Integrating OpenTitan as a Security Controller for Cryptographic Tasks in RISC-V SoCs

Integrating OpenTitan as a Security Controller for Cryptographic Tasks in RISC-V SoCs

Alberto Musa[1], Emanuele Parisi[1], Luca Barbierato[2], Edoardo Patti[2], Andrea Acquaviva[1] and Francesco Barchi[1]

[1]*Università di Bologna, Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi", Bologna, Italy*
[2]*Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy*

Abstract

RISC-V architectures are increasingly utilized in security-critical embedded systems, with OpenTitan standing out as a prominent open-source silicon Root-of-Trust. OpenTitan delivers essential functionalities, such as secure boot and execution integrity, but introduces notable area overhead. To mitigate this issue, we propose TitanSSL, a secure software stack that offloads cryptographic operations to OpenTitan. TitanSSL comprises an OpenSSL backend, a Linux driver for system communication, and a custom OpenTitan firmware. Communication between components is facilitated through a custom Application Binary Interface (ABI), ensuring the driver remains independent of the specific cryptographic operations executed by the OpenSSL engine. This design supports extensibility, enabling the integration of additional cryptographic primitives and enhancing the system flexibility. We evaluated TitanSSL on a System-on-Chip (SoC) featuring a CVA6 application core running Linux and OpenTitan, both clocked at 40 MHz on a Xilinx VCU118 FPGA. Our results reveal that, while there is communication overhead between system components, it is outweighed by substantial performance gains. Specifically, TitanSSL achieves speedups of 40x for SHA-256 and 82x for AES-256-CBC compared to a software-only implementation running on the CVA6 core. In addition, we provide design guidelines for future optimizations to improve system performance.

Keywords

RISC-V, Secure Systems, Software Stack

1. Introduction

The rise of open-hardware System-on-Chip (SoC) technology is reshaping various application domains, particularly in safety-critical and security-critical Cyber-Physical Systems (CPS). These systems require real-time responsiveness and dynamic operations to function effectively in demanding environments. However, these attributes introduce challenges in maintaining safety and resilience against cyber threats. To address these challenges, modern SoCs integrate specialized hardware components to strengthen security. For example, [1] presents an architecture in which an application processor and a silicon Root-of-Trust (RoT) coexist within the same SoC to enable secure boot functionality. More recently, [2] combines the CVA6 RISC-V application processor [3] with OpenTitan, an open-source silicon RoT [4].

The integration of OpenTitan as a Root-of-Trust offers a secure and isolated execution environment. However, this design comes at the cost of significant area overhead. For instance, [2] shows that OpenTitan's footprint exceeds 75% of the host platform's area. To justify this overhead, it is essential to maximize OpenTitan's capabilities, particularly its hardware support for cryptographic operations, which can accelerate widely adopted security libraries.

Building on this motivation, our first contribution investigates the following research question: "Given an OpenTitan-based SoC, can its memory isolation and cryptographic acceleration features be utilized to provide a secure backend for OpenSSL?" Integrating OpenTitan with OpenSSL removes the need for additional external security modules and leverages OpenTitan's private memory to securely store cryptographic keys and secrets, preventing their exposure in main memory. To achieve this integration, we develop an end-to-end software stack, comprising: *i)* an OpenSSL engine that interfaces with a Linux driver, *ii)* a Linux driver for managing secure communication between the host system

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT
alberto.musa@unibo.it (A. Musa); emanuele.parisi@unibo.it (E. Parisi); luca.barbierato@polito.it (L. Barbierato); edoardo.patti@polito.it (E. Patti); andrea.acquaviva@unibo.it (A. Acquaviva); francesco.barchi@unibo.it (F. Barchi)

and OpenTitan, and *iii)* a custom firmware for OpenTitan's Security Controller (SC), a RISC-V core responsible for executing cryptographic operations.

The second contribution of this work is the implementation of the software stack with the following key functional requirements: *i)* full utilization of OpenTitan's cryptographic accelerators, *ii)* secure management of secrets and cryptographic key generation, *iii)* seamless support for application-level virtual memory by the SC, and *iv)* communication between the application processors and the SC via a dedicated interface.

The third contribution is the validation and evaluation of our system. We characterize and analyze the performance of TitanSSL compared to software-only implementations and theoretical limits of accelerators.

Our experimental results demonstrate that, for typical cryptographic workloads, the overhead introduced by our TitanSSL software stack is outweighed by the performance gains provided by OpenTitan's accelerators. For example, encrypting a 2048-byte payload with AES-256 in CBC mode achieves a 16.5x speed-up compared to a pure software implementation. Similarly, SHA-256 processing of a 2048-byte payload results in a 3.8x speed-up. With a payload of just 16 KiB, the speedup for SHA increases to approximately 40x, while for AES, it exceeds 80x. Under these conditions, the accelerator utilization in OpenTitan achieves 50% of its ideal performance, as calculated in the absence of memory accesses. These results validate OpenTitan's effectiveness as a Security Controller and highlight opportunities for hardware and software optimizations in future designs.

The structure of the manuscript is described as follows. Section 3 describes the hardware architecture used for our experimental setup. Section 4 reports the software framework and the technologies used. Section 5 defines the analysis conducted on the system to establish and ensure its security measures. Section 6 explains the characterization methodology adopted to evaluate the solution, and it presents experimental results comparing the proposed software stack with the default OpenSSL implementation. Finally, Section 7 reports concluding remarks and future works.

2. Background and Related Works

Our work envisions a scenario where OpenTitan is integrated into the same System-on-Chip (SoC) alongside a RISC-V application core that lacks built-in cryptographic hardware support or other execution isolation mechanisms, as highlighted in [3]. In this configuration, OpenTitan acts as a Security Controller (SC), providing advanced security functionalities through its hardware accelerators and memory isolation capabilities, as detailed in [4]. Specifically, the OpenTitan core is responsible for executing cryptographic primitives, leveraging dedicated hardware accelerators, and isolating sensitive secrets from the application processor.

In the literature, similar implementations are found in solutions like EdgeLock Secure Enclave [5] and Keystone [6].

The EdgeLock Secure Enclave is a proprietary security subsystem integrated into NXP processors. It employs hardware-level isolation and cryptographic protections to ensure data integrity and confidentiality while delivering robust security features. However, as a proprietary solution, EdgeLock poses significant challenges when porting or replicating its functionality for custom architectures, limiting its applicability in open and flexible SoC designs.

Keystone, on the other hand, is an open-source framework for creating secure software environments (Enclaves) on RISC-V platforms. It achieves physical memory isolation through RISC-V Physical Memory Protection (PMP) [7], which restricts access to specific memory regions, even from privileged software. Keystone's versatility makes it suitable for diverse architectures and use cases. However, its reliance on a software-based Secure Monitor introduces overhead, as Enclaves share the same processor with other system components. This design choice may result in performance degradation, particularly in security-critical workloads requiring frequent context switches.

Other works in the literature propose and analyze various RISC-V architectures, focusing on their security constraints and requirements [8]. For instance, the survey by T. Lu [9] highlights the increasing

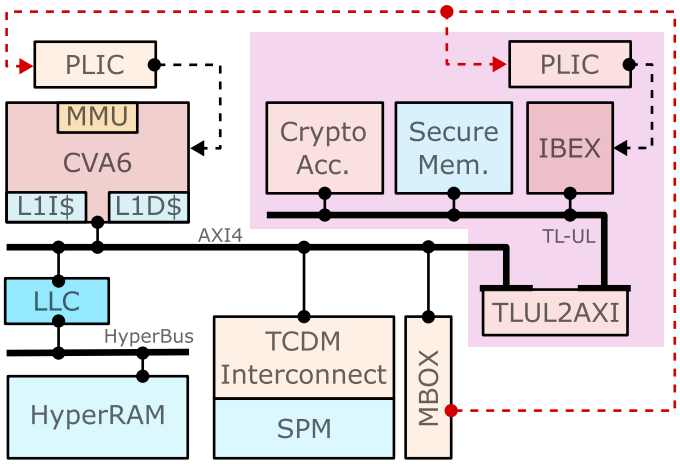

Figure 1: Outline of the SoC. The NoC buses and the memory hierarchy are highlighted.

adoption of RISC-V in the research community, especially in security applications. The survey discusses how the openness of RISC-V enables rigorous auditing, thereby enhancing hardware resilience against cyber threats. Proposed solutions span both software frameworks [10, 11] and hardware approaches [12]. For example, Schrammel et al. [12] introduce a hardware-based solution that optimizes context switches while providing effective isolation mechanisms for RISC-V architectures.

Programmable Root-of-Trust (RoT) intellectual properties (IPs), such as OpenTitan, have traditionally been employed for secure boot and security services. However, their potential as Security Controllers (SC) remains largely unexplored. A key advantage of this approach is that the isolation mechanisms are independent of any specific Operating System (OS) or architectural features of the host processor. This decoupling allows OpenTitan to be integrated alongside various processors, enabling it to act as a dedicated Security Controller. With hardware cryptographic accelerators, OpenTitan can significantly improve the performance of cryptographic tasks while maintaining a high degree of security, flexibility, and portability.

As far as we know, this work represents one of the first attempts to systematically characterize the trade-offs and microarchitectural implications of using OpenTitan as a Security Controller to support cryptographic tasks. Furthermore, we provide an OpenSSL-based evaluation demonstrating the practical benefits of this approach.

3. Hardware Architecture

The reference platform considered in this work is shown in Figure 1. It represents a state-of-the-art RISC-V based system, such as the one presented in [2] for secure autonomous Micro-Aerial Vehicles.

The considered platform features three main components: *i*) a RV64GC Application Core, namely CVA6, *ii*) a Programmable Multi-Core Accelerator, and *iii*) OpenTitan, the hardware Root-of-Trust. CVA6 [3] is a Linux-capable RV64GC core featuring a six-stage, single-issue, in-order pipeline. It is meant to run general-purpose platform configuration tasks and to manage the system peripherals. The Programmable Multi-Core Accelerator consists of a cluster of eight CV32E40-based cores [13] with custom ISA extensions designed to accelerate computationally intensive DSP and machine learning tasks. OpenTitan is an open-source silicon Root-of-Trust inspired by Titan [14]. It consists of a set of hardware accelerators to enable efficient computation of common cryptographic primitives, plus the OpenTitan Big Number core (OTBN), a programmable coprocessor for asymmetric public-key cryptography. Sensitive data are stored in the OpenTitan internal ROM and scratchpad, two tamper-proof memories featuring scrambling functionalities.

All Root-of-Trust functions are managed by Ibex [15], an RV32IMC microcontroller optimized for embedded control applications. Additionally, OpenTitan features a range of protection against physical

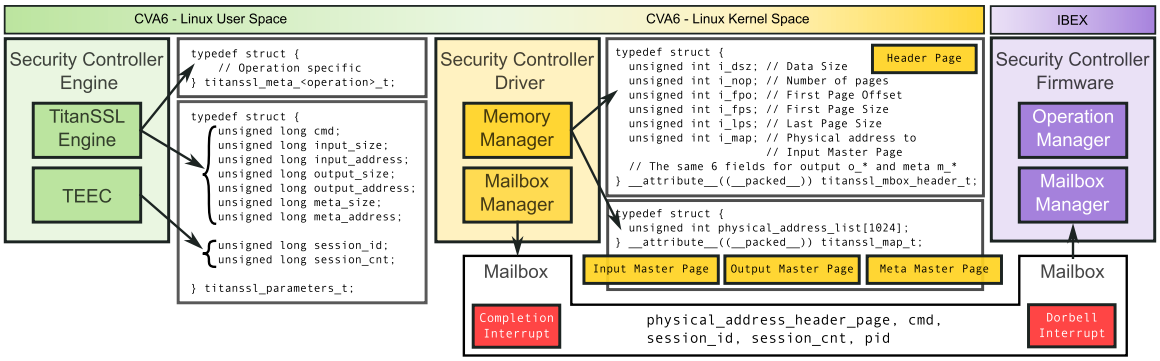

Figure 2: Software stack ranging from CVA6 to IBEX using the mailbox as the communication system. Data structures are highlighted to ensure driver agnosticity.

attacks, including power analysis countermeasures and tamper detection circuits. As mentioned, the target design integrates OpenTitan within the same silicon as the Application Processor and the Programmable Multi-Core Accelerator. A MailBox mediates communication between the CVA6 Application Core and OpenTitan, as detailed in Section 4; it consists of a shared memory region meant for data sharing, plus two specialized registers, namely Doorbell and Completion, whose LSB is connected to the interrupt controller of Ibex and CVA6 respectively, to implement an easy asynchronous acknowledging system between the two cores. The memory Hierarchy employed by the CVA6 Application Processor and the RoT controller consists of three levels: private cache, Last-Level Cache (LLC), and main memory. The CVA6 core's private memory encompasses two L1 set-associative caches, a 16 KiB of L1 I-cache and a 32 KiB write-through L1 D-cache. The IBEX core lacks a cache memory but can access a 32 KiB SRAM scratchpad exclusive to the RoT system. Both systems share access to the LLC, an eight-way set-associative cache with 128 KiB, and 256 lines. Finally, the SoC utilizes a 32 MiB main memory composed of four HyperRAM chips, each with a 64 Mbit capacity, situated off-chip.

4. TitanSSL Software Architecture

The TitanSSL software stack is shown in Figure 2. As described in section 3, it redirects OpenSSL cryptographic tasks, such as SHA-256, RSA, and AES, required by applications running on the Application Processor to the Security Controller (the OpenTitan core). As a result, the TitanSSL software stack is divided into two parts: one that operates on the Application Processor and another that runs on the Security Controller.

The software stack within the Application Processor is essential for enabling secure communication between untrusted environments, where applications utilize the OpenSSL library [16], and the Security Controller. At the core of this stack is the TitanSSL Security Controller Engine (TitanSSL-SCE), which integrates OpenSSL functionalities with the Global Platform standard [17] through the implementation of the Global Platform Client API [18] (represented by the TEEC component in the figure). This component establishes secure communication with the TitanSSL Security Controller Driver (TitanSSL-SCD) and, ultimately, the TitanSSL Security Controller Firmware (TitanSSL-SCF).

The TitanSSL-SCD is the final component of the software stack residing in the Application Processor. Its role is to facilitate data exchange between the TitanSSL-SCE and the TitanSSL-SCF, using a communication mailbox to transfer information in both directions. The software components operating within the Application Processor are categorized into two spaces: the User Space (depicted in green) and the Kernel Space (depicted in yellow), as shown in Figure 2.

The components running in the User Space include the two key elements of the TitanSSL-SCE. These components intercept cryptographic operations such as encryption and digital signature generation, which are triggered by user-defined applications utilizing the OpenSSL library.

The first component of the TitanSSL Security Controller Engine is the OpenSSL Engine, a library extension mechanism defined by OpenSSL. Its purpose is to redirect the execution of cryptographic

operations to the Security Controller. This engine interfaces with the Global Platform Client API (TEEC), a library specification designed to manage security devices [18].

The OpenSSL engine has two main responsibilities: i) Initializing a communication context (`TEEC_Context`) and a communication session (TEEC_Session) as specified by the GP Client API; and ii) Redirecting cryptographic requests (e.g., SHA-256, AES, RSA) to the lower layers of the system.

The second component of the TitanSSL Security Controller Engine is the Global Platform Client API implementation, which abstracts the underlying Security Controller to the OpenSSL Engine. This abstraction ensures the secure establishment of a communication channel with the TitanSSL-SCD, facilitating proper data exchange through TEEC_Context and TEEC_Session. It also manages the transfer of the necessary data for cryptographic operations to Kernel Space, using the TEEC_Operation structure.

Contextual information is passed to the next layer through a data structure called titanssl_parameters_t. To maintain driver agnosticism, the engine encapsulates all operation-specific information within a meta-information data structure. For instance, AES algorithms require initialization vectors (IVs) or keys, while RSA operations demand exponents and modulus values to handle asymmetric keys. Each cryptographic algorithm needing such supplementary information is represented by a dedicated data structure, such as titanssl_meta_<operation>_t.

The TitanSSL-SCD, residing in the Kernel Space, is responsible for managing requests directed to the Security Controller and enabling communication between the Application Processor and the Security Controller via the mailbox. This component exposes its functionalities through a character device, accessible to the TitanSSL-SCE. When a command is issued, the Device Driver processes the data by translating virtual addresses from User Space to physical addresses, which the Security Controller can utilize. This translation occurs in two steps: i) Pinning the memory page to prevent data corruption, and ii) Translating the virtual address to a corresponding physical address.

The TitanSSL-SCD includes a Mailbox Manager responsible for managing data exchange between the Application Processor and the Security Controller.

The Application Binary Interface (ABI) designed for sending information to the firmware consists of: i) Three Master Pages, which store the physical addresses of the input, output, and meta-information memory pages (provided by the application and engine); ii) A Header Page, which contains navigation information for the physical input, output, and meta-information pages, including the physical address of the Master Pages, the number of pages, offsets, and page sizes; iii) Data for the mailbox, which includes the physical address of the Header Page, the requested command, session information, and the process id (pid) of the application invoking the operation.

Additionally, the mailbox triggers interrupts to the IBEX and CVA6 cores using the Doorbell and Completion registers.

The TitanSSL Security Controller Firmware (TitanSSL-SCF) is responsible for executing cryptographic operations within the secure environment provided by OpenTitan. Upon receiving a command from the Mailbox, the TitanSSL-SCF extracts the requested operation, its parameters, and the Header Page's memory address. By accessing physical memory directly, the firmware retrieves the data structure sent by the Application Processor, which contains all the information to execute the requested cryptographic operation.

The Mailbox Manager in the Security Controller remains idle until it is signaled by the Doorbell register. Upon receiving the signal, it fetches the cryptographic task request and the physical memory address of the structure containing the operation's inputs and parameters. The Cryptographic Accelerator Manager then processes the requested operation, which may involve tasks such as generating a hidden ephemeral key, signing or verifying a signature, computing a cryptographic hash, or encrypting a data sequence.

Once the cryptographic task is completed, the Mailbox Manager activates the Completion register, signaling the Mailbox Manager on the Application Processor to proceed with subsequent actions.

The system employs a zero-copy approach, where the firmware running on IBEX writes the output directly to memory without using an intermediate buffer or delegating tasks to the driver. Both the CVA6 application processor and IBEX access memory through the last-level cache (LLC), though CVA6

features an additional level of cache. To maintain memory consistency, the driver flushes the first-level cache before initiating the remote procedure call to IBEX. This call is blocking, ensuring that memory remains unaltered by the application during OpenTitan operations. However, in multithreaded or shared memory environments, it is the application's responsibility to prevent concurrent access to buffers being used by OpenTitan.

5. Security Assumptions and Implications

The security assumptions for our system are based on the threat model outlined in the OpenTitan analysis for SCRAMBLE-CFI [19]. In this analysis, the authors examine how the OpenTitan chip can withstand attacks when an adversary gains physical access to the system. They describe how OpenTitan is inherently equipped with countermeasures designed to thwart such attacks, preventing the manipulation of stored and transmitted data, as well as the instructions within the chip. Consequently, the threat model for this section assumes that the attacker does not have physical access to the system but can manipulate input data to gain unauthorized privileges to execute cryptographic tasks or extract sensitive information stored within OpenTitan, all without needing direct physical access.

Our analysis combines the STRIDE framework [20], the OWASP Top 10 [21], and CWE [22] classifications to assess potential software vulnerabilities. We have excluded certain OWASP vulnerabilities that do not apply to the analyzed system. The remaining vulnerabilities are mapped to STRIDE categories based on the potential attacks they may enable if exploited. This threat model covers the entire system architecture, assigning criticality levels to components ranging from 0 (not evaluated) to 9 (involving sensitive data). Sensitive data categories include secrets and critical physical addresses, essential for system security.

Our evaluation uses the STRIDE approach to assess both software components and data flows: *i)* Software components are analyzed for vulnerabilities related to Spoofing, Repudiation, and Elevation of Privilege. *ii)* Data flows are examined for risks of Tampering, Information Disclosure, and Denial of Service. It is important to note that the Application and its data flows are not included in this analysis. In this context, the OpenSSL Engine, which is rated with a criticality level of 1, operates with minimal privileges and does not store sensitive data. Since the OpenSSL Engine is accessible to all users, an analysis of Spoofing and Repudiation risks is deemed unnecessary. Moreover, Elevation of Privilege concerns are addressed by subsequent authentication layers.

In contrast, the GP Client API is assigned a criticality level of 2 due to its role as the gateway to Kernel Space on the Host Processor. This component incorporates authentication mechanisms that help mitigate Spoofing and Elevation of Privilege threats. Furthermore, the Session feature in the GP standard tracks access by processes, providing logging and monitoring capabilities to prevent Repudiation.

Within Kernel Space, the Device Driver (criticality level 5) helps mitigate threats related to Broken Access Control and Broken Authentication, thereby limiting the risk of Spoofing and Elevation of Privilege attacks. The kernel's logging system also addresses Repudiation and Logging concerns. Both the Kernel Space's Mailbox Drivers and the Secure Space follow a common architectural framework, although they differ in execution privileges (5 vs. 7) due to their distinct logging and monitoring capabilities.

The Secure Firmware is isolated by default and remains secure, with its primary vulnerability stemming from the potential tampering of mailbox data. Our data flow analysis specifically targets the potential leakage of sensitive information between the Device Driver and the Host Processor's mailbox, as such a breach could jeopardize the entire system architecture. It is worth noting that other data flows are protected against vulnerabilities such as Incorrect Input Validation or Buffer Overflow, effectively preventing unauthorized access beyond buffer boundaries.

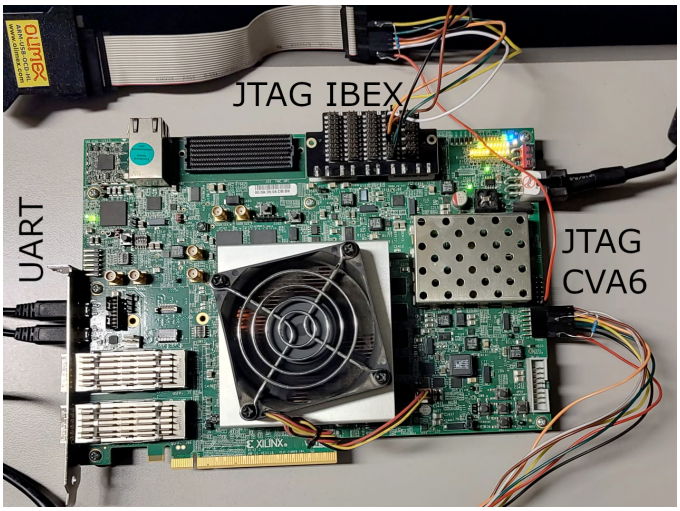

Figure 3: Experimental setup on a Xilinx VCU118. Two Olimex JTAG adapter are used to program the CVA6 and IBEX core.

6. Experimental Results

To evaluate the effectiveness of the proposed software stack, we analyze the performance of two widely used cryptographic algorithms: SHA-256 and AES-256-CBC. Specifically, we compare the runtime performance of cryptographic operations offloaded to OpenTitan against a purely software-based implementation executed on the host core. TitanSSL was tested on a System-on-Chip (SoC) synthesized on a Xilinx Virtex UltraScale+ VCU118 FPGA (the experimental setup is shown in Figure 3), running Linux v5.10.7. The OpenTitan subsystem provides access to the JTAG interface through the FMC connector and the CVA6 processor via the PMOD slot on the board. Programming of the system is carried out using GDB in conjunction with OpenOCD, while interactions with the Linux shell are facilitated through the UART interface available on the FPGA.

The evaluation of the system was conducted directly on the Linux system running on CVA6 using the openssl speed command, specifying: i) the developed engine, ii) the payload size, and iii) the algorithm to be tested through the engine and the envelope library (EVP). For example, to evaluate the performance of SHA-256 and AES-256-CBC with a 1 KiB payload, the following commands can be used:

Table 1
Results obtained with openssl speed on the system running on the FPGA at 50MHz

Payload	SHA [KiB/s]		Speedup	OT	AES [KiB/s]		Speedup	OT
[Byte]	OpenSSL	TitanSSL		Limit	OpenSSL	TitanSSL		Limit
16	43	12	0.29x	0.0 %	216	52	0.24x	0.1 %
32	87	31	0.36x	0.1 %	239	140	0.59x	0.4 %
48	130	36	0.28x	0.1 %	248	177	0.72x	0.5 %
64	118	55	0.46x	0.2 %	250	191	0.76x	0.5 %
112	194	82	0.42x	0.3 %	259	394	1.52x	1.0 %
128	178	116	0.65x	0.4 %	260	350	1.35x	0.9 %
240	257	137	0.53x	0.5 %	264	627	2.38x	1.6 %
256	238	186	0.78x	0.6 %	264	402	1.52x	1.0 %
496	303	169	0.56x	0.6 %	265	386	1.46x	1.0 %
512	288	382	1.33x	1.3 %	265	578	2.18x	1.5 %
1008	330	705	2.13x	2.4 %	267	4233	15.86x	10.8 %
1024	320	537	1.68x	1.8 %	267	3840	14.38x	9.8 %
2032	347	1136	3.27x	3.9 %	268	3501	13.08x	9.0 %
2048	341	1300	3.82x	4.4 %	268	4436	16.57x	11.4 %
4080	355	2049	5.78x	7.0 %	267	4352	16.33x	11.1 %
4096	352	1850	5.25x	6.3 %	266	4338	16.30x	11.1 %
16384	357	14222	39.80x	48.5 %	259	21333	82.45x	54.6 %

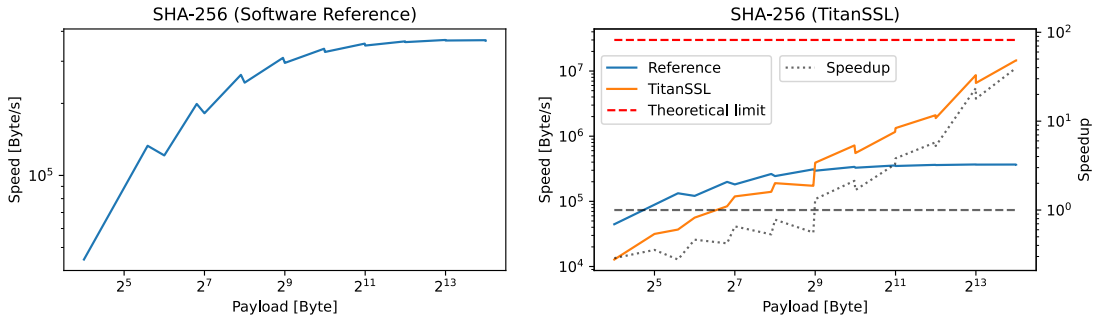

Figure 4: SHA2-256 Results

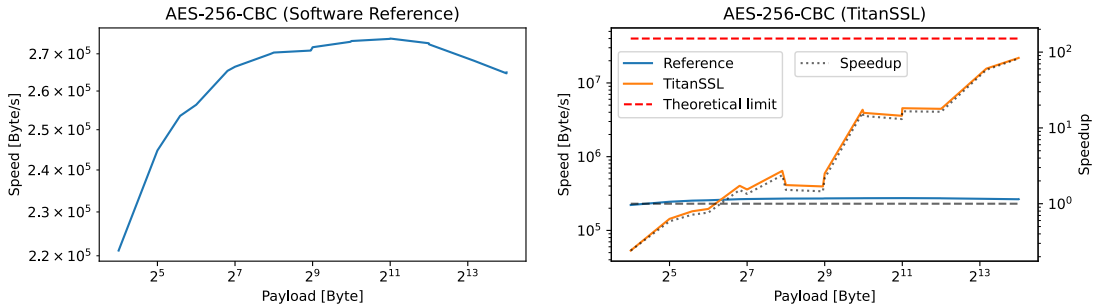

Figure 5: AES-256-CBC Results

```
openssl speed -engine titanssl -bytes 1024 -mr -evp sha256;
openssl speed -engine titanssl -bytes 1024 -mr -evp aes-256-cbc;
```

The output of the first command is structured as follows:

```
+DT:sha256:3:1024
+R:126169:sha256:3.000000
+F:22:sha256:43065685.33
```

This output can be interpreted as follows:

- +DT:sha256:3:1024: Indicates the start of a new test session for a digest algorithm. Specifically, this test is for the SHA-256 hashing algorithm. It specifies the test duration in seconds (3 seconds in this case) and the block size (in bytes) used during the test.
- +R:126169:sha256:3.000000 Represents a result record. Here, 126169 is the total number of data blocks (each of size 1024 bytes) processed during the 3-second test.
- +F:22:sha256:43065685.33: Provides the calculated throughput at the end of the test. The value 22 denotes a numeric code representing a specific unit or metric (for SHA-256, it refers to throughput in bytes per second). The value 43065685.33 represents the measured throughput in bytes per second, indicating the rate at which the SHA-256 algorithm processed data during the test

We also evaluate the performance of our system against the theoretical maximum throughput of the accelerators. OpenTitan features an SHA accelerator, which processes 64 bytes per 80 clock cycles when operating in SHA2-256 mode. Similarly, the AES accelerator, configured in unmasked mode, processes 16 bytes per 16 clock cycles. Given that the FPGA system operates at a clock frequency of 40 MHz, the theoretical maximum performance for the test system is 28.61 MiB/s for SHA2-256 and 38.15 MiB/s for AES-256. These values represent ideal conditions, excluding overheads such as data movement between memories or the encryption schemes applied to the algorithms.

Table 1 presents the performance results for SHA2-256 and AES-256-CBC, comparing the software implementation in OpenSSL with our TitanSSL software stack. For each test, operations were performed

on varying payload sizes. To observe the effects of memory paging, payload sizes slightly smaller than multiples of 4 KiB were also tested, allowing us to capture the overhead introduced by managing additional pages.

The table columns display, for both algorithms, the operation speed in KiB/s, the speedup achieved by TitanSSL relative to OpenSSL, and the percentage of TitanSSL's performance compared to the theoretical limits of the corresponding hardware accelerator.

In addition to providing secure in-element processing, our results show that the overhead of the TitanSSL software stack is compensated at 512 bytes for SHA2-256 and 128 bytes for AES-256. For larger payloads, TitanSSL achieves significant speedup over the pure software implementation, reaching 3.8x for SHA2-256 and 16.5x for AES-256 with 2 KiB payloads. For a payload size of 16 KiB, the speedup rises to 39.8x for SHA2-256 and 82.4x for AES-256, achieving approximately 50% of the theoretical performance of the hardware accelerators.

Figures 4 and 5 illustrate the performance trends for operations performed using OpenSSL and TitanSSL. In both figures, the left-hand graph shows the performance of the software-only implementation, which peaks at a payload size of 2^{11}, equivalent to 2 KiB, for both algorithms. The right-hand graph compares the performance of TitanSSL to the software baseline and the theoretical performance of the hardware accelerator. Both algorithms demonstrate a performance trend that increases with payload size, converging toward the ideal performance. Drops in performance are visible due to the overhead of handling additional memory pages and any padding required by the cryptographic algorithms. The gray horizontal line in the graphs highlights the point at which the overhead of the software stack is neutralized, resulting in a speedup greater than 1x. We also tested RSA with 1024-bit keys obtaining a speedup of 1.4x in signing operations and 1.0x in verification operations.

7. Conclusion

In this paper, we introduced a software stack that utilizes OpenTitan as a Security Controller (SC) within a RISC-V-based System-on-Chip (SoC). This stack enables applications operating in insecure environments to securely interact with the OpenTitan SC for cryptographic operations, as well as the generation and storage of sensitive data. To facilitate this integration, we developed a custom OpenSSL engine (TitanSSL-SCE), a communication driver (TitanSSL-SCD), and firmware for the Security Controller (TitanSSL-SCF). As a result, applications can perform cryptographic operations through the OpenSSL API, leveraging the security and acceleration features provided by OpenTitan.

To evaluate the performance of the TitanSSL-SCE and TitanSSL-SCD components, we conducted benchmarks on a Linux environment running on a System-on-Chip (SoC) equipped with a CVA6 application core and OpenTitan. The SoC was implemented entirely on an FPGA, operating at a frequency of 40 MHz. All evaluations were performed directly on the system under test using the OpenSSL speed application.

Although the TitanSSL software stack introduces some overhead, it achieved significant performance improvements; for payloads of 16KiB, we obtained speed-ups of 39.8x for SHA-WITH RESP256 and 82.4x for AES-256-CBC. For payloads of 16KiB, TitanSSL reached approximately 50% of the ideal performance achievable by the underlying accelerators. Moreover, TitanSSL consistently outperforms OpenSSL, even with smaller payloads as low as 512 bytes. These performance gains are realized while leveraging key features of OpenTitan, such as its memory isolation, secure boot, and root-of-trust capabilities.

In future work, we plan to extend TitanSSL's evaluation by testing it with real-world network workloads, including those generated by communication protocols such as HTTPS and TLS.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT, Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the authors reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

[1] B. Parno, J. M. McCune, A. Perrig, Roots of Trust, Springer New York, New York, NY, 2011, pp. 35–40. URL: https://doi.org/10.1007/978-1-4614-1460-5_6. doi:10.1007/978-1-4614-1460-5_6.

[2] M. Ciani, et al., Cyber security aboard micro aerial vehicles: An opentitan-based visual communication use case, in: 2023 IEEE International Symposium on Circuits and Systems (ISCAS), 2023, pp. 1–5. doi:10.1109/ISCAS46773.2023.10181732.

[3] F. Zaruba, L. Benini, The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27 (2019) 2629–2640. doi:10.1109/TVLSI.2019.2926114.

[4] lowRISC CIC, Opentitan official documentation, 2019. https://opentitan.org/book/doc/introduction.html.

[5] NXP, Edgelock secure enclave, 2019. https://www.nxp.com/products/nxp-product-information/nxp-product-programs/edgelock-secure-enclave:EDGELOCK-SECURE-ENCLAVE.

[6] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, D. Song, Keystone: An open framework for architecting trusted execution environments, in: Proceedings of the Fifteenth European Conference on Computer Systems, EuroSys '20, Association for Computing Machinery, New York, NY, USA, 2020. URL: https://doi.org/10.1145/3342195.3387532. doi:10.1145/3342195.3387532.

[7] K. Cheang, C. Rasmussen, D. Lee, D. W. Kohlbrenner, K. Asanović, S. A. Seshia, Verifying risc-v physical memory protection, 2022. arXiv:2211.02179.

[8] M. R. Fadiheh, D. Stoffel, C. Barrett, S. Mitra, W. Kunz, Processor hardware security vulnerabilities and their detection by unique program execution checking, in: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 994–999. doi:10.23919/DATE.2019.8715004.

[9] T. Lu, A survey on risc-v security: Hardware and architecture, 2021. arXiv:2107.04175.

[10] G. Andrade, D. Lee, D. Kohlbrenner, K. Asanovic, D. Song, Software-based off-chip memory protection for risc-v trusted execution environments, UC Berkeley (2020).

[11] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, K. Asanović, Keystone: An open framework for architecting tees, 2019. arXiv:1907.10119.

[12] D. Schrammel, S. Weiser, S. Steinegger, M. Schwarzl, M. Schwarz, S. Mangard, D. Gruss, Donky: Domain keys–efficient in-process isolation for risc-v and x86, in: Proceedings of the 29th USENIX Conference on Security Symposium, 2020, pp. 1677–1694.

[13] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, L. Benini, Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices, IEEE Trans. Very Large Scale Integr. Syst. 25 (2017) 2700–2713. URL: https://doi.org/10.1109/TVLSI.2017.2654506. doi:10.1109/TVLSI.2017.2654506.

[14] S. Johnson, D. Rizzo, P. Ranganathan, J. McCune, R. Ho, Titan: enabling a transparent silicon root of trust for cloud, in: Hot Chips: A Symposium on High Performance Chips, volume 194, 2018.

[15] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, L. Benini, Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications, in: 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017, pp. 1–8. doi:10.1109/PATMOS.2017.8106976.

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

[17] G. P. Group, Global platform official website, 2023. https://globalplatform.org/.

[18] G. P. Group, Global platform client api documentation, 2023. https://optee.readthedocs.io/en/stable/architecture/globalplatform_api.html#tee-client-api.

[19] P. Nasahl, S. Mangard, Scramble-cfi: Mitigating fault-induced control-flow attacks on opentitan, 2023. arXiv:2303.03711.

[20] B. Potter, Microsoft sdl threat modelling tool, Network Security 2009 (2009) 15–18.

[21] M. Bach-Nutman, Understanding the top 10 owasp vulnerabilities, arXiv preprint arXiv:2012.09960 (2020).

[22] C. W. Enumeration, 2022 cwe top 25 most dangerous software weaknesses, 2022. https://cwe.mitre.

org/top25/archive/2022/2022_cwe_top25.html.