# Lightweight Secure-Boot Architecture for RISC-V System-on-Chip

Jawad Haj-Yahya[1,2] Ming Ming Wong[1] Vikramkumar Pudi[1,3] Shivam Bhasin[1] Anupam Chattopadhyay[1]
[1]Nanyang Technological University (NTU), Singapore
[2]Institute of Microelectronics, Agency for Science Technology and Research (ASTAR), Singapore
[3]Indian Institute of Technology (IIT), Tirupati, India

***Abstract*—Securing thousands of connected, resource-constrained computing devices is a major challenge nowadays. Adding to the challenge, third party service providers need regular access to the system. To ensure the integrity of the system and authenticity of the software vendor, secure boot is supported by several commercial processors. However, the existing solutions are either complex, or have been compromised by determined attackers. In this scenario, open-source secure computing architectures are poised to play an important role for designers and white hat attackers.**

**In this manuscript, we propose a lightweight hardware-based secure boot architecture. The architecture uses efficient implementation of Elliptic Curve Digital Signature Algorithm (ECDSA), Secure Hash Algorithm 3 (SHA3) hashing algorithm and Direct Memory Access (DMA). In addition, the architecture includes Key Management Unit, which incorporates an optimized Physical Unclonable Function (PUF) for providing keys to the security blocks of the System on Chip (SoC), among which, secure boot and remote attestation. We demonstrated the framework on RISC-V based SoC. Detailed analysis of performance and security for the platform is presented.**

***Keywords*—**Secure Boot, Physical unclonable Function, ECC, ECDSA, Hardware Security, IoT, Key Management,

## I. Introduction

Boot is one of the critical points during the lifetime of a secure system where system integrity can be compromised. Many attacks break the software while the device is powered down [1], performing an attack that, for example, replaces the OS kernel image with one that has been tampered with, making the system vulnerable. One of the principles applied in secure systems is the generation of a chain of trust for all software parts running from first boot loader up to trusted applications, established from a root of trust that cannot easily be tampered. This is known as a secure boot sequence. Thus, all secure devices, including laptops, desktops, smart-phones and IoT devices need to implement secure boot to assure system integrity.

The existing architectures of secure boot are either complex, or have been compromised by determined attackers [2], [3], [4]. The recent growth of open-source hardware [5], [6], [7] enables a transparent and thorough evaluation of the security protocols. Contributing to this effort, this work suggests a robust secure boot framework for the open source RISC-V processors. To the best of our knowledge, this is the first hardware based secure boot framework for RISC-V based SoCs.

**Our architecture** introduces a lightweight hardware-based secure boot. The architecture defines a centralized Code Authentication Unit (CAU) that is responsible for verifying the integrity of the chain of trust, starting from first boot loader until the application as depicted in Figure 1. CAU uses efficient implementation of Elliptic Curve Digital Signature Algorithm (ECDSA) and Secure Hash Algorithm 3 (SHA3) hashing algorithm. The use of ECDSA enabled easier integration of Physical Unclonable function (PUF) for asymmetric key generation and management, which is realized using the Key Management Unit (KMU). Moreover, a Direct Memory Access (DMA) is used by the CAU for fast code reading and authentication. We demonstrated the framework on RISC-V based System On Chip (SoC), whereas the high level architecture of the system is illustrated in Figure 2.

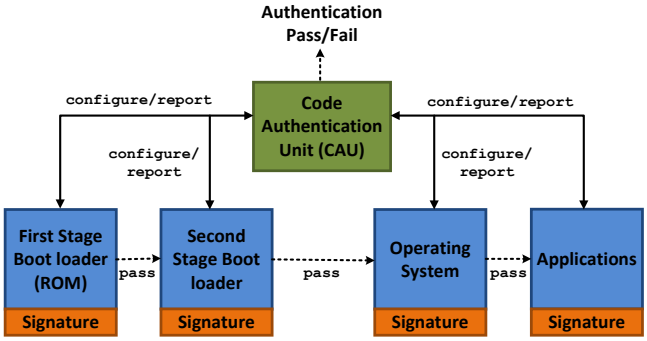


Fig. 1. Code Authentication Unit verifies the integrity of the chain of trust, starting from first boot loader up to the application.

This work have the following contribution:

- Present new lightweight architecture for secure boot that ensure system integrity and authenticity of the software vendor.
- A framework including key management unit using optimized Physical Unclonable Function (PUF) implementation.
- Integration of the Code Authentication Unit into modern RISC-V SoC.

The rest of this manuscript is organized as follows: Section II describes prior work and introduces secure boot concepts. Section III describes in high level out RISC-V based secure SoC. Our framework for secure boot is described in section IV. Security analysis of secure boot is carried out at V. Section VI present the framework evaluation. Section VII concludes.

## II. Background and Related Works

Several RISC-V based projects focus on enhancing the hardware security of the underlying SoC. Shakti-T [8] employs the concept of base and bounds to ensure that pointers access only valid memory regions. SMARTS [9] implemented

Memory Protection Unit (MPU) that was integrated into RISC-V SoC. The proposed framework provides integrity and confidentiality of the external memory (DRAM). Keystone [10] initiated an open-source project for building trusted execution environments (TEE) with secure hardware enclaves, based on the RISC-V architecture. Sanctum [11] introduced software based secure boot and remote attestation process. However, their work focus on software implementation of secure boot that have several drawbacks compared to hardware implementation, as explained in section V. This work presents a lightweight hardware based secure boot architecture for RISC-V SoC, and describes Key Management Unit that uses PUF.

### A. Secure Boot

The first boot protection mechanism was proposed by Arbaugh et al. in [12]. They describe a way to verify the integrity of a system by constructing a chain of integrity checks of every stage of the boot process. Every stage in the boot process has to verify the integrity of the next stage.

Secure boot is described in the Unified Extensible Firmware Interface (UEFI) specification since version 2.2 [13]. UEFI secure boot verifies the integrity of each stage of the boot process by computing a hash and comparing the result with a cryptographic signature. A key database of trustworthy public keys needs to be accessible during boot time so that the signature can be verified. In secure boot, if any integrity check fails, the boot will be aborted. If the boot process succeeds, the system is expected to be running in a trusted state. This definition of secure boot is widely accepted in the security community ([14] and references therein).

Additional terms and variants of secure boot that describe the integrity checks process, such as - *Trusted Boot*, *verified boot*, *authenticated boot*, *certified boot*, or *measured boot* - are also found in commercial products and some research articles, all with different connotations. For example, Intel processors support secure boot in two modes - measured and verified modes. For both modes microcode on the processor is the root of trust for the boot sequence [15]. In the measured mode a Trusted Platform Module (TPM) is responsible for storing and attesting to the measurements, while in verified mode each component is signed by the manufacturer and these signatures are verified prior to loading the component. Another variant of secure boot mechanism is where the root of trust is built on immutable hardware that is integrated inside a dedicated secure processor. For instance, Hardware Validated Boot (HVB) is an AMD-specific form of secure boot that roots the trust to Read Only Memory (ROM) [16]. The ROM validates the secure boot key, which will later be used to validate the larger processor firmware that is fetched from the system flash.

### B. Digital Signature Authentication

Digital signature based authentication is a well-known technique arising from public-key cryptography. It is used in most web browsers (for SSL) and email packages. All public-key cryptographic systems have their security based on certain mathematical problems that are difficult to solve. For example, RSA [17], has its security based on the difficulty of factoring integers. Similarly, Elliptic Curve Cryptography (ECC) [18] has its security based on the elliptic curve discrete logarithm problem (ECDLP). The most popular signature scheme which uses elliptic curves is called the Elliptic Curve Digital Signature Algorithm (ECDSA), the most popular encryption scheme is called the Elliptic Curve Integrated Encryption Scheme (ECIES) and the most popular key agreement method is called Elliptic Curve Diffie-Hellman (ECDH) [19].

NISTs standard for Digital Signatures [20] recommends using a prime field, $GF(p)$ , or a binary extension field $GF(2^m)$ for Elliptic Curves. Binary Extension Fields have the advantage that field additions can be performed by XOR operations, therefore no carry is involved. This leads to implementations that require lesser area and have better performance. While Standards for Efficient Cryptography Group (SECG) [21] has defined the Koblitz curve secp256k1 that is used by online cryptocurrency Bitcoin [22]. NIST curves are more widely used and has received more scrutiny than other SECG curves.

### C. Physical Unclonable Function (PUF)

Physical Unclonable Functions extract volatile secret keys from semiconductor manufacturing variation that only exist when the chip is powered on. The first documented use of PUF generated keys in a secure processor setting was in the AEGIS processor [23]. The PUF was used to generate a symmetric key shared with the client through a cryptographic protocol. Recently, PUFs are used as symmetric key generators in commercial products such as Xilinx Ultrascale Zynq FPGAs [24] and Intel/Altera [25] FPGAs. For Public Key Algorithms (PKA), PUF can be used to generate a random seed for asymmetric (public/private) key generator inside of a secure processor, such as [26].

## III. Overview of Secure RISC-V SoC

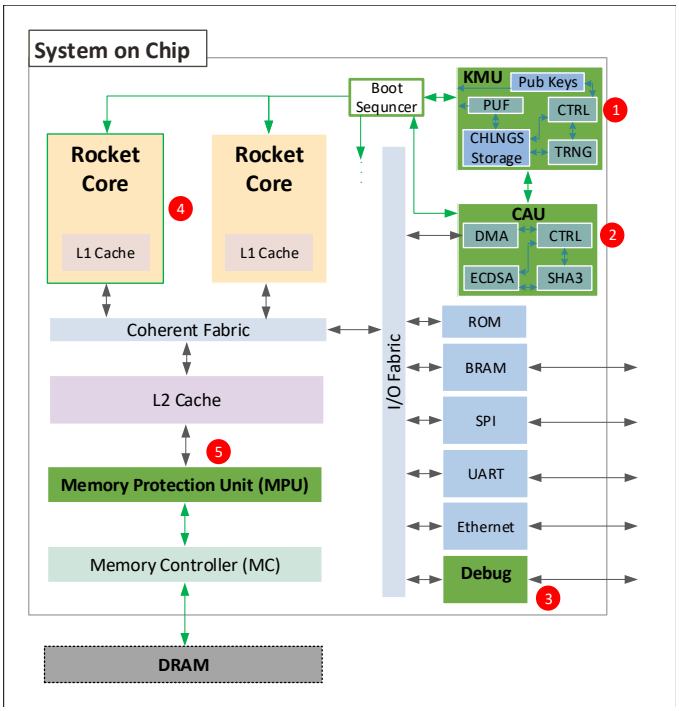


Fig. 2. Architecture of our secure SoC. Main security units are highlighted: (1) Key Management Unit, (2) Code Authentication Unit, (3) Secure Debug, (4) Trusted Execution Environment and (5) Memory Protection Unit.

The secure SoC presented in this study is built upon lowRISC [6] as the baseline processor. LowRISC is an open source processor which implements the open RISC-V Instruction Set Architecture (ISA) [27], [28]. The secure system developed in this project is equipped with security functionality, such as, secure boot, encryption and authentication of off-chip memory, key management and cryptographic accelerators as depicted in Figure 2.

Essentially, a secure SoC has to be robust against the attacks in both hardware and software. Therefore, the developed secure RISC-V SoC implements various features to protect against known attacks on the computing system hardware, and the software that is running on it. As illustrated in Figure 2, these features include:

- **Key Management Unit** 1 generate and distribute the keys to the various security blocks, such as Code Authentication Unit (CAU), Secure Debug and Memory Protection Unit.
- **Code Authentication Unit** 2 to protect against attacks such as: Image Hacking, Botnet Enrolling and Cold-Boot attack. **secure boot** is implemented using Code Authentication Unit (CAU) and Boot-Sequencer block.
- **Secure Debug** 3 to protect against various hardware threats such as Key Extraction, Illicit Debugging, Probing and Side Channel Attacks (SCA).
- **Trusted Execution Environment (TEE)** 4 guarantees isolated execution environment for the trusted application, this feature is essential for protecting against attacks such as: Software Exploitation, Privilege Escalation and Botnet Enrolling.
- **Trusted Off-Chip Memory** 5 is an essential feature that protects against Side-Channel Attacks (SCA), Probing and Key Extraction from main memory. In addition, it is used by the TEE to load and execute trusted applications, thereby protecting the code and data of the running applications.

This paper highlight two major designs, namely the Code Authentication Unit (CAU) and the Key Management Unit (KMU) which play the vital roles in providing *secure boot* in the secure RISC-V SoC.

## IV. Framework of Trusted and Secure Boot in RISC-V SoC

A feasibly secure RISC-V SoC shall establish a chain of trust at the early stage of boot sequence and needed be maintained during the OS and applications run. The chain starts with the first stage where the immutable bootloader is run from the read-only-memory (ROM). This bootloader cryptographically verifies the signature of the second stage bootloader in the chain and then, this subsequent bootloader again cryptographically verifies the signature of the next software image(s) and so forth. At the end of the chain, a trusted application is verified before it is running on the trusted execution environment (TEE) of the SoC.

### A. Bootloader (BBL) in RISC-V

The main bootloading procedure instructions for RISC-V is defined in the Berkeley Boot Loader (BBL), which is the actual bootloader for the SoC. BBL performed several functions such initializing all peripheral devices, setting up page table and virtual memory and loading the choice of kernel (Linux) into virtual memory and finally boot the kernel.

The entire boot flow is conducted in multiple stages. In **Stage 1,** the configured hardware RISC-V SoC contains a storage ROM loaded with bare metal boot (C program). This boot program is required to load the actual bootloader, BBL. Next, **Stage 2** involved copying/moving the BBL to DDR RAM. The size of BBL is in fact larger than on-chip boot ROM. Hence, in the recent development of RISC-V, there are two options available to facilitate this process,

- SD-Boot: The BBL is loaded from digital card storage connected externally to the RISC-V core.
- Etherboot: The bare metal boot loader advertises its presence on the network to incoming ARP queries.

With that, the loaded program will be an ELF file which contains BBL with the choice of kernel (such as Linux). In this second option, the ELF file (BBL+kernel) is segmented into UDP packets before sending over the network. Final **Stage 3** involved launching BBL in machine mode and sets up the machine mode trap vectors and proceed with execution of the kernel.

### B. Secure Boot as Chain of Trust

The boot process in RISC-V begins by executing the hard-coded instructions (termed as *ROMSlave* located at `uncore/src/main/scala/rom.scala`) which its sole purpose is to jump from the reset vector (termed as *ResetVector*) to the start of the BRAM.

Therefore, these hard-coded instructions are calculated based on *ResetVector* which is characterized in a configuration file (chisel code located in `src/main/scala/Configs.scala`). The initial reset vector is described in function *makeBootRom()* (chisel code located in `src/main/scala/LowRISCChip.scala`) which configured with instructions that boot should either starts from BRAM (for normal FPGA execution) or directly to DDR.

In order to achieve *secure boot* flow in RISC-V SoC, the proposed framework presented in this study establishes a chain of trust by enforcing *code authentication procedure* in between the bootloading stages. Such authentication is performed via the presented Code Authentication Unit (CAU) which integrates signature verification procedures.

### C. CAU in RISC-V SoC

A hardware component called Code Authentication Unit (CAU) cryptographically verifies the signature of each trusted code that will be running on the processor using Elliptic Curve Digital Signature Algorithm (ECDSA). CAU verifies each software code that is to be executed, starting from the first boot up to the applications. Such mechanism effectively prevents unauthorized or modified code from being executed in our secure SoC.

The recent RISC-V development employs re-configurable memory mapping (instead of fixed memory map), where the addresses are automatically generated upon the build

instructions for peripheral customization. This approach is more advantageous as this will ensure correct alignment of start addresses to prevent crossing address boundaries. The final memory map (termed as *ConfigString*) will be used and automatically linked to C-programs (as well as other hardware modules).

Therefore, the hard-coded instructions *ROMSlave* comprised of *jump* instruction, *ResetVector* and *ConfigString*. This piece of code will be signed and the signature is appended at end of the code line;

`rom.array() ++ p(ConfigString).toSeq ++ signature` (refer function in *makeBootRom()*).

The initial boot ROM is mapped to memory address $0x10000$ of which also marks the first data CAU module will begin to perform signature verification. The obtained value will be matched with the signature appended at the end of the code. Positive match result shall lead to the next boot stage, otherwise RISC-V Rocket tile (*RocketTile()*) shall be suspended.

### D. Application: Linux in RISC-V SoC

In this case study, We selected Linux version 4.2 (as the choice of kernel) to be run on secure RISC-V SoC. A bare metal boot program (*boot.c*) is compiled and preloaded in the BRAM which will copy (*boot.bin*, consisted of BBL with Linux appended as payload) from SD card and move to DDR. The process is also as depicted in Figure 3.

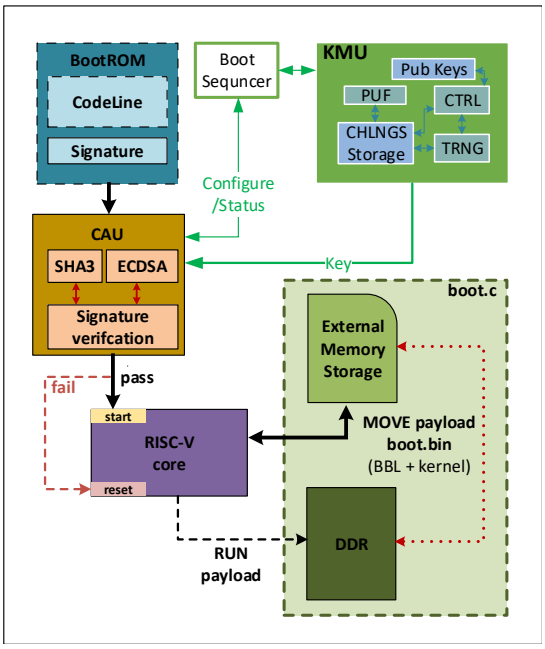


Fig. 3. Secure Boot Flow in RISC-V SoC

As mentioned previously, Linux is not booted directly, as it goes via Berkeley boot loader which sets up the machine mode (including handlers for unaligned fetches/stores and emulated instruction traps). Note that the boot ROM not only enforce jump instruction from *ResetVector* to BRAM location. It also serves another purpose which is to store the start address of peripherals for the use of Linux. This conveniently avoids the need for a BIOS to determine or to calculate these addresses.

Next, the boot proceeds with launching the Linux in supervisor mode and ensuring the start address is aligned with the device memory mapping (listed in boot ROM). Upon the execution of the kernel, the BBL continues running as hypervisor (underneath the kernel) to serve the peripheral interrupt request from the actual FPGA hardware.

### E. Boot Sequencer and Key Management Unit

Boot Sequencer and Key Management Unit (KMU) are shown in Figure 3. The Boot Sequencer is a Finite State Machine (FSM), that sequences the flow of the SoC's boot process. This hardware function is called every time a new stage of the boot is about to run. the Boot Sequencer communicates with the KMU to send the relevant key to the CAU. Later it asks the CAU to read the corresponding code from memory and authenticate it's signature using the ECDSA. The Boot Sequencer collects the authentication result (pass or fail) and based on it, decides if to continue with the boot sequence or to abort it. At the current implementation, the Boot Sequencer is a hardware state machine, while in future version it can be replaced by small micro-controller for the sake of better *programmability* and *patchability*.

The Key Management Unit (KMU) is responsible for the generation and distribution of symmetric and asymmetric encryption keys. Whereas, it consists of True Random Number Generator (TRNG), Physical Unclonable Function (PUF), Challenges Storage and One-Time Programmable (OTP) memory. The TRNG generates a random challenges for symmetric keys challenges, the challenges are stored with corresponding identification into a storage. To retrieve a specific key, the challenge is fed into the PUF and a response is outputted that is used as key. These symmetric keys are used at part of the security blocks inside the SoC, such as, Memory Protection Unit (memory encryption and integrity) and Secure Debug Unit. The Code Authentication Unit uses Asymmetric keys required by Public Key Algorithms. The public keys are fused into the Key Management Unit using OTP memory. Key Management Unit architecture details are not presented here due to space limitation, at section VI we discuss in more details the design of the PUF.

## V. Security Analysis

Several existing secure boot architectures have been compromised by determined attackers. For instance, AMD's Hardware Validated Boot (HVB) [16] architecture for secure boot was shown to have severe security vulnerabilities that enabling bypassing the secure boot flow. Whereas, CTS labs revealed a set of security vulnerabilities that affected AMD processors [4]. According to the CTS labs report, an attacker can exploit the set of vulnerabilities referred to as *MASTERKEY* and *FALLOUT* in the AMD secure processor. This can allow an attacker to install unsigned malicious software; with the highest possible privileged access; inside the AMD secure processor by bypassing the Hardware Validated Boot. Consequently, a malicious application can be used to leak encryption keys that are used by the AMD Secure Encrypted Virtualization (SEV)[1] protected Virtual Machines, thus compromising the confidentiality of the data in those Virtual Machines.

[1]SEV is a security feature that mainly addresses the high-privileged system software class of attacks by providing encrypted Virtual Machine isolation

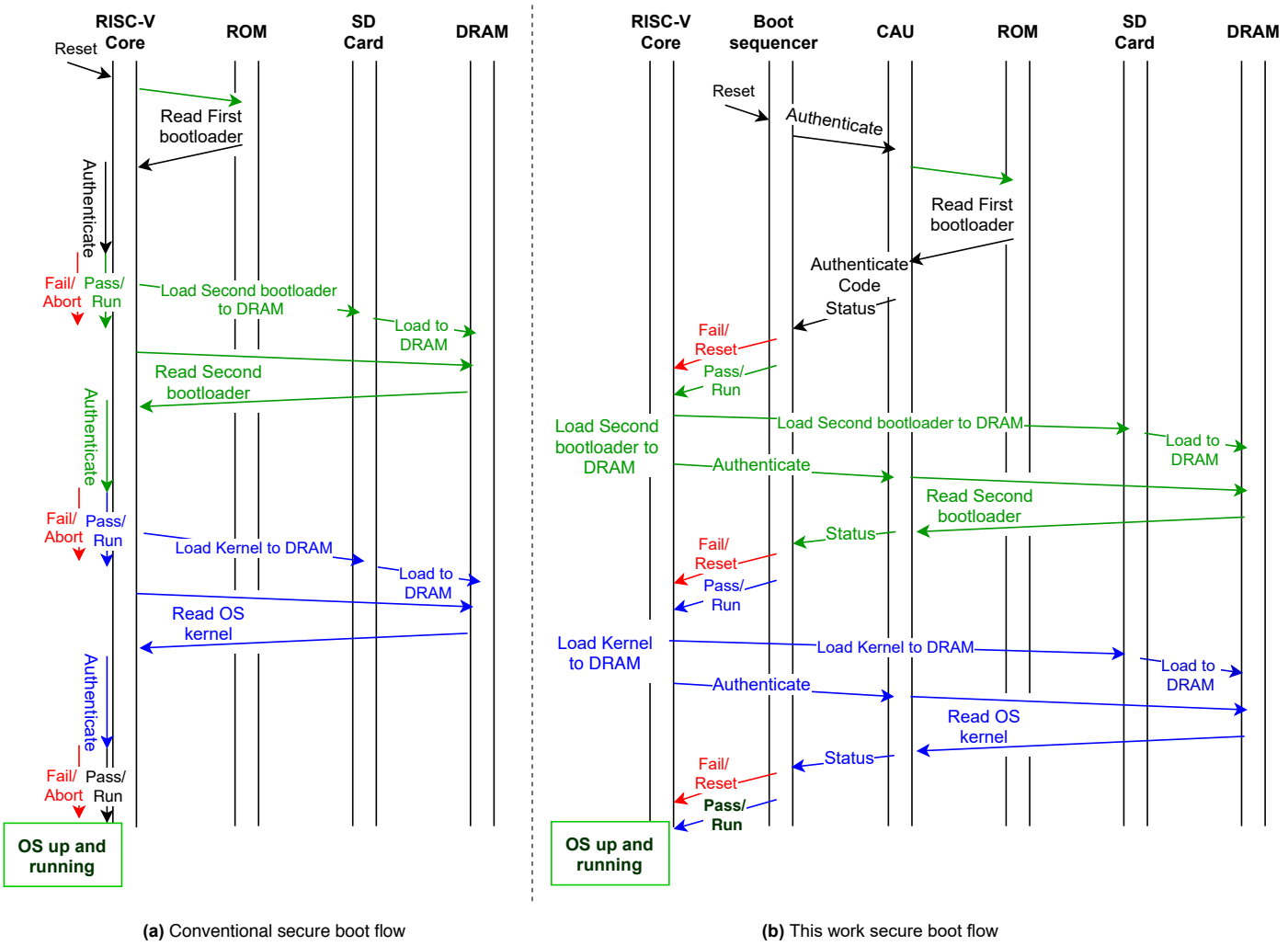


Fig. 4. Secure boot flow diagrams. Diagram (a) depicts a conventional flow that uses software based implementation (such as Sanctum[11]) of SHA3 and ECDSA that runs on the RISC-V processor. Diagram (b) shows this work's flow that uses hardware units to implement the secure boot.

Figure 4 describes secure boot flow diagrams, whereas diagram (a) depicts a conventional flow that uses software based implementation (such as Sanctum [11]) of SHA3 and ECDSA. The software runs on processor core , such as RISC-V core. On the other hand, diagram (b) shows this work's flow, whereas it uses hardware units (CAU and Boot Sequencer) to implement the secure boot. The main advantages for this implementations over conventional ones is as follows.

- The Hardware implementation has better performance and energy-efficiency over the software implementation as shown in section VI.
- Running the software implementation of SHA3 and ECDSA on the RISC-V processor assumes that the RISC-V is part of the trusted computing base (TCB). While including a relatively big design on the TCB might introduce security vulnerabilities similar to Meltdown [29] and Spectre [30] that cannot be easily detected due to the design size of the processor's core. On the other hand, our frameworks assumes that the CAU and the Boot Sequencer are trusted, and the RISC-V processor is not part of the TCB for the secure boot process.
- The secure boot in this work's framework is controlled by centralized unit (the CAU), thereby enabling a better control over the flow, allowing easier security analysis, and post fabrication flaws fixes and patching[2].

## VI. Evaluation

In this section, we evaluate the various components that are used at our framework, including Code Authentication Unit (SHA3 and ECDSA) and the Key Management Unit (specifically, the PUF therein).

### A. Code Authentication Unit

Code Authentication Unit consists of three main blocks: 1) DMA for reading the data from memory 2) optimized SHA3 implementation and 3) ECDSA verification. The DMA is configured by the boot sequencer to read the code from

[2]The current implementation uses hardware only, while our architecture is extendable with firmware component that enables post silicon flaws fixes.

TABLE I. ECDSA VERIFICATION IMPLEMENTATION RESULTS WITH BINARY CURVE SECT233R1.

| | Baseline SW Intel IVB | Vectorized SW Intel IVB+AVX | Our HW FPGA |
|---|---|---|---|
| Cycles | 2,226,927 | 405,330 | 6,720 |
| Frequency (MHz) | 2,000 | 2,000 | 100 |
| Time(ms) | 1.11 | 0.22 | 0.067 |
| Power (mW) | 2,100 | 2,600 | 386 |
| Energy (mJ) | 2,331 | 572 | 0.025 |
| Energy Efficiency (mJ x ms) | 2,587.4 | 125.84 | 0.0016 |

the memory and calculate it's hash using the SHA3 hardware block, whereas the last block is the code signature. The Key Management Unit sends the corresponding public key to the CAU. Later, code hash, code signature and the public key fed into the ECDSA block to verify the digital signature correctness. Evaluation of the ECDSA and the SHA3 implementations is follows.

*1) ECDSA evaluation:* ECDSA verification part was implemented in hardware, we used the sect233r1 curve with the recommended parameters by Standards for Efficient Cryptography(SEC) [31]. Table I compares ECDSA hardware implementation with software implementation running on Intel Ivy Bridge (IVB) modern processor [32]. Kintex-7 KC705 FPGA is used for hardware implementation. Second and third columns represent the results of the scalar and vectorized software implementations respectively (that ran on Intel Ivy Bridge processor). The last column shows the hardware implementation. The vectorized software implementation uses Advanced Vector Extensions [32]. While, the number of cycles and power consumption for software implementations were obtained from [33] and [34] respectively.

Table I shows higher number of cycles for the software implementations compared to the hardware implementation. When running the Ivy Bridge core at $2.0GHz$ core frequency. The execution times for both scalar and vectorized software implementation is much higher compared to the FPGA hardware implementation. Whereas, the hardware implementation is 16 and 3 times faster than the scalar and vectorized software implementations respectively. In addition, the *Energy* consumption and *Energy-Efficiency* are more than four orders of magnitude higher at the software implementations compared to the hardware one. This being said, the Energy-Efficiency gap between software a hardware ECDSA implementation might be even higher when running software at RISC-V core (as in Sanctum [11] ECDSA implementation) core compared to Ivy Bridge modern core.

Our ECDSA verification core was implemented using binary extension field $GF(2^{233})$ for Elliptic Curves. Binary Extension Fields have the advantage that field additions can be performed by XOR operations, therefore no carry is involved. This leads to implementations that require lesser area and have better performance. Our implementation uses existing elliptic curve cryptography (ECC) modules that implements parallel ECC processor design presented in [35]. This design requires 8 cycles for ECC point addition and 3 cycles for computing ECC point doubling. Therefore, the ECC point multiplication requires less number of cycles compared to the existing designs. Detailed description on the hardware requirements of the ECDSA verification module is as summarized in Table II.

TABLE II. HARDWARE REQUIREMENT AND POWER CONSUMPTION FOR ECDSA VERIFICATION (SECT233R1).

| | Used | Available | Util% |
|---|---|---|---|
| Slice LUT | 27170 | 203800 | 13.33 |
| LUT as Logic | 26450 | 203800 | 12.98 |
| LUT as Memory | 720 | 64000 | 1.13 |
| Slice Registers | 6722 | 407600 | 1.65 |
| Register as Flip Flop | 6722 | 407600 | 1.65 |
| Register as Latch | 0 | 407600 | 0.00 |
| Multiplexer | | | |
| F7 Muxes | 684 | 101900 | 0.67 |
| F8 Muxes | 0 | 50950 | 0.00 |

*2) SHA3 evaluation:* An optimized design of SHA3 [36] is deployed in this work. The design comes with a simplified round constants (RC) generator and new subpipelined transformation round, with unrolling factor of 2 and followed by 2-stages pipelining in between adjacent rounds (as depicted in Figure 5. The simplified RC generator is smaller in hardware area size because only the non-zero bits of the RC is stored. In SHA3, there are only maximum number of 8 non-zero bits observed in each round constant value. Therefore, the length of the 24 pre-calculated constants value is effectively reduced to a byte size.

As for the new subpipelined transformation round, the pipeline registers are inserted after the Theta ($\theta$) operation. It is observed that the longest delay in the first half of the computation is constituted of 5 XORs while the second part which includes Pi ($\pi$) to Iota ($\iota$) covers the longest delay of 2 XORs, 1 AND and 1 XOR. Such subpipelining approach reduce the critical path by approximately half and leading to achieving maximal clock frequency.

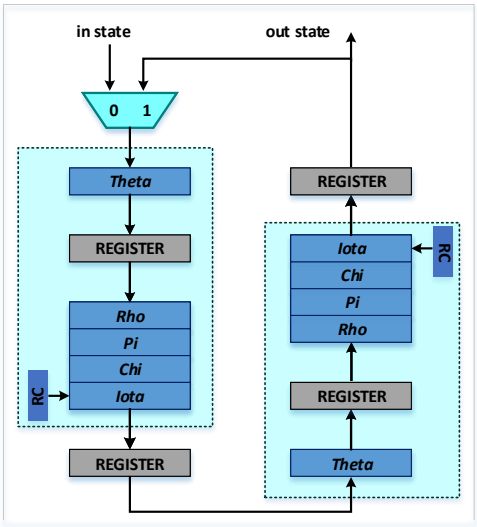


Fig. 5. Optimized SHA3 Implementation

Subpipelining is efficient in critical path reduction where the maximum attainable frequency can be greatly improved. Meanwhile, unrolling promotes throughput enhancement via hardware replication as the total execution cycles is reduced. In this SHA3, the register pipelines are inserted upon every unrolled module. Thus, the simultaneous message processing is enabled without imposing additional delays in the existing critical path.

TABLE III. EXPERIMENTAL RESULT FOR RECENT SHA3 HASH IMPLEMENTATIONS ON FPGA. THE NOTATION $S$, $P$ AND $U$ INDICATE THE APPROACHES USING SUBPIPELINING, PIPELINING AND UNROLLING RESPECTIVELY.

| Work | Approach (S/P/U) | Fmax (MHz) | Area (Slices) | Throughput (Gbps) | Efficiency (Mbps/Slices) |
|---|---|---|---|---|---|
| Wong et al. (2018) [36] | U [k=2]<br>P [n=2]<br>S [n=2] | 344 | 1,406 | 16.51 | 11.47 |
| Ioannou et al. (2015) [37] | U [k=2]<br>P [n=2] | 352 | 2,652 | 16.90 | 6.37 |
| Ioannou et al. (2015) [37] | U [k=2]<br>P [n=2] | 391 | 2,296 | 18.77 | 8.17 |
| Michail et al. (2015) [38] | U [k=3]<br>P [n=3] | 352 | 3,197 | 25.34 | 7.93 |
| Michail et al. (2015) [38] | U [k=3]<br>P [n=3] | 391 | 3,965 | 28.15 | 7.10 |
| Michail et al. (2015) [38] | U [k=4]<br>P [n=4] | 357 | 4,632 | 34.27 | 7.40 |
| Michail et al. (2015) [38] | U [k=4]<br>P [n=4] | 392 | 4,117 | 37.63 | 9.14 |
| Mestiri et al. (2016) [39] | S [n=2] | 317 | 4,793 | 6.34 | 1.32 |

FPGA implementation and the comparison with the recent works of SHA3 are as reflected in Table III, the highest throughput performance was presented in the work by [37], [38] where the authors employed higher unrolling factor in the SHA3 hash implementations. However, this is traded off with much larger amount of slices needed. Therefore, considering the throughput/area factor, the SHA3 deployed in this work has the highest implementation efficiency (11.47 Mbps/Slices) compared to the existing works.

### *B. CoLPUF in Key Management Unit*

This work integrated a configurable LFSR-based PUF (CoLPUF) that was presented at [40]. The PUF is part of the Key Management Unit (KMU), which is used for generating the unique and non-reproducible secret key. The design uses a Linear Feedback Shift Register (LFSR) and the output in each state of the LFSR serves a the new challenge to the PUF. CoLPUF is advantageous as the design is capable of extending the length of the response bits through LFSR. In other words, it promotes the concept of constant-resource scalability to increase the response size without any considerable increase in hardware.

There are two main phases in deriving the secure key from the PUF and error correction codes, namely the initialization and the re-generation of the key. In the first stage, a syndrome is computed from the PUF through error correction code (ECC) encoding circuit and stored in a memory as a pair with the challenge. This is used to correct any detected errors found in the PUF response. Next, in the latter stage, the noisy response from the PUF is taken and its corresponding syndrome (for a specific challenge) are sent to the ECC decoder. With the aid of the stored syndrome, the ECC will correct the errors occurred in the response of the PUF. Errors occurs due to any changes in the voltage and temperature. To be precise, this design employed BCH(15,7,2) as error correction code, where a 7-bit message is encoded into 15-bit code (7-bit message with 8-bit syndrome) and the correction can be performed up to 2 errors in any position.

The hardware implementation for the designed CoLPUF on Nexys-4 Artix-7 are as tabulated in Table IV. In addition to that, the benchmarking with various existing PUF designs reported in the literature is summarized in Table V. Note that the specific comparison metrics used here are the uniqueness (U) and the reliability (R). Based on the comparison results, it is evident that CoLPUF used in this work demonstrates good uniqueness and reliability.

TABLE IV. HARDWARE SPECIFICATION FOR COLPUF ON ARTIX-7 FPGA PLATFORM

| Parameter | Encoder | Decoder | PUF |
|---|---|---|---|
| #Slices Register | 14 | 67 | 331 |
| #Slices LUT | 13 | 105 | 4245 |
| #LUT- FFpairs | 10 | 47 | 331 |

TABLE V. COMPARISON OF VARIOUS PUF DESIGNS

| PUF Design (response size) | U(%) Uniqueness | R (%) Reliability | FPGA Platform | Area Consumption |
|---|---|---|---|---|
| CoLPUF (128) [40] | 49.22 | 99.99 | Artix-7 | 256 ROs<br>16-bit LFSR |
| RO PUF (128) [41] | 46.15 | 99.52 | Virtex-4 | 1024 ROs |
| SRAM PUF (128) [42] | 49.97 | >88 | FPGA | 4800 SRAM memory bits |
| Latch PUF (128) [43] | 46 | >87 | Spartan-3 | 2x128 slices |
| Flip Flop PUF (4096) [44] | $\approx 50$ | >95 | Virtex-2 | 4096 Flip Flops |
| Butterfly PUF (64) [45] | $\approx 50$ | 94 | Virtex-5 | 130 slices |
| CRO PUF (127) [46] | 43.50 | 96 | Spartan-3 | 64 slices for ROs |

## VII. CONCLUSION

We have presented secure boot framework which was implemented into our RISC-V lightweight SoC. The framework uses optimized Elliptic Curve Digital Signature Algorithm (ECDSA), Secure Hash Algorithm 3 (SHA3) hashing algorithm, Physically Unclonable Function (PUF) and Direct Memory Access (DMA). We demonstrated the framework on RISC-V based SoC. Detailed analysis of performance and security for the platform is presented. In future work, we plan to extend our secure processor framework to include more security features. Remote attestation will be implemented on top of the secure boot flow. In addition, we plan to integrate trusted execution environment into the RISC-V SoC.

## ACKNOWLEDGEMENT

This research is supported by NRF-BICSAF project (Project ID: NRF2016NCR-NCR001-006).

## REFERENCES

[1] A. Furtak, Y. Bulygin, O. Bazhaniuk, J. Loucaides, A. Matrosov, and M. Gorobets, "Bios and secure boot attacks uncovered," in *The 10th ekoparty Security Conference*, 2014.

[2] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux, "Laser-induced fault injection on smartphone bypassing the secure boot," *IEEE Transactions on Computers*, 2018.

[3] A. Cui and R. Housley, "Badfet: defeating modern secure boot using second-order pulsed electromagnetic fault injection," in *11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association*, vol. 180, 2017.

[4] C. Lab., "Severe security advisory on amd processors." [Online]. Available: https://safefirmware.com/amdflaws_whitepaper.pdf, 2018.

[5] DARPA., "Intelligent Design of Electronic Assets (IDEA) and Posh Open Source Hardware (POSH)." [Online]. Available: https://www.darpa.mil/attachments/eri_design_proposers_day.pdf, 2018.

[6] A. Bradbury, G. Ferris, and R. Mullins, "Tagged memory and minion cores in the lowrisc SoC," *Memo, University of Cambridge*, 2014.

[7] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanovic, and K. Asanovic, "A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators," in *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*, Sept 2014, pp. 199–202.

[8] A. Menon, S. Murugan, C. Rebeiro, N. Gala, and K. Veezhinathan, "Shakti-T: A RISC-V Processor with Light Weight Security Extensions," in *Proceedings of the Hardware and Architectural Support for Security and Privacy*. ACM, 2017, p. 2.

[9] M. M. Wong, J. Haj-Yahya, and A. Chattopadhyay, "SMARTS: secure memory assurance of RISC-V trusted SoC," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2018, p. 6.

[10] K. Project, "Secure hardware enclave," [Online]. Available: https://keystone-enclave.org/, 2018.

[11] I. Lebedev, K. Hogan, and S. Devadas, "Secure boot and remote attestation in the sanctum processor," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 46–60.

[12] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. IEEE, 1997, pp. 65–71.

[13] Unified-EFI, "Unified extensible firmware interface specification: Version 2.2 d," 2010.

[14] S. Sau, J. Haj-Yahya, M. M. Wong, K. Y. Lam, and A. Chattopadhyay, "Survey of secure processors," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2017 International Conference on*. IEEE, 2017, pp. 253–260.

[15] X. Ruan, "Boot with integrity, or dont boot," in *Platform Embedded Security Technology Revealed*. Springer, 2014, pp. 143–163.

[16] W. Arthur, D. Challener, and K. Goldman, *Platform Security Technologies That Use TPM 2.0*. Berkeley, CA: Apress, 2015, pp. 331–348. [Online]. Available: https://doi.org/10.1007/978-1-4302-6584-9_22

[17] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[18] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," *Designs, codes and cryptography*, vol. 19, no. 2-3, pp. 173–193, 2000.

[19] A. Liu and P. Ning, "Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Proceedings of the 7th international conference on Information processing in sensor networks*. IEEE Computer Society, 2008, pp. 245–256.

[20] US Department of Commerce/National Institute of Standards and Technology, "Digital signature standards," Federal Information Processing Standards Publication.

[21] S. Blake-Wilson and M. Qu, "Standards for efficient cryptography (sec) 2: Recommended elliptic curve domain parameters," *Certicom Research, Oct*, 1999.

[22] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[23] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the AEGIS single-chip secure processor using physical random functions," in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005, pp. 25–36.

[24] Xilinx, "Developing tamper-resistant designs with zynq ultrascale+ devices." [Online]. Available: https://www.xilinx.com.

[25] Altera/Intel, "Secure device manager for Intel R Stratix 10 devices provides fpga and soc security."

[26] J. Kong, F. Koushanfar, P. K. Pendyala, A.-R. Sadeghi, and C. Wachsmann, "Pufatt: Embedded platform attestation based on novel processor-based pufs," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[27] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The RISC-V instruction set manual, volume I: user-level ISA, version 2.0," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[28] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanovic, "The RISC-V instruction set manual volume II: privileged architecture version 1.9," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129, Jul 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.html

[29] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[30] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv preprint arXiv:1801.01203*, 2018.

[31] S. SEC, "2: Recommended Elliptic Curve Domain Parameters," *Standards for Efficient Cryptography Group, Certicom Corp*, 2000.

[32] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah, "A 22nm IA multi-CPU and GPU system-on-chip," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*. IEEE, 2012, pp. 56–57.

[33] M. Bluhm and S. Gueron, "Fast software implementation of binary elliptic curve cryptography," *Journal of Cryptographic Engineering*, vol. 5, no. 3, pp. 215–226, 2015.

[34] J. Haj-Yahya, A. Mendelson, Y. B. Asher, and A. Chattopadhyay, *Energy Efficient High Performance Processors: Recent Approaches for Designing Green High Performance Computing*. Springer, 2018.

[35] C. Rebeiro and D. Mukhopadhyay, "High speed compact elliptic curve cryptoprocessor for fpga platforms," in *International Conference on Cryptology in India*. Springer, 2008, pp. 376–388.

[36] M. M. Wong, J. Haj-Yahya, S. Sau, and A. Chattopadhyay, "A new high throughput and area efficient SHA-3 implementation," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[37] L. Ioannou, H. E. Michail, and A. G. Voyiatzis, "High performance pipelined FPGA implementation of the SHA-3 hash algorithm," in *2015 4th Mediterranean Conference on Embedded Computing (MECO)*, June 2015, pp. 68–71.

[38] H. E. Michail, L. Ioannou, and A. G. Voyiatzis, "Pipelined SHA-3 implementations on FPGA: Architecture and performance analysis," in *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, ser. CS2 '15. New York, NY, USA: ACM, 2015, pp. 13:13–13:18. [Online]. Available: http://doi.acm.org/10.1145/2694805.2694808

[39] H. Mestiri, F. Kahri, M. Bedoui, B. Bouallegue, and M. Machhout, "High throughput pipelined hardware implementation of the KECCAK hash function," in *2016 International Symposium on Signal, Image, Video and Communications (ISIVC)*, Nov 2016, pp. 282–286.

[40] "colpuf : A novel configurable lfsr-based puf."

[41] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th Annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 9–14. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278484

[42] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "Fpga intrinsic pufs and their use for ip protection," in *Cryptographic Hardware and Embedded Systems - CHES 2007*, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 63–80.

[43] D. Yamamoto, K. Sakiyama, M. Iwamoto, K. Ohta, T. Ochiai, M. Takenaka, and K. Itoh, "Uniqueness enhancement of puf responses based on the locations of random outputting rs latches," in *Cryptographic Hardware and Embedded Systems – CHES 2011*, B. Preneel and T. Takagi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 390–406.

[44] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic pufs from flip-flops on reconfigurable devices," in *ser. 3rd Benelux Workshop on Information and System Security (WISSec 2008)*.

[45] S. S. Kumar, J. Guajardo, R. Maes, G. Schrijen, and P. Tuyls, "Extended abstract: The butterfly puf protecting ip on every fpga," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, June 2008, pp. 67–70.

[46] D. Merli, F. Stumpf, and C. Eckert, "Improving the quality of ring oscillator pufs on fpgas," in *Proceedings of the 5th Workshop on Embedded Systems Security*, ser. WESS '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:9. [Online]. Available: http://doi.acm.org/10.1145/1873548.1873557