Reconfigurable Hardware Root-of-Trust for Secure Edge Processing

Alan Ehret, Eliakin Del Rosario, Carsten Schwicking†, Karen Gettings †, Michel A. Kinsy
Secure, Trusted, and Assured Microelectronics (STAM) Center
Ira A. Fulton Schools of Engineering, Arizona State University
{ aehret1, mkinsy)@asu.edu
† MIT Lincoln Laboratory
{carstens, karen.gettings}@ll.mit.edu

2021 IEEE High Performance Extreme Computing Conference (HPEC) | 978-1-6654-2369-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/HPEC49654.2021.9622830

***Abstr*act— In this work, we introduce key security primitives for secure edge processing based on a reconfigurable hardware Root-of-Trust. We present a reference architecture, named RECORD SoC, that makes use of these security primitives. These modules can be configured to support a variety of security features, including isolated firmware, I/O access policies, and digital signature verification of an initially untrusted application. We demonstrate that a hardware root-of-trust can be implemented flexibly and efficiently for an edge system vulnerable to physical access-based attacks, requiring only a 16.8% area overhead. Except for a one-time application verification at startup, the security features we examine represent only 0.08% of the latency required to process a sample of sensor data.**

I. Introduction

The shift to accelerator driven mobile architectures has significantly improved the energy efficiency of embedded systems in recent years [1]. These improvements have led to the rise of a new class of "edge" systems that perform much of their computation locally rather than relying on a remote infrastructure [2]. Edge systems leverage improved energy efficiency to support smaller devices, longer battery lifetimes and deployment of more devices for a growing variety of applications.

Edge systems cover a wide range of applications, but their deployment environments all share common features. Specifically, most edge devices are deployed outdoors, in remote locations, far away from the people responsible for their security. Edge systems face unique security challenges due to the characteristics of their deployments [3]. While other classes of systems, like data centers, can rely on the physical security of their building to keep intruders out, mitigating most physical access-based attacks, edge systems do not have this luxury. Unattended edge devices are vulnerable to complete take-over by an attacker with physical access.

The limited power and performance of edge systems exacerbates the security challenges they face. Size and weight requirements often limit edge systems to small single-use batteries. With such tight power constraints, many edge systems have little overhead available for security measures. Anti-tamper techniques exist to prevent physical access to a device without destroying any critical information it may hold. However, many anti-tamper techniques require constant power which is frequently impractical for battery powered devices that must last years in the field. As edge devices are deployed into more security or safety critical applications, attacks that leverage physical access to execute arbitrary code on edge devices must be properly mitigated.

To that end, we explore uses of the RECORD SoC to provide security guarantees based on a hardware root-of-trust in an energy efficient edge system. The RECORD SoC is an accelerator driven architecture with a hardware Root-of-Trust (RoT) monitoring system to ensure that only authorized applications are executed and that they are executed without abusing their access to hardware resources or local data. The RECORD SoC leverages a hierarchy of domain specific accelerators to perform the required processing with maximum energy efficiency. Low energy accelerators perform initial processing on sensor data to determine if further processing by higher energy accelerators is necessary. A programmable Finite State Machine (pFSM) and bus access controls monitor the state of application execution and enforce security policies. Isolated, on-chip memories provide support for trusted firmware and untrusted application code. Several e-Fuses [4] ensure user configurable hardware modules, including the pFSM, bus access policies, and firmware memories, cannot be altered after device deployment, creating the hardware root-of-trust. The RECORD SoC provides a configurable hardware root-of-trust that enables co-design of applications and hardware-backed security features. A configurable hardware root-of-trust allows a single fabricated SoC architecture to support multiple hardware, software co-designs, allowing RECORD to be used in a broader range of applications. Our contributions in this work include:

- An exploration of the hardware, software co-design space to examine how specific security features (e.g. firmware and application code isolation) can be implemented on the RECORD SoC.
- A comparison of RECORD hardware root-of-trust features with alternative implementations, such as separate physical hardware.
- An analysis of the latency overheads created by the hardware and software-based security policies.

. DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

978-1-6654-2369-4/21/$31.00 ©2021 IEEE

II. Related Work

Keystone is an open source framework for developing "trusted execution environments" [7]. Keystone provides system architects with a toolbox of hardware primitives and software runtimes that can be combined in different ways to mitigate a variety of threat models. Features of the Keystone framework can be combined to mitigate attacks based on arbitrary code execution, physical probing, or side-channel attacks. Modules in the Keystone framework include a secure boot ROM (zeroth stage bootloader), trusted source of randomness, monotonic counters, on-chip memory, and partitioned caches, to name a few. Keystone relies on a Security Monitor running in Machine mode (the highest privilege mode in RISC-V) to provide software-based security features.

At a high level, Keystone and RECORD share a similar goal, namely, supporting a configurable set of features for building a trusted system. However, RECORD and Keystone differ in a few key aspects. The first difference is the scope of their target performance, area and energy budgets. The second is when the system specific configuration occurs. Keystone provides a flexible set of open source hardware and software primitives for use when designing a new system. All decisions about the trusted hardware modules are made at design time, before device fabrication. Meanwhile, RECORD provides a fixed set of configurable hardware features that a programmer can adapt to their requirements during their application development. Supporting hardware, software co-design of the trusted hardware modules and application program is crucial for maximizing efficiency of energy constrained edge systems. Supporting configuration after device fabrication also allows the configurable hardware to be used in a greater number of applications, enabling the creation of a more flexible system. The Keystone framework is not optimized for a single class of systems with similar performance, area, or energy requirements. As such, some of its primitives may not be suitable for the resource constrained operation necessary in edge devices. The RECORD SoC has been designed from the ground up to maximize energy efficiency while providing a hardware root-of-trust that ensures only authorized applications are executed.

III. Threat Model

As an edge device, the RECORD SoC must be resilient to the wide range of attacks possible when an attacker can gain physical access or close proximity to the device. In this work, we assume an attacker can physically access a RECORD device in its deployment environment. The attacker may interface with any I/O on the RECORD device, such as programming ports or the UART. The attacker is assumed to be capable of PCB level modifications, i.e., removing or replacing ICs and tampering with or probing any PCB traces. The attacker is assumed to treat the RECORD SoC as a black box. Attacks based on IC decapsulation, circuit editing, or fault injection (not based on I/O ports) are outside the scope of the threat model.

There are several ways to attack a design prior to its deployment. For example, supply chain attacks targeting either the device's hardware or software. However, these attacks are not specific to edge systems and require mitigations beyond designing specific features into the architecture. As such, we assume the RECORD SoC is trojan free and is initially configured by a trusted application developer. Attacks prior to device deployment are beyond the scope of this work. Specific attacks that fall within the scope of the threat model include scan-chain based attacks [8], off-chip memory probing or modification [6], timing side-channel attacks [9], and power or clock fault injections [10].

IV. Architecture Description

The RECORD SoC is an accelerator focused architecture designed to support a variety of applications in a common domain such as audio signal processing or machine vision. The RECORD architecture's focus on edge deployments makes energy efficiency a top priority. Energy efficiency is maximized by organizing accelerators into a hierarchy, with energy requirements increasing as data progresses through the accelerator hierarchy. Accelerators at the bottom of the hierarchy perform initial processing of sensor data which reduces the computation larger and more energy intensive accelerators must perform. The RECORD SoC architecture and its features are described in detail in [5]. A brief summary of the relevant features is included in this section.

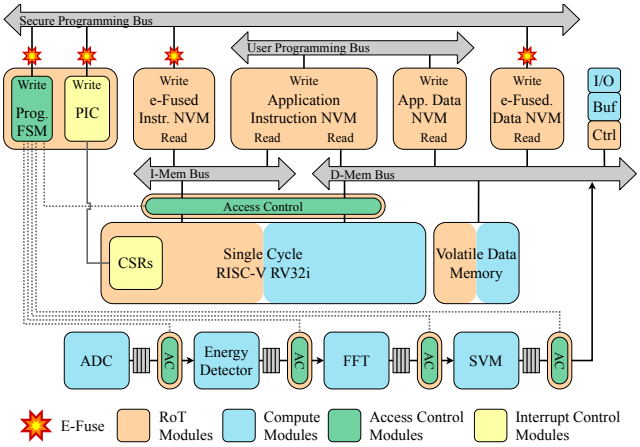

Fig. 1. The RECORD SoC Architecture

The RECORD SoC creates a hardware Root-of-Trust with configurable hardware modules that monitor the state of application execution and enforce user defined security policies. The key modules that form the hardware Root-of-Trust include a programmable Finite State Machine (pFSM), hardware-enforced bus access policies, e-Fused configuration memory, and a RISC-V microcontroller. The pFSM supports user defined transitions between active security policies. The Access Control (AC) modules attached to each bus master in the SoC enforce the currently active security policy, preventing unauthorized bus accesses at the hardware level. The e-Fused configuration memories enable users to co-design and test different FSMs and access control policies with their application software before making them permanent and deploying the device. The RISC-V microcontroller boots from e-Fused non-volatile memory (NVM) to ensure only trusted code is executed at bootup during deployment. The microcontroller can perform security related tasks that are too complex to implement with the pFSM and access control modules alone. Collectively, the pFSM, access controls, e-Fused configuration memory and microcontroller are known as the Root-of-Trust (RoT) Unit.

The RoT Unit is meant to provide a flexible set of security features that an end user can configure to meet their

978-1-6654-2369-4/21/$31.00 ©2021 IEEE

requirements. The flexibility of programmable security features creates a more general purpose SoC that can meet the energy and performance requirements of a wide range of applications while still providing strong deployment-specific security guarantees which are backed by hardware. The flexibility to tailor hardware security policies to each application or device deployment improves energy efficiency by removing unnecessary features. One security feature assumed to be included in all deployments is the isolation of a trusted device firmware and an initially untrusted application program. Section V goes into detail about individual security features and how they are supported.

The SoC has been designed to allow system architects to quickly replace accelerators (at design time) to support a different application domain. The RECORD RoT Unit does not rely on specific accelerators or their features for security. Instead, the RoT Unit connects to accelerators with a standard interface. In this work, the RECORD SoC is optimized for audio processing applications. However, different accelerators could be selected to tailor the SoC for other application domains.

Currently the RECORD SoC includes three accelerators optimized for audio processing applications. The first (lowest energy) accelerator is an energy detector. The energy detector accelerator analyzes an audio signal and requests an interrupt from the pFSM when an energy threshold is crossed. The energy detector accelerator eliminates further processing of quiet signals. The next accelerator in the hierarchy is an FFT accelerator. The FFT accelerator transforms the audio signal into the frequency domain for more detailed analysis. The application program may analyze the FFT output to determine if more energy intensive processing is warranted. For example, the application program may check for the presence of specific frequencies. The final accelerator in the hierarchy implements a Support Vector Machine (SVM) to classify the frequency spectrum of the signal into one of several categories, e.g. "person" or "not a person". Then, the application program may read the classification result from the memory mapped FIFO to output the results or perform some other action.

V. Security Feature Design and Implementation

A. Configurable Module Use

Co-design of hardware-enforced security policies and application programs is at the heart of the RECORD SoC design philosophy. Supporting hardware root-of-trust configuration during application development provides several advantages, namely, a more flexible architecture and increased energy efficiency. With the pFSM and access control modules, the same SoC can support hardware-based monitoring of program execution and access policy enforcement for a variety of applications.

pFSM Design – Application developers generate a pFSM configuration by creating a state transition diagram. Each state describes an access policy and may or may not trigger a microcontroller interrupt, activating the firmware. State transitions occur when the FSM inputs meet a specified condition. For example, a microcontroller write to the FSM input CSR or an accelerator interrupt request. The state transition diagram is converted into Next State and FSM Output LUTs for writing to the pFSM. Figure 2 depicts an example FSM design. For clarity, policy details are replaced with a single policy number 0-3 in the diagram. In the example FSM, the device enters the START state after a power-on reset. In the START state, firmware verifies the digital signature of the stored application. Upon a failed verification, the FSM enters an ERROR state and prevents execution of any additional software. If verification is successful, the FSM enters a loop of states that wait for accelerator interrupt requests, then acknowledge them by initiating a microcontroller interrupt. While the FSM is in the IDLE state, the application performs any necessary accelerator setup. When the energy detector accelerator's threshold is crossed, it requests an interrupt to move data up the accelerator hierarchy. As the FSM expects the energy detector interrupt request, it transitions states from IDLE to ED_INTR which triggers an interrupt in the microcontroller. After the ED_INTR state, the FSM immediately transitions to the wait state for the next accelerator (WAIT_FFT). The "wait and acknowledge" cycle repeats for the remaining accelerators in the hierarchy. After the interrupt for the final accelerator in the hierarchy is triggered, the FSM enters the IDLE state again. Here, the application may respond to the final results before re-activating the lowest level accelerator and repeating the cycle for the next batch of sensor data to be processed.

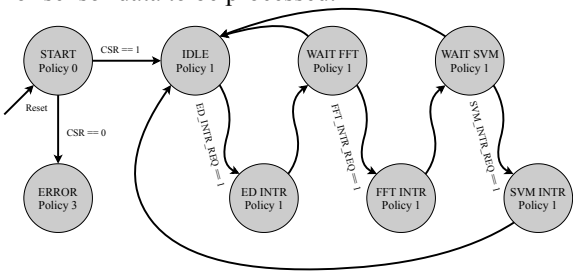

Fig. 2. State Transition Graph for pFSM.

Access Policy Design – Access policies are specified for each FSM state with an address range and read/write permissions. Read permissions on the instruction fetch access control module are equivalent to execute permissions. To minimize access to nonessential hardware resources, policy designers must avoid including unnecessary memory or peripherals in the address ranges covered by their policies. The memory map of the RECORD SoC has been carefully laid out to support access to only the essential hardware resources in a variety of situations. For example, an application can be granted access to application data memory, peripherals, and accelerators without authorizing access to firmware data memory. Similarly, at startup, a firmware bootloader can access firmware volatile and non-volatile data memory with a single range of addresses without enabling access to any other device resources. Access policies can be directly encoded to binary for uploading to the RECORD SoC access policy registers.

Figures 3 and 4 show the RECORD memory map and several example access policies for the microcontroller data and instruction buses. The example policies show that many different types of access can be created with simple contiguous ranges of addresses. For example, a policy for use when firmware verifies a signature of the application instruction memory can be created without overlapping accelerators, I/O or data memory reserved for the application program. Eliminating access to the read-only application instruction memory creates a policy suitable for most (non-I/O) firmware operations.

978-1-6654-2369-4/21/$31.00 ©2021 IEEE

Supporting I/O access in firmware does require access to all data memory and accelerator interfaces, but the firmware is part of the trusted compute base and can be expected not to abuse that access. Applications can gain access to the I/O but also must be granted access to the accelerators. If this is an issue for specific applications, all I/O can be handled by the trusted firmware instead. The most restricted application access policy prevents access to any memory outside the reserved volatile data memory. Data bus access can also be cut off completely while the microcontroller is idle.

The instruction bus policies (Figure 4), are simpler than the data bus policies. Typically, one policy will provide read-only access to the e-Fused firmware instruction NVM. Another policy will provide access to application instruction NVM. Multiple isolated applications can be supported with additional policies to segment the application instruction NVM.

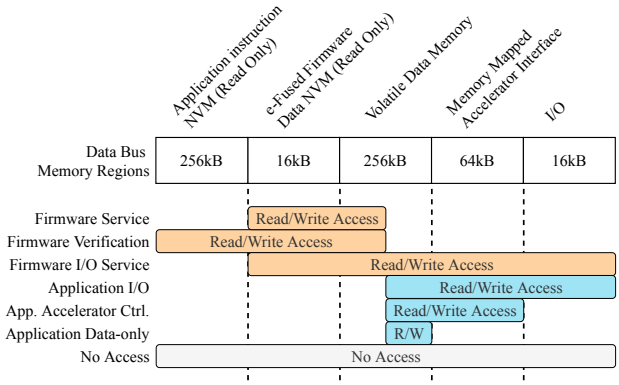

Fig. 3. Data bus policy examples.

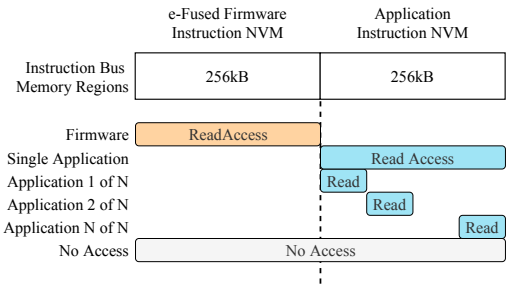

Fig. 4. Instruction bus plolicy examples.

Firmware Design – Firmware stored in the e-Fused non-volatile memory provides immutable, security critical sections of code that can be trusted after the device has been deployed in the field. Firmware is written and compiled with the same tools as any application program. Separate binary images are generated for the isolated e-Fused instruction and data NVM. The firmware represents the software side of the RECORD system's trusted compute base. To function correctly, the firmware and FSM configuration must be designed together and may coordinate. For example, the trusted firmware may request FSM state transitions to apply different security policies. Firmware can simply monitor the application execution or it can provide basic services such as I/O or accelerator access, in addition to its execution monitoring. The selected firmware mode (monitor or service provider) will vary depending on application energy and security requirements.

Programming and Configuration – Figure 5 demonstrates the RECORD SoC programming and configuration flow. The configuration process requires three main inputs, the pFSM state transition diagram, firmware code and application code. The state transition diagram is converted to Next State LUT and FSM Output LUT memory images. Access policies attached to each state are encoded to small binaries. Firmware and application code are compiled with a standard RISC-V compiler, e.g. [11]. The firmware binary requires no additional processing. The application binary is digitally signed with a trusted private key. The firmware includes the public key in its non-volatile read-only memory. The application signature is concatenated with the application non-volatile read-only memory image. Once the access policies, Next State LUT, FSM Output LUT, firmware image, application binary and application signature have been generated, all can be uploaded to the RECORD SoC. The application binary and its signature are programmed with the "User" i.e. non-secure programming bus. The remaining configuration sources are programmed with the "Secure" programming bus. With all the resources loaded onto the SoC, the whole system can be tested. If programming or configuration errors are discovered, the state transition diagram, firmware code, or application code can be updated and the whole configuration process repeated. Once a developer is satisfied with their system, the e-Fuse command can be issued over the Secure programming bus to prevent further modifications to the access policies, LUTs or firmware. Updated application binaries and associated signatures may still be uploaded over the User programming bus. The firmware boot process ensures that the application is still signed by the trusted public key.

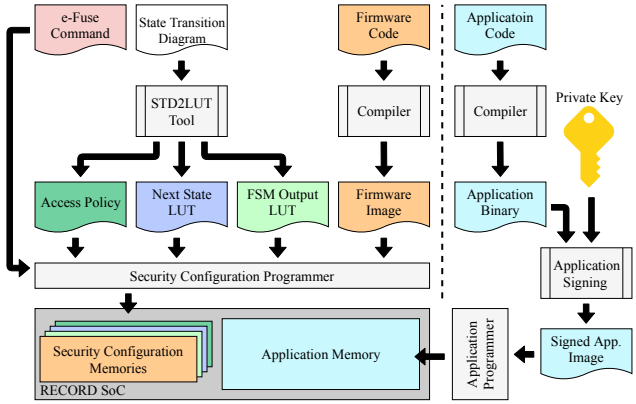

Fig. 5. Compilation and Configuration flow.

B. *Security Features*

The configurable hardware and software modules discussed above are combined to create a variety of different security features. Some are common to all applications while others are may be application or deployment specific. Table I lists the security features explored in this work, though others may be supported.

Boot Verification – At startup, the firmware should verify the authenticity and integrity of the application stored in the Application NVM. Verification can be performed with a digital signature algorithm, e.g. ECDSA [12], or some other means. The configurable hardware root-of-trust is not limited to any specific verification algorithm, allowing each developer to make

978-1-6654-2369-4/21/$31.00 ©2021 IEEE

appropriate security and performance trade-offs. To verify an application, the pFSM transitions to a dedicated verification state. The access control policies disable all accelerator access. On the data bus, only the application instruction memory, e-Fused firmware data NVM, and firmware-reserved volatile data memory can be accessed. On the instruction bus, only firmware code may be fetched. When the verification algorithm is complete, the firmware writes to the FSM input CSR to signal the result. The pFSM transitions to either an error state or the next state in the application.

I/O Access Control – Permission to use the I/O resources of the SoC can be restricted at different points of firmware or application execution. To limit use of I/O, the pFSM must be transitioned to a state with an access policy address range that does not extend to the I/O region of memory. Access to I/O could be disabled for long running periods of an application without the need for I/O, minimizing the time an attacker could exploit mistakes in the application to leak information. Alternatively, I/O could be completely handled by firmware with a system call style interface. Developers have the flexibility to choose based on their security and performance requirements as well as their confidence in the security/correctness of the application program.

Segmented Memory – Similar to the I/O access controls, the pFSM and access policies can be used to isolate sensitive sections of application memory. These sections of memory can be made available only when needed to prevent corruption from an exploited application program or other attack. To isolate segments of memory, the pFSM includes states for each segment to be isolated. Each state provides access to the associated segment of memory and any other data memory necessary to complete the applications current task. The pFSM encodes a specific order that individual segments are activated preventing un-expected use of the data and limiting the availability of the data to an attacker.

Isolated Applications – Whole applications can be isolated in a similar manner as application memory segments. A pFSM state can be created for each application. Access policies will isolate their instruction and data memory. Firmware can make use of hardware timers to time-multiplex applications. If hardware timers are not available, the pFSM could be configured as one. Although a pFSM based timer would hurt energy efficiency and limit the use of pFSM states for other features.

Accelerator Isolation – Individual accelerators can be isolated while they are idle by removing read and write permissions from the set access policy for each accelerator output. Isolating and disabling accelerators requires a dedicated pFSM state for each accelerator. The state enables read or write permissions for the given accelerator, while all other states disable all bus access. Disabling accelerators prevents attackers from abusing the accelerator hierarchy to waste resources with frequent or overlapping (i.e. multiple accelerators active simultaneously) activations. Disabling bus access for idle accelerators provides limited support for malicious accelerator IPs as well, although this is beyond the scope of the threat model considered in this work and will be examined in future work.

Accelerator Rate Limiting – An attacker may trigger frequent accelerator activations to waste the limited battery of a device. By triggering firmware interrupts between accelerator activations, the pFSM allows the firmware to track the frequency of activations and the usefulness of the data produced. If frequent activations strain the battery and provide little useful computation, the firmware can implement rate limiting by entering a sleep state before activating the next accelerator. Such a rate limiting scheme would require an extra pFSM state for each accelerator activation. Firmware would require minimal data bus access, e.g. firmware-reserved volatile data memory only. If the application is required to assist with rate limiting, then at least two pFSM states per accelerator would be required (one for firmware, one for application code).

TABLE I. SECURITY FEATURES AND REQUIREMENTS.

Feature	Required Resources
Boot Verification	1 pFSM state, 1 Policy, Firmware Support, Public Key
I/O Access Control	1 pFSM state, 1 Policy
Segmented Memory (Within Appliation)	1 pFSM state and 1 policy per Segment, Firmware Support
Isolated Applications	1 pFSM state and 1 Policy per App., Firmware Support
Accelerator Isolation	1 pFSM state and 1 Policy per Accelerator
Accelerator Rate Limiting	1 pFSM state and 1 Policy per Accelerator, Firmware Support

VI. SECURITY DISCUSSION

On the RECORD SoC, trusted firmware is meant to support and protect more complex and un trusted applications. Under the threat model described in Section III, the application developers are trusted not to create outright malicious applications. However, this does not mean the application is not vulnerable to exploits initiated by an attacker in the field. One of the primary goals of the Root-of-Trust Unit is to minimize the damage potential exploits of the application program can cause. Possible attacks include buffer overflow exploits and Return-Oriented-Programming (ROP) based attacks. An attacker could also attempt to modify or entirely replace the application program with a malicious version.

Much of the attack surface in the RECORD SoC, and low-power edge systems in general, is already restricted to an attacker. Device I/O remains one of the few avenues for attacks and memory corruption vulnerabilities continue to plague modern programs. To minimize the impact of memory corruption vulnerabilities, the pFSM and access policies are co-designed with the application to minimize the scope of accessible memory during I/O operations by transitioning to dedicated I/O states and access policies.

Software vulnerabilities are not the only threat faced by deployed RECORD devices. An attacker with physical access could easily re-program the non-volatile application memory (but not the e-Fused firmware memory) in the field. Without proper precautions, the program could be modified to produce incorrect results or perform entirely different tasks for the attacker. To prevent this, the RECORD firmware is expected to perform some form of authentication and integrity checking on the application before execution begins. This verification is supported with non-volatile data memory for digital signature storage. A separate e-Fused firmware data NVM supports storage of associated public keys to verify the application's digital signature. The public key must be stored in the e-Fused NVM memory to prevent attackers from altering it after

978-1-6654-2369-4/21/$31.00 ©2021 IEEE

deployment. Using schemes based on public and private keys means no secret must be stored on the RECORD SoC to verify the authenticity of an application.

For mitigation, the single-cycle RISC-V microcontroller performs any instruction in one clock cycle, making the development of constant-time software easier. Instruction and data caches are not included in the design to support constant memory access times. SRAM is used for the volatile memory in the system instead of DRAM to support lower and constant memory access latency. As an edge system, a relatively small amount of SRAM memory is sufficient for the RECORD SoC architecture. The constant instruction execution and memory access latency make it possible to mitigate most timing side channels as long as the software is correctly written. Without any caches, all cache-based side channels are also eliminated.

VII. Evaluation

To evaluate the overhead of RECORD's configurable security features, we emulate the RECORD SoC on an FPGA. The emulated SoC includes three accelerators for audio signal processing and classification, an Energy Detector, an FFT, and Support Vector Machine (SVM).

TABLE II. Comparison with Alternative Architectures.

Architecture	LUTs	BRAM Tiles
Unprotected MCU	2472	94
Two MCUs	4943	182
MCU + pFSM	2802	190

TABLE III. Security Overhead - Area.

Module	LUTs	BRAM Tiles	DSPs
Accelerator Interconnect	1,273	68	0
Energy Detector	300	0	1
FFT	4,001	7	10
SVM	16,348	0	4
RoT Unit Total	4,446	190	0
Accelerator Total	21,922	75	15

In Table II, we evaluate the overheads of the configurable hardware modules. We compare the area of the architecture described in Section IV with the area of a standalone microcontroller (MCU) without any hardware root-of-trust, and a two-microcontroller system that uses one MCU for the application and another for the trusted firmware and monitoring. In the two MCU system, the second MCU replaces the pFSM module. The area of modules common to each design, i.e. the accelerators and supporting logic, are omitted from the comparison. Intuitively, the two-MCU system is approximately double the area of the unprotected MCU system. Meanwhile the RECORD (MCU + pFSM) system, requires only 13.3% more LUTs than the unprotected MCU system. By leveraging a programmable FSM instead of a second microcontroller, RECORD saves a significant amount of area to achieve the same security guarantees. Additionally, the pFSM can process and respond to a change of inputs in a single clock cycle, while a microcontroller-only solution will take several clock cycles to make a function call and execute a response.

While Table II shows the overhead of the hardware root-of-trust modules relative to alternative implementations, Table III demonstrates that the area overhead relative to the compute modules in the system is reasonable. The three example accelerators and their interconnect use a total of 21,922 LUTs, with the SVM dominating the area. At 4,446 LUTs for the entire Root-of-Trust Unit (the pFSM, access controls, interrupt controller, microcontroller and e-Fused memories), the RoT Unit represents 16.8% of the entire design logic. The RoT Unit takes up 71.7% of the system memory, although approximately half of this memory is dedicated to the deployment specific application. Relative to the LUT area, DSP area is likely negligible. As the RoT Unit does not include any DSPs, neglecting them only makes the area comparisons more conservative.

TABLE IV. Latency Comparison

Operation	Latency (Cycles)	Latency %
Energy Detector	66,536	9.25%
FFT	589,000	81.88%
SVM	63,200	8.78%
Firmware Interrupts & pFSM Transtions	602	0.08%
Firmware Application Signature verification	180,000,000	-

Table IV compares the latency of each accelerator computation and major RoT Unit operations, including the initial verification of the application's digital signature and firmware interrupts caused by pFSM state transitions. Each accelerator takes on the order of tens or hundreds of thousands of cycles to complete a computation. Approximately 719k cycles are required to complete processing of one data sample with the hierarchy of accelerators.

With accelerator isolation and separate data access policies during each accelerator execution two pFSM state transitions per accelerator are required, for a total of 6 transitions for each data sample processed. Out of the 719k cycles, only 602 (0.08%) are used by the firmware to process interrupts triggered by pFSM transitions. These 602 cycles are spent for each data sample processed.

An ECDSA [12] based application signature verification takes approximately 180 million cycles to complete. This is two orders of magnitude more than the cycles needed to process a single data sample. However, the signature verification only needs to be performed once when the device boots. Therefore, the percentage of total latency it consumes depends on how long the device stays powered on, hence the omission of latency percentage value in Table IV. The RECORD SoC is optimized for low-energy edge systems that are expected to be deployed and powered on for many weeks or months, if not several years. Over such a long duration, such an expensive one-time verification can be tolerated. With 1MHz clock, the verification would take only three minutes. If a faster verification is necessary, other works have explored cryptographic schemes more amenable to implementation on edge systems [13].

VIII. Conclusion

In this work, we have analyzed several security primitives for the RECORD SoC, its hierarchy of accelerators and its reconfigurable hardware root-of-trust. Our analysis shows that the area overhead of the hardware root-of-trust related modules

978-1-6654-2369-4/21/$31.00 ©2021 IEEE

is 43% less than the baseline system. Performance measurements show that only 0.08% of an example application is spent on security related firmware execution.

REFERENCES

[1] M. D. Hill and V. J. Reddi, “Accelerator-level parallelism,” CoRR, vol. abs/1907.02064, 2019. [Online]. Available: http://arxiv.org/abs/1907.02064

[2] W. Shi and S. Dustdar, "The Promise of Edge Computing," in Computer, vol. 49, no. 5, pp. 78-81, May 2016.

[3] A. Ehret, K, Gettings, B. Jordan, and M. Kinsy, “A Survey on Hardware Security Techniques Targeting Low-Power SoC Designs,” IEEE High Performance extreme Computing Conference, 2019.

[4] C. Kothandaraman, S. K. Iyer, and S. S. Iyer, “Electrically programmable fuse (efuse) using electromigration in silicides,” IEEE Electron Device Letters, vol. 23, no. 9, pp. 523–525, Sep. 2002

[5] A. Ehret, E. Del Rosario, K. Gettings, and M. A. Kinsy, “A Hardware Root-of-Trust Design for Low-Power SoC Edge Devices,” IEEE High Performance Extreme Computing Conference (HPEC), 2020

[6] S. Paley, T. Hoque and S. Bhunia, "Active protection against PCB physical tampering," 2016 17th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 2016, pp. 356-361

[7] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovi´c, and D. Song, “Keystone:An open framework for architecting trusted execution environments,” inProceedings of the Fifteenth European Conference on Computer Systems,2020.

[8] K. Rosenfeld and R. Karri, "Attacks and Defenses for JTAG," in IEEE Design & Test of Computers, vol. 27, no. 1, pp. 36-47, Jan.-Feb. 2010, doi: 10.1109/MDT.2010.9.

[9] P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: CRYPTO, 1999.

[10] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," in Proceedings of the IEEE, vol. 100, no. 11, pp. 3056-3076, Nov. 2012.

[11] “Risc-v gnu toolchain,” https://github.com/riscv/riscv-gnu-toolchain, 2021.

[12] Libecc Library. https://github.com/ANSSI-FR/libecc

[13] Singh, S., Sharma, P.K., Moon, S.Y. et al. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. J Ambient Intell Human Comput (2017).

978-1-6654-2369-4/21/$31.00 ©2021 IEEE