OP-TEE powered OpenSSL Engine enhancing Digital Signature security for ARM Architectures

OP-TEE powered OpenSSL Engine enhancing Digital Signature security for ARM Architectures

Franco Volante*, Francesco Barchi†, Edoardo Patti*, Lorenzo Bottaccioli*, and Luca Barbierato*
* Politecnico di Torino, Torino, Italy. Emails: {name.surname}@polito.it
† University of Bologna, Bologna, Italy. Emails: {name.surname}@unibo.it

***Abstract*—This paper presents a novel approach to enhancing the security of OpenSSL software for ARM architectures by leveraging an open source Trusted Execution Environment (TEE), so-called OP-TEE. The approach involves establishing communication between an OpenSSL Engine and a secure execution environment within OP-TEE, protecting cryptographic operations and sensitive data (e.g. private keys) against potential hardware and software vulnerabilities. The architecture is tested on a Digital Signature scenario using an ARM SoM based on the NXP/Freescale i.MX7 processor. The study unveils that the proposed architecture incurs a latency overhead due to the connection to OP-TEE. Conversely, the architecture exhibits an increase in execution time compared to standard OpenSSL software for data block sizes of $4\,\mathrm{MB}$, with a manageable overhead of $32\,\mathrm{ms}$. This overhead is deemed acceptable, given the security enhancements introduced by the architecture. The research underscores the significance of leveraging OP-TEE in addressing emergent cybersecurity challenges, thus bolstering the resilience of OpenSSL software in ensuring the security of connected devices.**

***Index Terms*—ARM, Trusted Execution Environment, OP-TEE, OpenSSL Engine**

I. Introduction

ARM architectures have become widely used across a spectrum of computing devices, from smartphones and tablets to embedded systems, Internet of Things (IoT) devices, and even supercomputers [1]. These systems are characterized by their highly dynamic operations, necessitating fast responses to function effectively in critical environments, thereby presenting challenges in ensuring safety and security against cyber threats. As technology advances, cyber-attacks have become increasingly sophisticated, prompting corresponding advancements in security measures. However, these enhancements often come at the expense of greater resource consumption.

To address the risks posed by cyber threats, secure execution has emerged as a promising approach to safeguard the integrity and confidentiality of software execution. Secure execution involves running programs within a controlled environment that isolates them from the underlying operative system and other programs, thereby thwarting unauthorized access or tampering [2]. Several solutions have been introduced in the literature and the commercial industry sector to implement the concept of *secure execution*. Amongst them, Global Platform (GP), a technical standards organisation, aims to define a common standard that describes a secure environment, named Trusted Execution Environment (TEE), operating alongside the regular operating system, the Rich Execution Environment (REE), to provide a trusted execution environment for sensitive operations. The GP played a crucial role in grouping the various definitions of TEE provided over the years, which were analysed in [3]. TEE environments are designed to be tamper-resistant, isolated from the rest of the system, and cryptographically secured to protect against unauthorized access, malware, and other security threats. Although the definition provided by the GP does not specify the architectural components of a TEE in detail, it outlines the necessary APIs, structures, and software layers required for communication and security. In the realm of ARM architectures, OP-TEE [4] is renowned as the preeminent TEE solution compliant with the GP standard, universally recognized for its widespread adoption and robust security features. Its prominence within the ARM ecosystem stems from its ability to provide a secure execution environment for Trusted Applications (TA), bolstering integrity and confidentiality while enabling seamless integration into various applications and industries.

Existing literature demonstrates different solutions that leverage on the security capabilities of TEEs, particularly OP-TEE, to provide robust solutions for communication protocols. For instance, in MQT-TZ [5], OP-TEE secure storage reinforces communication in the MQTT protocol. Segarra et al. achieve this by reducing the attack surface of the MQTT broker through a two-layer encryption model that utilizes OP-TEE secure storage and environment. Wang et al. [6] instead present a key agreement algorithm fortified by Trustzone [7]. Their research primarily aims to furnish secure storage solutions for keys on Android devices. Furthermore, Valero et al. [8] delved into the realm of TEEs as applied to cloud services, presenting a solution that enables customers to execute critical tasks within secure cloud environments, called TEE-as-a-Service or TEEaaS.

This research focuses on enhancing the security of OpenSSL software [9] by implementing an engine that redirects cryptographic operations to OP-TEE. This endeavour establishes a solid foundation for accommodating all the cryptographic algorithms essential for the Transport Layer Security (TLS) protocol, utilizing both an OpenSSL Engine, which provides high portability to all Linux-based architectures, and OP-TEE, which ensures security. Unlike previous studies, the proposed solution implements secure execution locally, thus reducing dependence on cloud services [8]. Furthermore, the archi-

tecture establishes a generic access point to TEE, providing an adaptable solution suitable to diverse use cases within ARM architectures. This differs from previous research efforts, which targeted specific use cases such as communication via the MQTT protocol [5] and secure key storage in Android devices [7].

This is achieved by employing an OpenSSL Engine, which exposes various algorithms to the User Space and redirects cryptographic tasks to the TEE within OP-TEE, where they are securely executed. For instance, security assurance stems from sensitive information, such as the private key utilized for data encryption, being exclusively stored and accessed within the isolated and secure TEE, impervious to external access. Furthermore, TA, authenticated with keys stored within the OP-TEE OS, ensure that algorithm execution remains unaffected by external factors, such as vulnerabilities within OpenSSL software or the ARM architecture [10]. The architecture underwent testing utilizing the Digital Signature algorithm to assess the system's encryption and message digest operations performance on an ARM SoM based on the NXP/Freescale i.MX7 processor. This testing provides a comprehensive benchmark regarding the slowdown in performance introduced by the heightened security features, such as safe execution of cryptographic algorithms and safe storing of keys.

The structure of the manuscript is described as follows. Section II describes the methodological framework and justifies the technologies used. Section III describes the experimental setup used to benchmark the solution. Section IV presents experimental results on the OpenSSL Engine compared to the default OpenSSL implementation. Finally, Section V reports concluding remarks and future works.

II. Methodology

The proposed methodology provides a comprehensive software suite designed to enable OpenSSL utilization of the secure execution environment provided by OP-TEE for ARM architecture. The overall architecture outlined in this manuscript is depicted in Figure 1. Utilizing the OP-TEE framework, processors within the ARM architecture are segregated into two distinct environments: the *i) REE* and the *ii) TEE*.

The REE comprises the untrusted world, and its primary role is to enable communication with the user via general applications that exploit the OpenSSL software [9]. It includes the software components to support high-level access to a customized OpenSSL engine. The User Space within the REE encompasses the OP-TEE SSL Engine, an OpenSSL Engine capable of interfacing with the GP TEE Client API. An OpenSSL Engine serves as a library-defined extension mechanism within OpenSSL. At the same time, the GP TEE Client API comprises a collection of APIs for preparing data and communication parameters required for interfacing with the TEE. It is made accessible through the OP-TEE framework [11].

This component intercepts cryptographic operations, such as digital signatures, required by user-defined software applications and is triggered by the application using the OpenSSL library. Using this methodology, all operations provided by the OpenSSL API will be propagated through the OpenSSL Security Controller (SC) Engine and the GP TEE Client API. Specifically, the OP-TEE SSL Engine processes data received from the OpenSSL API and transmits it to the GP TEE Client API using three structures: *i) TEEC Context*, *ii) TEEC Session* and *iii) TEEC Operation*, as defined in the GP TEE Client API [11].

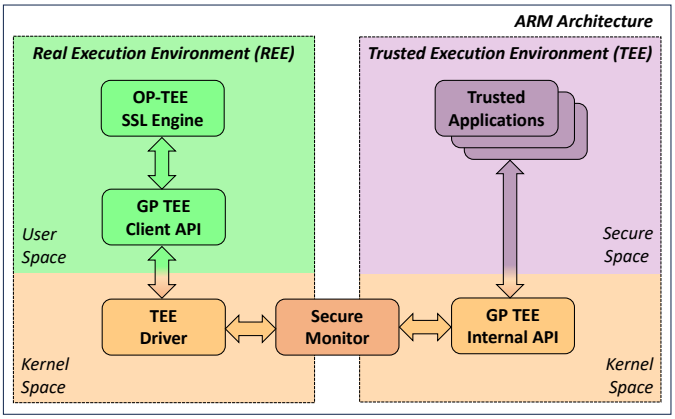

Fig. 1: OP-TEE powered OpenSSL Engine enhancing security for ARM Architectures

The TEEC Context establishes the connection between the REE and TEE, permitting access only to authorized users. The TEEC Session contains information regarding the current operation to be performed, including the type of algorithm and temporary data to be shared with the TA. At the same time, the TEEC Operation represents the single command invocation to the TA, along with the requested parameters and response. Each algorithm exposed by the OP-TEE SSL Engine must populate these structures accurately and use the GP Client API offered by the OP-TEE framework to communicate with the Kernel Space of the REE.

Within the Kernel Space, the TEE Driver operates as a driver exclusively accessible to privileged users, managing communication with the TEE. This communication is facilitated through the Secure Monitor, a dedicated instruction provided by the OP-TEE framework for transitioning from the secure world of the TEE to the non-secure world of the REE. Upon executing the Secure Monitor instruction, the CPU enters secure monitor mode, accessing all hardware components, including protected peripherals and memory regions. This enables the execution of operations within the TEE. The TEE encompasses the secure sorld, providing a secure environment conducive to executing critical tasks or storing sensitive information. Analogous to a conventional operating system, it consists of a Kernel Space and a User Space, denoted as the Secure Space in Figure 1.

Located within the Kernel Space is the GP TEE Internal API, defined by the OP-TEE framework. This component assumes various roles within the TEE, including facilitating communication between TA, managing memory allocations, and handling sessions. In the proposed solution, the GP TEE

Internal API processes data originating from the REE via the Secure Monitor, redirecting cryptographic tasks and requests to the appropriate TA residing within the Secure Space of the TEE.

Lastly, Trusted Applications are signed and secure binary applications that adhere to specific directives outlined by the Global Platform standard. Each TA provides a cryptographic algorithm corresponding to those exposed by the OP-TEE SSL Engine in the User Space of the REE. The primary function of the TA is to execute the requested cryptographic tasks in the isolated environment and, when necessary, securely store sensitive data such as private keys in the protected memory of the TEE.

III. Experimental Setup

To assess the effectiveness of the proposed architectural design, a Digital Signature algorithm has been evaluated, consisting of the combination of ECDSA with curve P-256 and SHA-256.

ECDSA P-256 represents a cryptographic algorithm utilized for digital signatures, operating within the domain of elliptic curve cryptography and using a specific curve designated as P-256 [12], as delineated in FIPS 186-4 [13]. This algorithm implementation comprises two main functions: *i)* the selection of the key and *ii)* the encryption function. The private key is stored inside the TA before the algorithm's execution and is never exposed to the REE. The correct ID and privileges are necessary when calling the corresponding TA to access it. The encryption function inputs the message digest from the OP-TEE SSL Engine and signs it with the key selected before inside the TEE, making the operation safe against software and hardware threats. On the other hand, SHA-256 serves as a cryptographic hash function that yields a 256-bit (32-byte) hash value. Renowned for its extensive application in data integrity verification and digital signatures, SHA-256 offers robust security measures against various attacks. The execution of the SHA-256 algorithm is done in a different TA, isolated and dedicated only to message digests.

The benchmarking of the proposed architecture has been performed on an ARM SoM based on the NXP/Freescale i.MX7 processor [14], which combines power-efficient ARM Cortex-A7 and Cortex-M4 cores. Within this architecture, OP-TEE was deployed to harness the capabilities of the dual-core ARM Cortex-A7 processor. Specifically, one core was designated as the REE, while the other core operated independently as the TEE.

All experiments utilised a customized Linux Kernel v5.15.71, configured through the Yocto Project to accommodate OP-TEE support. The time required for performing a Digital Signature within OP-TEE across various input sizes is evaluated to gauge the solution's performance. Subsequently, these results are compared with the base software implementation of OpenSSL. Each measurement is executed using functions from the Google Benchmark library [15], which conducts multiple iterations of the functions to ascertain a reliable execution time.

IV. Experimental Results

The performance impact of the proposed solution was assessed through three combinations of algorithms for conducting Digital Signature operations across varying input block sizes, ranging from $512\,\mathrm{B}$ to $4\,\mathrm{MB}$, as depicted in Figure 2.

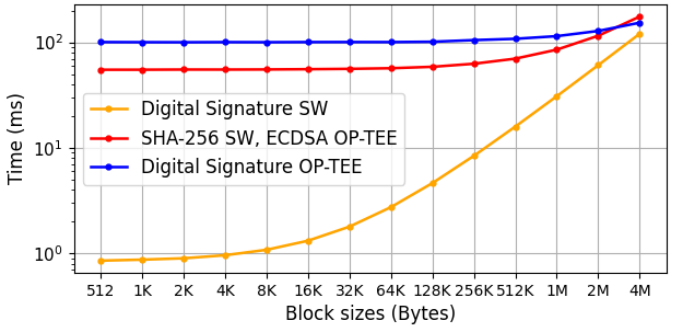

Fig. 2: Execution time for different block sizes on different scenarios, namely *i)* Digital Signature Software (SW) execution, *ii)* SHA-256 SW execution and ECDSA OP-TEE execution, and *iii)* Digital Signature OP-TEE execution.

As part of the test scenario, the first combination involved the OpenSSL standard software implementation of both ECDSA-P256 and SHA-256 algorithms, illustrated in orange within Figure 2. Subsequently, the execution time of the proposed ECDSA implementation is evaluated while retaining the default SHA-256 OpenSSL implementation, denoted by the red line in the figure. Lastly, the blue line in Figure 2 illustrates the execution time for Digital Signature operations using the proposed architecture.

The findings illustrate how the process of key selection within the TEE and the execution of algorithms within a secure environment introduce an anticipated latency for the whole digital signature of approximately $100\,\mathrm{ms}$ at worst. This latency arises due to the requirement for opening two Sessions with the appropriate operations executed within the TEE, namely *i)* the SHA-256 TA, and *ii)* the ECDSA-P256 TA. When both ECDSA-P256 and SHA-256 are executed using the proposed architecture, two sessions must be initiated, leading to the above-mentioned performance observed in the blue line of Figure 2. The red line, in contrast, demonstrates how the secure storage, retrieval, and utilization of the private key employed for ECDSA-P256 introduces only a single Session overhead over the digital signature operation, averaging around $54\,\mathrm{ms}$.

Further analysis is presented in Table I, which showcases the difference in execution time among various combinations of algorithms for three distinct block sizes: *i)* $512\,\mathrm{B}$, chosen as a benchmark for worst-case scenarios; *ii)* $32\,\mathrm{kB}$, representing a standard payload size in typical communications; and *iii)* $4\,\mathrm{MB}$, offering insights for future implementations and enhancements. SHA-256 helps alleviate the overhead incurred by the proposed architectures for larger block sizes, performing $17\,\%$ faster for $4\,\mathrm{MB}$ blocks. This can be attributed to the optimized OP-TEE implementation of the SHA-256 algorithm

TABLE I: Software execution time of the Digital Signature operation, and its decomposition into the SHA-256 and ECDSA-P256 operation with the overhead introduced by the proposed architecture.

Algorithms		**Block sizes**		
		512 B	32 kB	4 MB
SHA-256	Software	0,02 ms	0,94 ms	119,46 ms
	Overhead	+45,52 ms	+44,53 ms	-21,90 ms
ECDSA-P256	Software	0,83 ms	0,85 ms	0,98 ms
	Overhead	+54,12 ms	+54,25 ms	+54,79 ms
Digital Signature	Software	0,85 ms	1,79 ms	120,44 ms
	Overhead	+99,64 ms	+98,78 ms	+32,89 ms

for ARM architectures and the priority given to cryptographic tasks by the core dedicated to OP-TEE. In contrast, the second row reaffirms how the execution of the ECDSA-P256 algorithm within the TEE imposes a similar overhead for all block sizes, averaging around 54.3 ms, stemming from the initialization of a Session, the key selection process within the TEE, and the algorithm execution. Unlike the case of SHA-256, the performance does not scale with block sizes, as the signing operation consistently operates on a fixed block size of *ii)* 32 kB, representing the digest of the input message. Lastly, the third row represents the cumulative effect of the overheads above. It explains the disparities in execution time between the standard OpenSSL software implementation and the proposed architecture, reaching almost the same execution time for higher payloads.

It is worth noting that the implementation of SHA-256 may be perceived as redundant, as both the plaintext message and its corresponding digest are ultimately stored in variables residing within the REE. However, these results present an intriguing perspective for future implementations. For instance, SHA-256 is the foundation for various algorithms, such as Hash-based Message Authentication Code (HMAC), which require a key stored within the TEE. Moreover, SHA-256 is integral to every cypher suite in the TLS protocol, indicating that for the proposed implementation to fully support it, the SHA algorithm must be executed within a TEE.

V. Conclusion

This work presents a comprehensive approach to enhancing the security and performance of OpenSSL software for ARM architectures by leveraging the capabilities of Trusted Execution Environment, specifically OP-TEE. The proposed solution establishes a secure execution environment within the TEE, safeguarding cryptographic operations and sensitive data against cyber threats.

Through experimental evaluation, it is demonstrated that while the proposed architecture introduces a latency overhead due to secure key selection and algorithm execution within the TEE, the overall performance impact is manageable. The results reveal that for larger block sizes, the overhead incurred by the proposed architecture is mitigated by the optimized implementation of cryptographic algorithms within OP-TEE, particularly for SHA-256.

Furthermore, the study highlighted the potential for future enhancements, including full support for the TLS protocol and integration with additional cryptographic algorithms. As cyber threats continue to evolve, leveraging TEEs like OP-TEE represents a crucial step towards mitigating risks and safeguarding sensitive information across a diverse range of applications and industries. By providing a secure and adaptable solution for ARM architectures, this research contributes to strengthening the resilience of OpenSSL software against evolving cyber threats.

Acknowledgment

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 manuscript reflects only the authors' views and opinions, and the Ministry cannot be considered responsible for them.

References

[1] T. N. Rahman, N. Khan, and Z. I. Zaman, "Redefining computing: Rise of arm from consumer to cloud for energy efficiency," *arXiv preprint arXiv:2402.02527*, 2024.

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

[3] O. Hosam and F. BinYuan, "A comprehensive analysis of trusted execution environments," in *2022 8th International Conference on Information Technology Trends (ITT)*, 2022, pp. 61–66.

[4] A. Nehal and P. Ahlawat, "Securing iot applications with op-tee from hardware level os," in *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2019, pp. 1441–1444.

[5] C. Segarra, R. Delgado-Gonzalo, and V. Schiavoni, "Mqt-tz: Hardening iot brokers using arm trustzone : (practical experience report)," *IEEE*, 2020.

[6] Y. Wang, W. Gao, X. Hei, I. Mungwarama, and J. Ren, "Independent credible: Secure communication architecture of android devices based on trustzone," *IEEE*, 2020.

[7] N. Bernard, D. Martin, A. Bailey, H. Cho, and S. Martin, "Trustzone explained: Architectural features and use cases," *null*, 2016.

[8] J. M. J. Valero, P. M. S. Sánchez, A. Lekidis, P. Martins, M. G. Pérez, A. H. Celdrán, and G. M. Pérez, "Trusted execution environment-enabled platform for 5g security and privacy enhancement," *null*, 2022.

[9] O. S. Foundation, "Source code for the openssl software," 1998, https://github.com/openssl/openssl.

[10] T. N. Rahman, N. Khan, and Z. I. Zaman, "Redefining computing: Rise of arm from consumer to cloud for energy efficiency," *World Journal of Advanced Research and Reviews*, vol. 21, no. 1, p. 817–835, Jan. 2024. [Online]. Available: http://dx.doi.org/10.30574/wjarr.2024.21.1.0017

[11] G. P. Group, "Global platform client api documentation," 2023, optee.readthedocs.io/en/stable/architecture/globalplatform_api.html.

[12] M. Adalier and A. Teknik, "Efficient and secure elliptic curve cryptography implementation of curve p-256," in *Workshop on elliptic curve cryptography standards*, vol. 66, no. 446, 2015, pp. 2014–2017.

[13] T. A. Hall and S. S. Keller, "The fips 186-4 elliptic curve digital signature algorithm validation system (ecdsa2vs)," *National Institute of Standards and Technology: Gaithersburg, MD, USA*, 2010.

[14] V. LTD., "Nxp/freescale i.mx7 datasheet," 2016, https://www.variscite.com/wp-content/uploads/2017/12/VAR-SOM-MX7_VAR-SOM-MX7-5G_datasheet.pdf.

[15] Google, "Source code for the google benchmark library," 2023, https://github.com/google/benchmark?tab=readme-ov-file.