures. In addition, while this article targets OpenSSL 1.1.1 to satisfy strict legacy constraints (including RISC-V platform compatibility), a Provider-based implementation will be investigated to align with the OpenSSL 3.x pluggability model and forthcoming ENGINE deprecation, reusing the same backend-agnostic descriptors and data-transfer abstractions introduced in this work.

REFERENCES

[1] P. Kocher et al., "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1–19.

[2] M. Lipp et al., "Meltdown," in *Proc. 27th USENIX Secur. Symp.*, vol. 63, 2020, pp. 46–56.

[3] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "SoK: Understanding the prevailing security vulnerabilities in TrustZone-assisted TEE systems," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1416–1432.

[4] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted execution environments: Properties, applications, and challenges," *IEEE Secur. Privacy*, vol. 18, no. 2, pp. 56–60, Mar. 2020.

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

[6] O. S. Foundation. *(1998–2024) OpenSSL Project*. Accessed: Oct. 6, 2025. [Online]. Available: https://github.com/openssl/openssl

[7] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, document RFC 8446, 2018. Accessed: Oct. 6, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446

[8] Codenomicon and OpenSSL Project. (2014). *The Heartbleed Bug*. Accessed: Oct. 6, 2025. [Online]. Available: https://heartbleed.com/

[9] B. Mao, J. Liu, Y. Wu, and N. Kato, "Security and privacy on 6G network edge: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1095–1127, 2nd Quart., 2023.

[10] M. M. Fouda, Z. M. Fadlullah, M. I. Ibrahem, and N. Kato, "Privacy-preserving data-driven learning models for emerging communication networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 27, no. 4, pp. 2505–2542, Aug. 2025.

[11] "Guidelines on hardware-rooted security in mobile devices," Initial Public Draft, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, NIST Special Publication 800-164, Oct. 2012. Accessed: Oct. 6, 2025. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/164/ipd

[12] A. Nehal and P. Ahlawat, "Securing IoT applications with OP-TEE from hardware level OS," in *Proc. 3rd Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA)*, Jun. 2019, pp. 1441–1444.

[13] F. Volante, F. Barchi, E. Patti, L. Bottaccioli, and L. Barbierato, "OP-TEE powered OpenSSL engine enhancing digital signature security for ARM architectures," in *Proc. 20th Int. Conf. Synth., Modeling, Anal. Simulation Methods Appl. Circuit Design (SMACD)*, Jul. 2024, pp. 1–4.

[14] B. Mao et al., "A blockchain-enabled cold start aggregation scheme for federated reinforcement learning-based task offloading in zero trust LEO satellite networks," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 6, pp. 2172–2182, Jun. 2025.

[15] E. Parisi et al., "Assessing the performance of OpenTitan as cryptographic accelerator in secure open-hardware system-on-chips," in *Proc. 21st ACM Int. Conf. Comput. Frontiers*, May 2024, pp. 172–179.

[16] Z. Kou, W. He, S. Sinha, and W. Zhang, "Load-step: A precise TrustZone execution control framework for exploring new side-channel attacks like flush+evict," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 979–984.

[17] T. Xu, A. A. Ding, and Y. Fei, "TrustZoneTunnel: A cross-world pattern history table-based microarchitectural side-channel attack," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2024, pp. 1–11.

[18] M. Ciani et al., "Unleashing OpenTitan's potential: A silicon-ready embedded secure element for root of trust and cryptographic offloading," *ACM Trans. Embedded Comput. Syst.*, vol. 24, no. 5, pp. 1–29, Sep. 2025.

[19] A. Musa et al., "TitanSSL: Towards accelerating openssl in a full RISC-V architecture using OpenTitan root-of-trust," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.* Cham, Switzerland: Springer, 2024, pp. 169–183.

[20] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone explained: Architectural features and use cases," in *Proc. IEEE 2nd Int. Conf. Collaboration Internet Comput. (CIC)*, Nov. 2016, pp. 445–451.

[21] H. Yang and M. Lee, "Demystifying ARM TrustZone tee client API using O+P-TEE," in *Proc. SMA*, 2020, pp. 100–106.

[22] W. Mao, P. Jiang, and L. Zhu, "BTAA: Blockchain and TEE-assisted authentication for IoT systems," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12603–12615, Jul. 2023.

[23] J. Jang and B. B. Kang, "3rdParTEE: Securing third-party IoT services using the trusted execution environment," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 15814–15826, Sep. 2022.

[24] B. Ngabonziza, H. Cho, A. Bailey, D. Martin, and S. Martin, "TruSpy: Cache side-channel attacks on trustzone," in *Proc. IEEE Int. Symp. High Assurance Syst. Eng. (HASE)*, 2016, pp. 98–103.

[25] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache attacks on mobile devices," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, Austin, TX, USA, Aug. 2016, pp. 549–564.

[26] C. Lesjak, D. Hein, and J. Winter, "Hardware-security technologies for industrial IoT: TrustZone and security controller," in *Proc. IECON 41st Annu. Conf. IEEE Ind. Electron. Soc.*, Nov. 2015, pp. 2589–2595.

[27] 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.

[28] M. Boubakri, F. Chiatante, and B. Zouari, "Open portable trusted execution environment framework for RISC-V," in *Proc. IEEE 19th Int. Conf. Embedded Ubiquitous Comput. (EUC)*, Oct. 2021, pp. 1–8.

[29] G. P. Group. (2023). *Global Platform Client API Documentation*. [Online]. Available: optee.readthedocs.io/en/stable/architecture/globalplatformapi.html

[30] V. Ltd. (2016). *Var-Som-Mx7 System-on-Module Datasheet*. Accessed: Oct. 6, 2025. [Online]. Available: https://www.variscite.com/wp-content/uploads/2017/12/VAR-SOM-MX7VAR-SOM-MX7-5Gdatasheet.pdf

[31] G. P. Group. (2023). *Global Platform Official Website*. [Online]. Available: https://globalplatform.org/

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement