

Received March 13, 2020, accepted April 7, 2020, date of publication April 13, 2020, date of current version May 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2987617

Quick Boot of Trusted Execution Environment With Hardware Accelerators

TRONG-THUC HOANG^{1,2}, (Graduate Student Member, IEEE),
CKRISTIAN DURAN¹, (Student Member, IEEE),
DUC-THINH NGUYEN-HOANG¹, (Student Member, IEEE),
DUC-HUNG LE¹, (Member, IEEE), **AKIRA TSUKAMOTO**²,
KUNIYASU SUZAKI^{2,3}, AND **CONG-KHA PHAM**¹, (Member, IEEE)

¹Department of Information and Network Engineering, The University of Electro-Communications (UEC), Tokyo 182-8585, Japan

²National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 135-0064, Japan

³Technology Research Association of Secure IoT Edge Application based on RISC-V Open Architecture (TRASIO), Tokyo 101-0022, Japan

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

This work was supported by the project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

ABSTRACT The Trusted Execution Environment (TEE) offers a software platform for secure applications. The TEE offers a memory isolation scheme and software authentication from a high privilege mode. The procedure uses different algorithms such as hashes and signatures, to authenticate the application to secure. Although the TEE hardware has been defined for memory isolation, the security algorithms often are executed using software implementations. In this paper, a RISC-V system compatible with TEEs featuring security algorithm accelerators is presented. The hardware accelerators are the SHA-3 hash and the Ed25519 elliptic curve algorithms. TileLink is used for the communications between the processor and the register of the accelerators. For the TEE boot, the software procedures are switched with the accelerated counterpart. Comparing to the software approach, a 2.5-decade increment is observed in the throughput of the signature procedure using the SHA-3 acceleration for big chunks of data. The Ed25519 performs 90% better compared to the software counterpart in execution times.

INDEX TERMS TEE, SHA-3, Ed25519, RISC-V.

I. INTRODUCTION

A Trusted Execution Environment (TEE) prevents unauthenticated code from running by using hashing, certificate signing, and cryptography. A clear example of these environments is the TEE boot, which authenticates a boot-loader by using manufacturer-given keys to sign the payload from an untrusted device such as external flash memories. The security data is often protected over address regions by scaling the privilege level in a processor. In the RISC-V Instruction Set Architecture (ISA) specification, a machine mode runs first when the processor is in a reset machine state. The processor then escalates to the user mode, protecting the access of previously configured memory regions using the Physical Memory Protection (PMP) scheme [1]. In ARM, exists a security extension named TrustZone, which can be configured to protect memory addresses through a barrier [2].

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam ¹.

The implementation of different TEEs can be described in various aspects. The execution environment is only generated once in ARM TrustZone, and Intel SGX and Keystone can be generated dynamically, making them Enclaves. TrustZone and Keystone have two different memory models named Normal and Secure World, which can be configured at startup. The trusted boot depends on the implementation of the system and the initial ROM, usually being authenticated by external cryptographic modules. Intel SGX is the only CPU with a clear point of trust, which in comparison, can be omitted in ARM TrustZone. Keystone implements the Sanctum method but is still in the early implementation stages [3]. Beside this status, the Keystone enclave offers novelty due to be implemented in RISC-V processors, the security algorithms are quantum-resistant, and there is open-source which allows further modifications in the root of trust. Some studies have been demonstrated the security of hardware-based TEEs like ARM TrustZone and Intel SGX. Ning *et al.* study the hardware-assisted TEEs of the ARM TrustZone and the Intel SGX, showing that the deployment can improve the

security with low-performance overhead [4]. Mukhtar *et al.* present a comparative analysis between Intel SGX and ARM TrustZone, indicating a failure to protect data against software side-channel attacks [5]. Benhani *et al.* evaluate the security of ARM TrustZone in Field-Programmable Gate Array (FPGA) System-on-Chip (SoC) inserting malicious circuits in the buses [6].

The use of TEE is increasing in different applications where utilize the memory isolation and authentication of the framework. Abdi *et al.* utilize TEEs to guarantee a baseline physical safety against cyber-attacks who can compromise the controller unit [7]. Guan *et al.* shield applications over compromised operative systems and physical attacks using a lightweight run-time system based on ARM TrustZone [8]. Staffa *et al.* present a study of the security treats over humanoid robots and the first hardware-based solution using TEEs [9]. Fan *et al.* implement TEEs on mobile devices [10]. IBBE-SGX construct an encryption scheme for group access control over the cloud secured by TEEs [11]. Jang *et al.* construct a TEE defense framework using a partially privileged process that can communicate with the hypervisor and TrustZone without depending on the kernel [12]. SPEED proposes a secure and generic deduplication system in Intel SGX, improving performance by reusing computation results by identifying redundant computation [13]. HyperMI presents virtual machine protection, featuring security against compromised hypervisors by isolating guests in a secure execution environment [14]. MicroTEE designs a TEE on a microkernel software architecture with the necessary services for the application layer [15]. Ladjel *et al.* evaluate the use of TEE-based computing for personal data in a large number of participants [16]. Gollamudi *et al.* present a core calculus for secure decentralized distributed applications using standard cryptographic mechanisms and the TEE platforms [17]. DER-TEE presents a smart inverter using a TEE-based architecture [18]. Finally, TEE-Perf offers performance measurement tools for TEEs, injecting profiling code, and tracing the program execution [19].

The TEE can be performed in several ways, and each one composes a framework that communicates the limited with the trusted privileges. An *enclave* framework uses memory protection in user mode to store sensitive data in selected areas of the address space. Sanctum is an enclave that isolates on software level over memory pages utilizing a memory translation modification in hardware [3]. Mi6 protects cache coherence attacks by using speculative out-of-order multi-core processors and sanctums [20]. Keystone is a framework for RISC-V processors that takes advantage of the physical memory protection standard to authenticate the execution of programs in a safe environment [21]. TIMBER-V is also an isolation scheme for RISC-V processors that provides security with low overhead [22].

TEEs offer a strategy in software and hardware to offer isolation and trusted execution for applications. The actual encryption and security algorithms are offered by

external cores. In the case of Intel SGX and ARM TrustZone, the system can implement a Trusted Platform Module (TPM). This TPM implements standard security algorithms and crypto-primitives like SHA-1, SHA-256, RSA, ECC, AES-128, HMAC, and MGF1 for trusted computing [23], [24]. The TPM can be implemented in software or hardware, as long as the access to the TPM is isolated by the trusted environment. Most of the new computer assets with an Intel architecture processor contains an isolated TPM in hardware, but ARM architectures rarely contain a hardware implementation. In Keystone and other RISC-V trusted environments, such specification is not defined clearly. Novel papers have specified a group of cryptographic primitives and endorsement algorithms. In the case of Keystone, SHA-3 and Ed25519 are necessary to implement a secure environment [21]. TIMBER-V performs authentication with SHA256 and HMAC algorithms [22].

Although hardware architectures have been proposed for the development of the TEEs in RISC-V, there is a reduced number of applications where the framework combines hardware cryptographic accelerators with the environment. In this paper, we present a RISC-V system for TEEs that includes hardware acceleration on the SHA-3 secure hash algorithm and the Ed25519 elliptic curve algorithm. The SHA-3 and Ed25519 were chosen because of their reasonably recent specification publications, the low overhead in their execution, and they are part of the current Keystone implementation [21]. Currently, SHA-3 is the latest secure hash standard, and its efficiency has been proven to be quantum-resistant due to the sponge function in Keccak-1600 [25]. On the other hand, the Ed25519, according to its authors [26], has the security level equivalent to that of a 3000-bit Rivest-Shamir-Adleman (RSA) key-strength, while its implementation cost is much cheaper than an equivalent RSA implementation. To conclude, combining the SHA-3 and Ed25519 with the customizable open-source processor of RISC-V makes a significant improvement for the TEE framework, especially for the TEE boot procedure. For the processor implementation, the Rocket chip generator is used to integrate the system and to bind the accelerators on the memory map [27]. The SHA-3 and Ed25519 hardware accelerators are wrapped using CHISEL Hardware Description Language (HDL) [28]. The TileLink buses are used for the processor to register communications [29]. The SHA-3 accelerator works by pushing 64-bit chunks of data over the input register. The Ed25519 accelerator is divided into two parts: the base-point multiplier, and the signature calculator. The base-point multiplier calculates a scalar multiplication over the base-point of the curve25519, useful for the key-pair generation and part of the signature calculation. The signature procedure performs the signature equation of the Ed25519, wrapping it also automatically over the large prime [26]. These accelerators are used in the TEE boot procedure, which utilizing the Keystone framework as the base environment [21]. The software calculations of the Keystone are implemented using hardware accelerators.

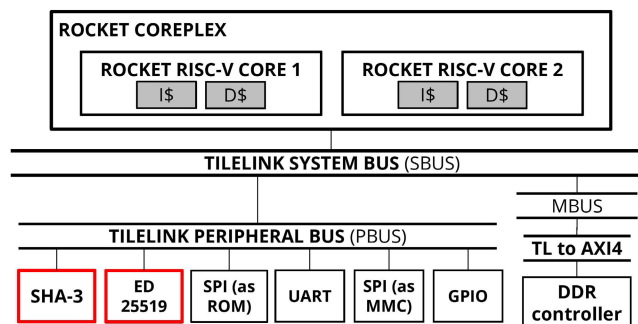


FIGURE 1. Hardware architecture of the SoC TEE.

The remainder of this paper is organized as follows. Section II describes the proposed hardware system. Section III depicts the software platform that manages the TEE boot using the hardware accelerators. Section IV shows the experimental results. Furthermore, Section V concludes the paper.

II. HARDWARE IMPLEMENTATION

The system for the TEE features RISC-V processors implementing the RV64IMAFDC instruction sets [1]. The RV64IMAFDC means 64-bit RISC-V ISA with Integer, Multiplication, Atomic, Floating-point, Double floating-point, and Compress extensions. The Rocket chip generator is used as the hardware platform to integrate the security accelerators [27]. The generator and all the security cores utilize the Chisel hardware description language to generate an intermediate Register Transfer Level (RTL) representation [28]. This RTL representation, named FIRRTL, is converted to Verilog to a future assembly in FPGA and Very-Large Scale Integrated-circuit (VLSI) flows [30], [31]. This generator is a part of a chip platform creation named Chipyard where includes all sorts of Chisel-based utilities and hardware for different hardware assemblies [32]. The generator also outputs some file artifacts like torture scripts for processor testing and a device tree file for operative systems to detect the hardware attached to the system. All the additional artifacts are generated according to the configurations passed to the generator. All the configuration matches the requirements for the Keystone TEE to run a security manager from the boot sequence [21].

Figure 1 presents the computer architecture of the security system. This system is compatible with the software implementation of the Keystone enclave [33]. Two RISC-V Rocket cores are implemented in the coreplex generator. Each core integrates a 2KB instruction cache and a shared Layer-1 data cache of 2KB. The coreplex is connected to the system bus (SBUS) using the TileLink protocol [29]. The generator also creates the peripheral bus (PBUS) and the memory bus (MBUS), interconnecting all the bus nodes together. The address space for each node is set according to the configurations of each slave connected to any inter-connector. The system contains several peripherals attached to the PBUS such as GPIO, MMC controller, UART, and ROM. The data

input and output are memory-mapped using register routers, converting writes and reads from TileLink to register controls through the Rocket generator. The new security cores are located in the PBUS, where data and triggers are pushed to several memory locations. The system also supports 1GB RAM using a DDR controller, intended to offer an execution memory space for an operative system. This controller is driven by an AXI4 bus, converted from the memory-cached MBUS [34]. The overall system and the accelerators, along with the generator integration, are included in a single repository [35].

The first implemented security accelerator for the Key-stone boot sequence is an SHA-3 hash calculator. Several hardware architectures have been proposed to offer an SHA-3 acceleration. Eissa *et al.* [36] proposed an SHA-3 instruction set for the RISC-V ISA by separating each operation in the keccak algorithm into different instructions. Ahmed and Farag [37] offered a design with a configurable keccak-f function that optimized for Hash Message Authentication Code (HMAC) and Pseudo-Random Number Generator (PRNG). Ioannou *et al.* [38] presented an FPGA implementation of SHA-3 that capable of handling multiple messages. Figure 2 shows the peripheral architecture of the SHA-3 accelerator wrapped to the TileLink bus. This accelerator contains a padding module and a Keccak-1600 round calculator [25]. The padding module retrieves 64-bit data from the register-router, then is pushed through the 576-bit buffer using a shifter. When the buffer is full, the accelerator performs a round calculation. A constant counter keeps track of the number of rounds and constant non-linearity of the iota phase of the Keccak round. The first round is calculated from the first 64-bit data push through the padding module. Every round state is stored in an 1600-bit status register. When the final data is pushed through the padding module, the round calculation performs the final rounds in the status registers, and then the first 512-bit word can be used for the hash output of the calculation. The usage of this module consists of pushing 64-bit data stream segments through the SHA-3 accelerator. The final stream that does not fit in a 64-bit sub-stream needs to be pushed with the help of the final size register, which is ignored for intermediate streams.

The next accelerator is presented in Figure 3. This architecture contains the base point multiplier, which performs the operation $P = sB$ in the curve25519 elliptic curve [26]. This multiplier is useful for generating the public and private keys and providing part of the calculations in the signature and verification of the Ed25519 algorithms in the Keystone TEE. The base point multiplier inputs the data through a memory-mapped RAM controller with write-only capabilities, where SHA-3 hashed private key is stored. The Ed25519 multiplier extracts each one of the bits from memory for the base point multiplication. Each one of the bits passes through a microcode finite-state machine, which triggers different states according to the initialization, 0- and 1-bit calculation states from the extraction, rounding logic, and final calculations. This finite-state machine provides

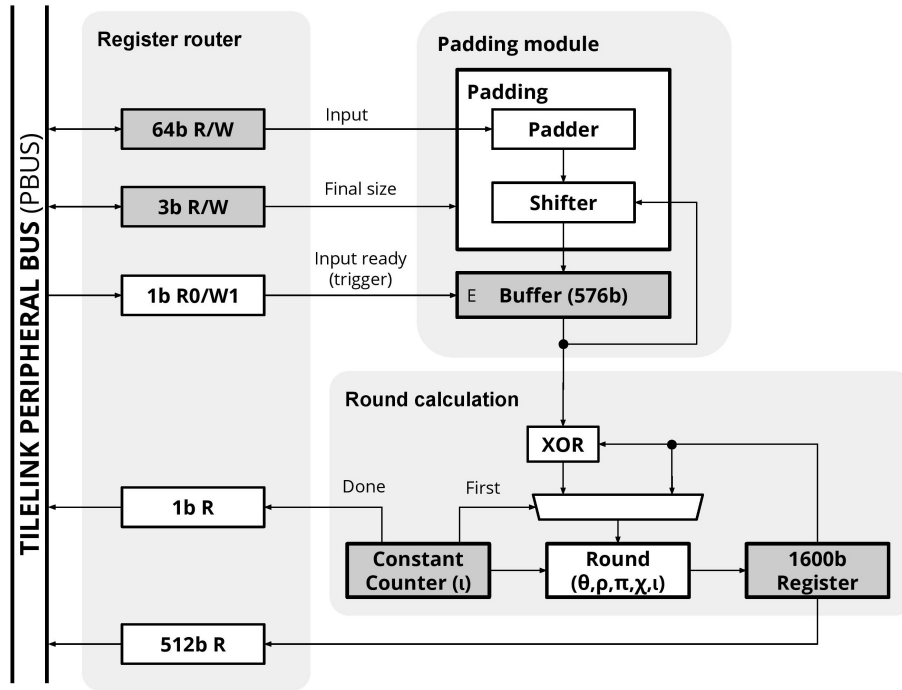


FIGURE 2. SHA-3 hashing datapath based on the algorithm specification in [25].

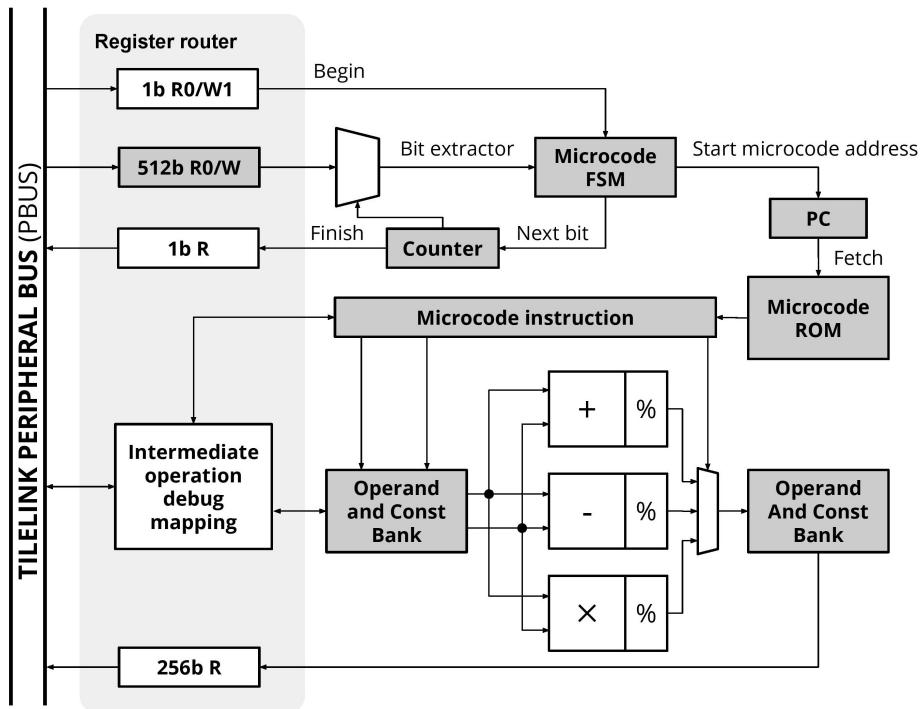


FIGURE 3. Ed25519 base point multiplier datapath based on the formula in [26].

the starting point to a microcode ROM for executing arbitrary instructions. These instructions are decoded and executed by a microprocessor containing the necessary modules for performing the 512-bit operations for the base point

calculation. The microprocessor is composed of an adder, a subtract module, and a multiplier. All the calculation modules are provided with a simple module calculator, useful for wrapping the value to the $2^{252} - 19$ prime number, necessary

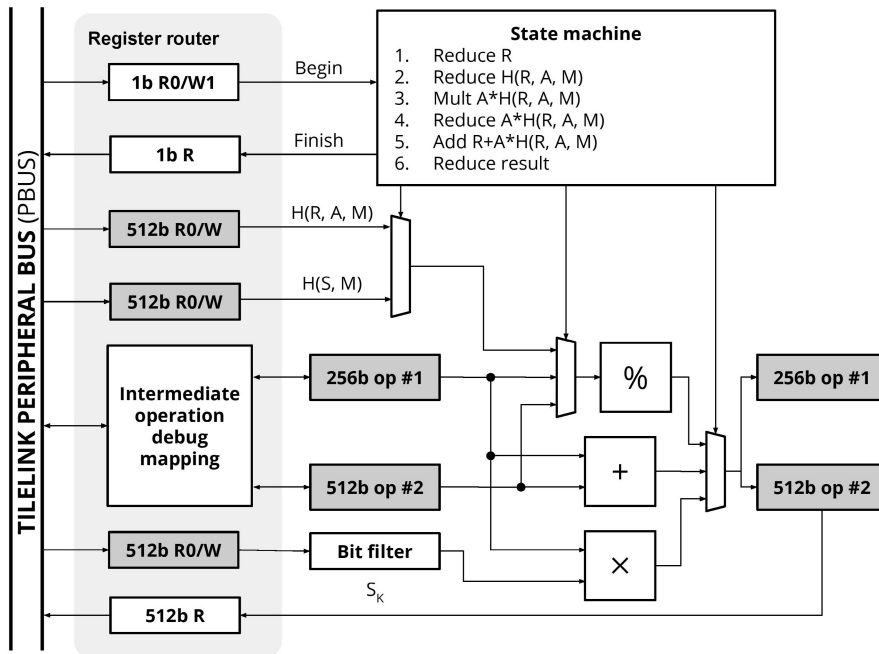


FIGURE 4. Ed25519 half signature datapath based on the formula in [26].

for this algorithm to perform the operations. The results for any operation can be stored in a 2KB memory bank, which acts as a register file, or the final output memory-mapped memory for the final output. The memory bank also contains useful constants mapped to specific addresses. The microcode contains instructions for performing vector-based operations according to the curve25519 definition.

The last accelerator is an Ed25519 signing datapath which performs the result of the operation $S = r + H(R, A, M) \times s$, which is half of the signature for any message M [26]. The other half of the signature can be calculated using the base point multiplier previously described. It is also noted that the number in each operation of the Ed25519's signing datapath is wrapped into the large prime l . This signature is used in the Keystone bootloader for signing the bootloader software to prevent tampering. Figure 4 shows the datapath for the signature calculation. The main three inputs of the signature are the hashed message with the public key and the half signature $H(R, A, M)$, the hashed message with the private key $H(S, M)$, and the public key A . This information is taken for the calculation of the other half of the signature triggering a begin enable from a memory-mapped register. The overall operation is directed by a simple state machine, which performs each one of the necessary operations of the signature equation. When the state machine is ready, the enable signal triggers further states for calculating each one of the steps of the equation. The datapath is composed by a 256-bit adder and multiplier, and a module calculator for wrapping value through the large prime l . The logic contains a 256-bit register for holding the sum results and a 512-bit register for the multiplication results. The state machine first reduces R from

the $H(S, M)$ value and stores it inside the 256-bit register. The reduction of $H(R, A, M)$ and subsequent multiplications and reductions with A are stored in the 512-bit register. The final sum and reduction are calculated from both the registers and stored in the final 512-bit register output.

III. TEE BOOT PROCEDURE

The TEE boot process is based on the Keystone platform [21]. The platform creates public and private keys from the keys of the manufacturer. This platform uses these keys to generate a signature of the bootloader to authenticate. The Keystone procedure is stored in a boot ROM on a fixed memory range where the multiple processors point the reset vectors, often named the Zero-State BootLoader (ZSBL). After the ZSBL stage, another bootloader usually utilized called the First-Stage BootLoader (FSBL). Then after the FSBL stage, the program to boot is a Linux bootloader like the Berkeley BootLoader (BBL) or OpenSBI, together with the Linux kernel and the initial ram filesystem. This program can be stored in any external media as a payload to authenticate. In this case, the payload is located in an external SD-card driven by an SPI controller.

Figure 5 describes the TEE boot process using the previous hardware architecture. First, the ZSBL in boot ROM locates and copies the FSBL from the external media (SD-card) to the main memory. Then, it jumps to the FSBL and executes there. After that, the FSBL locates and copies the BBL from the SD-card to the main memory. It also creates the Secure Monitor (SM) in the memory. The SM then extracts the initial seed for the key pair by hashing the copied BBL payload. The hash calculation is performed via the memory-mapped

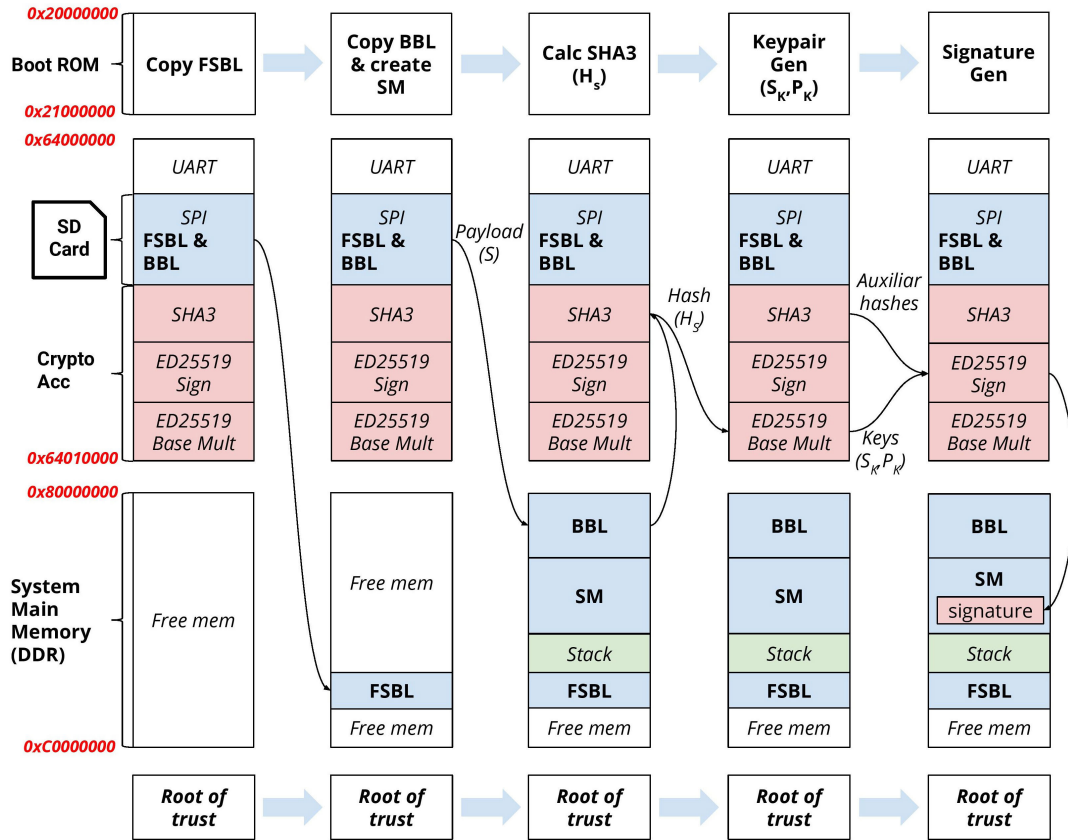


FIGURE 5. The TEE boot and root of trust procedure.

SHA-3 previously described in Figure 2 by pushing 64-bit chunks of data through the input register. The result of the hash is pushed to the Ed25519 base point multiplier for generating the public and secret pair of keys. With the help of the SHA-3 accelerator and the newly created keys, the SM performs the signature of the BBL payload then stores the result in a secure memory address for further use. At this point, the Linux kernel can be boot by executing the BBL. Finally, after booted, the BBL authentication can be done by the attestation function provided by the SM via signature verification.

As described before, most of the calculations for hashing and key-pair generation are done by pushing the data to certain data registers. The peripheral hardware reads the data from the registers and performs the desired output. The signature procedure follows a different way to accelerate the result. According to Figure 4, the signature receives calculated parts of the signature equation [26]. The algorithm 1 presents the procedure in software to calculate a full signature from the original data using the signature accelerator. This procedure uses the hardware SHA-3 and the Ed25519 base-point multiplier to obtain the two halves of the signature P and Q . The message and the secret key most-significant bits are first hashed using the hardware-accelerated SHA-3 and stored in r . To calculate R , the previous value is passed

Algorithm 1 Ed25519 Signature in Hardware Using SHA-3 and the Base Point Multiplier

```

1: procedure HW Ed25519sign(message, Pk, Sk)
2:   r ← HW SHA3(Sk[high], message)
3:   R ← HW Ed25519mult(r)           ▷ R = rB
4:   RAM ← HW SHA3(R, Pk, message)
5:   (In HW) H(S, M) ← r
6:   (In HW) H(R, A, M) ← RAM
7:   (In HW) SK ← SK
8:   (In HW) TriggerAndWait()
9:   return [P, Q]
10: end procedure

```

to the hardware Ed25519 base point multiplier. Again, the SHA-3 is used with the message, the public key, and the result of R . The next steps involve pushing the previously calculated data into the registers. The hardware procedure performs the operations to calculate the P and Q halves of the message signature.

IV. RESULTS

The proposed system is implemented in the Altera DE4 Development and Education board with the FPGA chip of Stratix-IV GX EP4SGX230. Table 1 shows the results

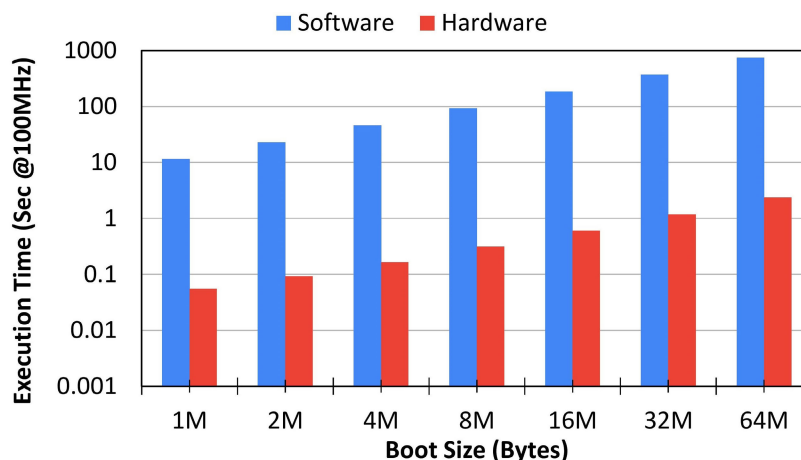


FIGURE 6. Comparison between software and hardware implementations of the TEE boot including SHA-3 and Ed25519 calculations using the Keystone bootloader.

TABLE 1. Synthesis results for a Stratix-IV GX Altera FPGA.

	SHA-3	Ed25519 mult	Ed25519 sign	Rocket Tiles
ALUTs	8108	2737	3969	24332
FFs	2790	4778	4617	15325
RAM Bits	0	65536	0	17680
DSP	0	48	0	32
Total	10898	73099	8586	57369
Logic (%)	3.4	2.3	2.6	12.4
RAM (%)	0.0	3.8	0.0	1.0
DSP (%)	0.0	3.7	10.0	2.4

of the implementation. Most of the logic utilization in the FPGA is occupied by the Rocket tiles, which occupies 12.4%, followed by the SHA-3 accelerator with 3.4%. The logic utilization in the Ed25519 signature and base-point multiplier are roughly the same around 2.3% and 2.6%. Although the logic increases between the Ed25519 accelerators, there is a significant increase in the usage of Digital Signal Processors (DSP) from 3.7% to 10% caused by bigger calculation units in the rounding machine. The Rocket tiles contain floating-point logic, integer multipliers, and dividers that synthesizes 2.4% of the DSP resources. The Ed25519 multiplier requires 8KB memory for the memory register bank for the temporal calculations, which occupies 3.8% of the total RAM utilization. The caches for the Rocket tiles contain 4KB of RAM, utilizing 1% of the RAM resources.

The FSBL was tested on this system with pure-software and hardware-accelerated algorithms for comparison. The SHA-3 and the Ed25519 hardware previously explained are used in the hardware-accelerated version, pushing the data of the payload for hashing, keypair generation, and signing. We run this environment with several payload sizes to measure time on authenticating the bootloader and the Linux kernel. Figure 6 presents the overall execution time for the Keystone bootloader to perform the root of trust in the TEE boot for several payload sizes. The payload includes

TABLE 2. Execution results for the Ed25519 key and signature process.

2MB bootloader	Software	HW SHA-3 SW Ed25519	HW SHA-3 HW Ed25519
Ed25519 genkey (ms)	109.5	93.4	4.6
Ed25519 sign (ms)	231019	82.6	4.7

the Linux bootloader and the Linux kernel, along with the initial filesystem. This payload was increased on the initial filesystem by putting random data on a file in several sizes. For both software and hardware implementations, the time increments exponentially. For any stream size, the execution time on the hardware-accelerated implementation of the Keystone framework decrements 2.5 decades compared to pure-software.

The algorithm presented in Figure 5 to authenticate the payload mostly uses the SHA-3 hashing, leaving the Ed25519 key-pair and signing as a new calculation process. Table 2 is presented to offer a broader perspective on the individual calculation execution of each one of the accelerators over a 2MB bootloader. A software-based implementation of both SHA-3 and Ed25519 presents an exponential increase of time compared to a hardware-only SHA-3 in the signature procedure, as the payload needs to be hashed twice. A hardware solution for the SHA-3 does not impact the execution time profoundly in the key-pair generation, as the hash to calculate is performed over a 256-bit stream. For key-pair calculation and signature process, there is a decrease of the execution time around 90% from the software to the hardware Ed25519 implementations, both with hardware SHA-3 hashing.

V. CONCLUSION

In this paper, we presented a system for TEEs featuring SHA-3 and Ed25519 accelerators. The system integrates two RISC-V cores with RV64IMAFDC ISA extensions using the

Rocket chip generator, including memory protection for the Keystone framework. The SHA-3 accelerator hashes data using a 64-bit register as input. The Ed25519 is divided into two main parts named the base-point multiplier for key-pair generation, and the signature accelerator. The software is composed of a root of trust to authenticate a Linux bootloader using Keystone for TEE boot. This software utilizes the accelerators by pushing the data over memory-mapped registers and triggers to obtain calculation results for the authentication signature. The system implementation is built in a high-end Altera Stratix-IV FPGA. The Rocket tiles occupy 12.4% of the total logic in the system. The included crypto-cores occupy a total of 8.3% combined in logic, but the Ed25519 utilizes up to 13.7% of the total FPGA DSPs to calculate multiplications and rounding. The total RAM used in the whole system is 4.7% distributed in the 4KB of cache included in a Rocket tile, and the 8KB of RAM used in the Ed25519 base-point multiplier to store temporal calculations. The calculation process for any bootloader payload to authenticate is reduced 2.5-decade milliseconds comparing the software and hardware implementations of the TEE boot. The SHA-3 acceleration decreases the execution time for message signature significantly, but not too much for the key-pair generation. The time of the key-pair and signature procedures are decreased from 90% using the Ed25519 acceleration.

ACKNOWLEDGMENT

This article is based on the results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

REFERENCES

- [1] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi a, "The RISC-V instruction set manual, volume I: User-level ISA, version 2.0," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html>
- [2] *TrustZone Technology for the ARMv8-M Architecture*, ARM, Cambridge, U.K., 2017. [Online]. Available: <https://developer.arm.com/docs/100690/0200>
- [3] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. 25th USENIX Secur. Symp.*, Aug. 2016, pp. 857–874.
- [4] Z. Ning, J. Liao, F. Zhang, and W. Shi, "Preliminary study of trusted execution environments on heterogeneous edge platforms," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 421–426.
- [5] M. A. Mukhtar, M. K. Bhatti, and G. Gogniat, "Architectures for security: A comparative analysis of hardware security features in intel SGX and ARM TrustZone," in *Proc. 2nd Int. Conf. Commun., Comput. Digit. Syst. (C-CODE)*, Mar. 2019, pp. 299–304.
- [6] 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.
- [7] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Preserving physical safety under cyber attacks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6285–6300, Aug. 2019.
- [8] L. Guan, C. Cao, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "Building a trustworthy execution environment to defeat exploits from both cyber space and physical space for ARM," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 3, pp. 438–453, May 2019.
- [9] M. Staffa, G. Mazzeo, and L. Sgaglione, "Hardening ROS via hardware-assisted trusted execution environment," in *Proc. 27th IEEE Int. Symp. Robot Hum. Interact. Commun. (RO-MAN)*, Aug. 2018, pp. 491–494.
- [10] Y. Fan, S. Liu, G. Tan, X. Lin, G. Zhao, and J. Bai, "One secure access scheme based on trusted execution environment," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Communications/12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2018, pp. 16–21.
- [11] S. Conti, R. Pires, S. Vaucher, M. Pasin, P. Felber, and L. R evill ere, "IBBE-SGX: Cryptographic group access control using trusted execution environments," in *Proc. Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2018, pp. 207–218.
- [12] J. Jang and B. B. Kang, "Retrofitting the partially privileged mode for TEE communication channel protection," *IEEE Trans. Dependable Secure Comput.*, Aug. 2018, p. 1.
- [13] H. Cui, H. Duan, Z. Qin, C. Wang, and Y. Zhou, "SPEED: Accelerating enclave applications via secure deduplication," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 1072–1082.
- [14] K. Lin, W. Liu, K. Zhang, and B. Tu, "HyperMI: A privilege-level VM protection approach against compromised hypervisor," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Communications/13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 58–65.
- [15] D. Ji, Q. Zhang, S. Zhao, Z. Shi, and Y. Guan, "MicroTEE: Designing TEE OS based on the microkernel architecture," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Communications/13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 26–33.
- [16] R. Ladjel, N. Anciaux, P. Pucheral, and G. Scerri, "Trustworthy distributed computations on personal data using trusted execution environments," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 381–388.
- [17] A. Gollamudi, S. Chong, and O. Arden, "Information flow control for distributed trusted execution environments," in *Proc. IEEE 32nd Comput. Secur. Found. Symp. (CSF)*, Jun. 2019, p. 304.
- [18] D. J. Sebastian, U. Agrawal, A. Tamimi, and A. Hahn, "DER-TEE: Secure distributed energy resource operations through trusted execution environments," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6476–6486, Aug. 2019.
- [19] M. Bailieu, D. Dragoti, P. Bhatotia, and C. Fetzer, "TEE-perf: A profiler for trusted execution environments," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 414–421.
- [20] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, Arvind, and S. Devadas, "MI6: Secure enclaves in a speculative Out-of-Order processor," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 42–56.
- [21] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, and K. Asanovi a, "Keystone: An open framework for architecting TEEs," 2019, *arXiv:1907.10119*. [Online]. Available: <http://arxiv.org/abs/1907.10119>
- [22] S. Weiser, M. Werner, F. Brassler, M. Malenko, S. Mangard, and A.-R. Sadeghi, "TIMBER-V: Tag-isolated memory bringing fine-grained enclaves to RISC-V," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2019, pp. 1–15.
- [23] *Trusted Platform Module 1.2 Main Specification*, Trusted Comput. Group, Beaverton, IR, USA, 2011. [Online]. Available: <https://trustedcomputinggroup.org/resource/tpm-main-specification/>
- [24] *Trusted Platform Module 2.0 Library Specification*, Trusted Comput. Group, Beaverton, IR, USA, 2016. [Online]. Available: <https://trustedcomputinggroup.org/resource/tpm-library-specification/>
- [25] J. M. Dworkin. (Aug. 2015). *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. [Online]. Available: <https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions>
- [26] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security Signatures," *J. Cryptograph. Eng.*, vol. 2, no. 2, pp. 77–89, Sep. 2012.
- [27] RISC-V Foundation. (2019). *Rocket Chip Generator*. [Online]. Available: <https://github.com/chipsalliance/rocket-chip>
- [28] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanovi a, "Chisel: Constructing hardware in a scala embedded language," in *Proc. 49th Annu. Design Autom. Conf.*, Jun. 2012, pp. 1212–1221.
- [29] SiFive. (Aug. 2019). *SiFive TileLink Specication*. [Online]. Available: <https://www.sifive.com/documentation/tilelink/tilelink-spec/>
- [30] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach, "Reusability is FIR-RTL ground: Hardware construction languages, compiler frameworks, and transformations," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 209–216.

[31] P. S. Li, A. M. Izraelovitz, and J. Bachrach, "Specification for the FIRRTL Language," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-9, Feb. 2016. [Online]. Available: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html>

[32] University of California at Berkeley. (2020). *Chippyard: An Agile RISC-V SoC Design Framework With in-Order Cores, Out-of-Order Cores, Accelerators, and More*. [Online]. Available: <https://github.com/ucbar/chippyard>

[33] K. Enclave. (2020). *Keystone: An Open-Source Secure Enclave Framework for RISC-V Processors*. [Online]. Available: <https://github.com/keystone-enclave/keystone>

[34] ARM Company, "AMBA AXI and ACE protocol specification," ARM, Cambridge, U.K., Tech. Rep. ARM IHI 0022D, 2011. [Online]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf

[35] C. Duran, T.-T. Hoang, D.-T. Nguyen-Hoang, D.-H. Le, A. Tsukamoto, K. Susaki, and C.-K. Pham. (2020). *Keystone hardware*. [Online]. Available: <https://github.com/ckdur/keystone-hardware>

[36] A. S. Eissa, M. A. Elmohr, M. A. Saleh, K. E. Ahmed, and M. M. Farag, "SHA-3 instruction set extension for a 32-bit RISC processor architecture," in *Proc. IEEE 27th Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2016, pp. 233–234.

[37] K. E. Ahmed and M. M. Farag, "Hardware/software co-design of a dynamically configurable SHA-3 System-on-Chip (SoC)," in *Proc. IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2015, pp. 617–620.

[38] L. Ioannou, H. E. Michail, and A. G. Voyiatzis, "High performance pipelined FPGA implementation of the SHA-3 hash algorithm," in *Proc. 4th Medit. Conf. Embedded Comput. (MECO)*, Jun. 2015, pp. 68–71.



DUC-HUNG LE (Member, IEEE) received the B.Sc. degree in physics and the M.Sc. degree in electronic physics from the University of Science–Vietnam National University of Ho Chi Minh City, in 2001 and 2005, respectively, and the Ph.D. degree in advanced science and engineering from The University of Electro-Communications (UEC), Tokyo, Japan, in 2013. From November 2019 to March 2020, he was with the Department of Information and Network Engineering, The University of Electro-Communications (UEC), Tokyo. He is currently with the Faculty of Electronics and Telecommunications, University of Science - Vietnam National University of Ho Chi Minh City. His research interests include design of digital systems on FPGA and integrated circuits, low-power digital IC design, and digital signal processing.



AKIRA TSUKAMOTO received the M.S. degree in computer science from Columbia University in the City of New York. He works at the National Institute of Advanced Industrial Science and Technology (AIST). His main focusing area is software engineering on networks, operating systems, and system security, who enthusiastic on any kind of technical development and have worked on products based on Cell/B.E. and ARM.



TRONG-THUC HOANG (Graduate Student Member, IEEE) received the B.Sc. degree in electronics and telecommunications and the M.S. degree in microelectronics from the University of Science - Vietnam National University of Ho Chi Minh City, Vietnam, in 2012 and 2017, respectively. He is currently pursuing the Ph.D. degree in information and network engineering with The University of Electro-Communications (UEC), Tokyo, Japan, and also a Research Assistant with the National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan.



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



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



DUC-THINH NGUYEN-HOANG (Student Member, IEEE) received the B.Sc. degree in electronics and telecommunications from the University of Science–Vietnam National University of Ho Chi Minh City, Vietnam, in 2019. He is currently working as a Research Assistant at the Department of Information and Network Engineering, The University of Electro-Communications (UEC), Tokyo, Japan.



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

...