Secure Design Flow of FPGA Based RISC-V Implementation

Ali Shuja Siddiqui, Geraldine Shirley, Shreya Bendre, Girija Bhagwat, Jim Plusquellic, Fareena Saqib
Electrical and Computer Engineering, University of North Carolina at Charlotte, Charlotte, USA
asiddiq6@uncc.edu, gnichola@uncc.edu, sbendre@uncc.edu, gbhagwat@uncc.edu, jimp@ece.unm.edu, fsaqib@uncc.edu

***Abstract*—In the process of globalization, heterogeneous SoCs play an important role in an embedded application, security aspects of such a system are crucial. The system is susceptible to many attacks out of which we focus on two main attacks, namely, boot time attacks, where malware are injected to leak information and modify the functionality and run-time software attacks causing memory corruption. In this paper, we propose a hardware/software-based solution to secure the system integrity by providing secure boot which prevents malicious and unauthorized software during startup and Information Flow Tracking (IFT) technique to track the spurious data during run-time and preventing buffer overflow attacks. This proposed solution is implemented on the RISC-V and provides a self-authentication mechanism for FPGAs using TPM.**

Keywords— Information Flow Tracking (IFT), Secure Boot, RISC-V, TPM, Run-time attacks

I. INTRODUCTION

The major concern in today's hardware design is the vulnerability towards untrusted entities which results in designing a secure architecture which is capable of firmware integrity and tracking the flow of information during run time and protecting the system from software-based attacks. Internet of Things is a special purpose small-scale devices that are typically connected in nature. Due to this nature, they may need to communicate with untrusted nodes. These untrusted nodes may try to exploit some software vulnerabilities that the vendor may not be aware of at that time and when they are deployed on the field, they become susceptible to application level as well as firmware level attacks. This may lead to a system compromise.

RISC-V is an open source ISA which allows optimization, modification, and usage according to specific requirements. This makes the system running on RISC-V more susceptible to attacks at any point in time. The processor can be untrusted at boot time or run time. In this paper, we propose a solution for securing the RISC-V processor by providing Secure Boot and Information Flow Tracking schemes for RISC-V to mitigate boot level as well as runtime software-based attacks.

Malware cause malfunction in systems, reduce productivity and leakage of sensitive and secret data. Specifically, malware developers have begun to target firmware that starts up the computer system, focusing on the gap between firmware and when the operating system starts. These are rootkit and bootkit attacks. Malware can attack systems in a pre-boot environment. This happens as there is no filter to distinguish between authentic and malware software. The Unified Extensible Firmware Interface (UEFI) proposes the secure boot which is a security standard which makes sure that the system boots using only software that is trusted. Secure boot ensures the integrity of firmware and software running on a platform. It allows only those software and firmware which are approved by trusted keys. It establishes the relationship between BIOS and software which gets launched. Secure boot prevents malicious software and unauthorized operating system during the system startup process. Malware may attack bootloaders in the absence of secure boot, and this could be the reason for not booting the system at all.

Runtime attacks are also powerful attacks. The attestation of the firmware is a security service where device proves that the firmware is attested with a trusted remote entity. Today, securing of computing platforms from malicious entities is a high priority task to protect sensitive information and data. Information Flow Tracking (IFT) is a security technique which tracks the untrusted data inputs during run-time and restricts the use of such inputs along with protecting the system from buffer overflow attacks. The main observation in IFT is the difficulty in preventing fault injection, code injection and protecting the system from a security breach. Thus IFT tracks the flow of the data during execution time to identify the malicious inputs and marks them as spurious or tainted input.

Hardware Trojans have become a major threat to reconfigurable devices. Trojans can be inserted in an IC design process which can alter the design functionality or leak secure information from the system. It is difficult to identify the Trojans during post-manufacturing, which results in various methodologies to detect, distinguish and mitigate such attacks. Providing device authentication by using physically unclonable functions or trusted platform modules can be further used to detect and prevent Trojan attacks by measuring the variations such as power and delay to capture anomalies or outshoots caused by Trojans.

We demonstrate TPM based secure boot and architectural level information flow tracking extensions to track the information flow within a processor. It considers the system as a logic function which has both trusted and untrusted inputs and outputs and based on the propagation tagged or tainted bits are added. Shadow logics are used to develop rules for trusted and untrusted inputs and tracks the flow of the data. Each register and memory should be shadowed to store a one-bit tainted value. Shadow logics can be integrated with IFT to trace the tag propagation of each operand. The RISC-V architecture can be configured to implement shadow registers and by adding new instructions to the architecture to access the registers.

II. SECURITY EXTENSIONS IN RISC-V PROCESSORS

A. RISC-V

RISC-V is an open source Instruction Set Architecture developed by UC Berkeley and is currently supported by the

RISC-V foundation. It is based on the reduced instruction set principles and has found several applications in different domains like IoT, embedded applications, etc. It supports 32, 64 and 128-bit instruction widths and has a fixed base integer ISA which is mandatory for all RISC-V processor implementations. The base integer ISA supports two primary variants namely: RV32I and RV64I which provides 32-bit and 64-bit user-level address spaces respectively [15].

The base integer ISA has fixed-length 32-bit instructions with variable length encoding possible. The base integer ISA consists of 32 general purpose registers and a program counter. The privilege levels are used to provide protection between different components of the software stack [14]. The three modes include machine, supervisor and reserved modes. Machine mode has the highest privilege and is used to run simple embedded system applications. This mode does not provide protection to the system against incorrect or malicious application code. The user mode is used to run secure embedded applications and protects the system against malicious application code. The supervisor mode is used in systems which have Unix like OS. The RISC-V processor is written in Chisel (Constructing Hardware in Scala Embedded Language) which converts the code into an intermediate RTL representation (FIRRTL) which is then converted to a Verilog or C++ code.

B. *Secure Boot with Trusted Modules*

To integrate secure boot, we propose the RISC-V interface with the secure and trusted module (TM) and demonstrate a secure boot chain of trust, that validates firmware checks. The firmware checks include the signature of each piece of boot software sequentially or hierarchically verification. If the signature is valid, the system boots and firmware gives control to the operating system. Secure boot mechanism relies on public/private key pairs and digest attestation to verify the digital signatures of all firmware and software before execution. When the boot is enabled, it checks the loading software and checks whether this software is signed with the trusted keys which are already present in the firmware. The proposed scheme blocks the malicious software by using Trusted Module (TM) which is a hardware security module or secure co-processor that implements cryptographic functions. These functions include encryption, data signing, and data sealing. We consider Trusted Platform Module TPM as Trusted module.

TPM specifications are defined by the Trusted Computing Group (TCG) [1]. All TPM implementation must follow the specifications however, the specifications do provide flexibility in terms of the functionality it can implement. TPM has tamper resistant non-volatile memory. This memory can be used for storing cryptographic objects including keys and other user-defined values. There are currently two specifications that are being followed are TPM 1.2 and 2.0. However, there is a shift from TPM 1.2 to TPM 2.0 due to the advanced features that TPM 2.0 provides. An overview of the differences between the two standards is given in Table 1.

TABLE I. *Comparison between TPM 1.2 and TPM 2.0*

Algorithm	***RSA 1024***	***RSA 2048***	***ECC NIST P256***	***ECC BNP256***	***AES 128***	***AES 256***	***SHA-1***	***SHA-2***
TPM 1.2	Yes	Yes	No	No	Optional	Optional	Yes	No
TPM 2.0	Optional	Yes	Yes	Yes	Yes	Optional	Yes	Yes

Trusted Module (TM) supports provision for implementing measurable boot using Platform Configuration Registers (PCR). PCRs are registers that hold cumulative hash values. These registers are populated using TPM_PCR_Extend and the data streaming enabled TPM_HASH structures. 256 bits of data is hashed using either SHA-1 or SHA-2 hashing algorithm on the TPM. Once the process is completed, the computed SHA value is added to an existing selected PCR value. Equation 1 shows the process of PCR extension.

$$SHA_{new} = SHA_{old} \parallel SHA_{Computed} \quad (1)$$

The boot process can be divided into stages, e.g. firmware, operating system, applications, etc. Figure. 1 shows the different stages in the secure boot software chain of trust. For measuring the boot process, the PCR is computed and verified for the next layer in the process before the execution can be passed to that layer. At the end of the process, the value of the PCR provides sequential attestation of all the components in the chain.

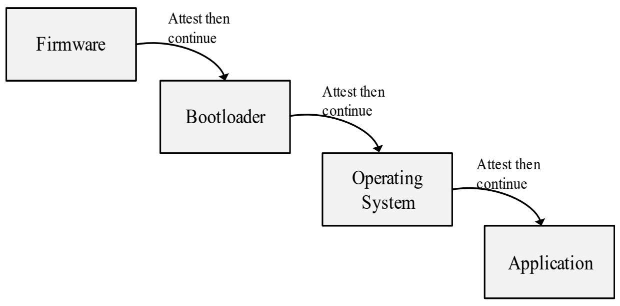


Figure 1. Secure Boot Software Chain of Trust

The secure boot is demonstrated on FPGA prototype of RISC-V and the performance evaluation is discussed.

C. *Information Flow Tracking*

There are various hardware and software-based defense mechanisms implemented on IFT techniques. It can be implemented on different types of architecture along with heterogeneous SoCs and hardware accelerators. Dynamic Information Flow Tracking (DIFT) is a hardware technique which protects the programs from malicious software attacks, but these implementations result in performance overhead by using additional physical memory to store the tag bits [8]. The authors extended the RI5CY/PULPino to develop a prototype for IoT applications. Tainted pointer technique is used to protect control and non-control data inputs resulting in a non-zero false positive rate with some synthetic false scenarios to mitigate memory attacks, but the overall performance overhead is high [10]. Code flow integrity and various defensive techniques for non-control data attacks are analyzed to develop a realistic and generic security technique for memory attacks[9][13].

Gate Level Information Flow Tracking (GLIFT) is another technique at gate level implementation which uses shadow logic to track the information flow of gates and marks the untrusted values [11]. This paper integrates the architecture level support for implementing the Information Flow

Tracking on a RISC-V processor to detect and reduce the impacts of various run-time attacks. This scheme protects bare-metal applications against memory corruption by adding tag bits to the untrusted data inputs. Figure 2 shows the register file, data memory and tag cache for storing the one-bit tags.

A separate area in the memory is dedicated to the tags which result in two separate accesses to the memory, one for the actual word and the other for the tag, that is done in parallel to not impact the performance. Additional instructions are added to read and write tags in the memory. To support the tagged memory the data cache is revised to store the tags alongside data. Tag functional units are added in the Rocket core pipeline to support the features. Fine-grained memory protection is achieved by using tag bits and tagged memory to mitigate buffer overflow attacks and memory corruption attacks.

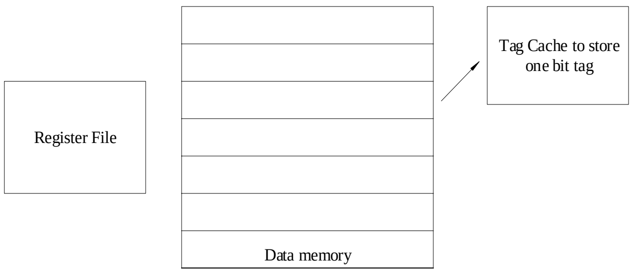


Figure 2. Separate Tag Cache with register and data memory

III. THREAT MODEL

SRAM Based FPGAs consist of a volatile memory which holds the configuration for all logic components of the Programmable Logic (PL) block of an FPGA. The bitstream is loaded into the configuration SRAM as a bootup process using a First Stage Boot Loader (FSBL) [2]. This method requires bitstream to be stored in a non-volatile storage medium, such as a flash, SD-card, etc. Whereas, during runtime, the bitstream can be loaded using PL fabric interfaces such as Internal Configuration Access Port (ICAP) [3] or Programmable Configuration Access Port (PCAP). Additionally, Dynamic Partial Reconfiguration can also be used to reconfigure a target area on the fabric during runtime. Dynamic Partial Reconfiguration allows modifying the configuration while the system is active.

FPGAs face the threat of malicious modification of bitstreams. An adversary can modify the bitstream to be configured during bootup. This attack can be achieved by modifying the source bitstream either locally or remotely using remote update mechanisms. This forces the tainted bitstream to be loaded the next time the FPGA is booted up or during a complete configuration reload. An adversary has the option to target the PL fabric during runtime. ICAP and PCAP allow readback as well as reconfiguration during runtime. An adversary can perform runtime modification attacks using either of the ports. As such, the modifications performed will last either till the next boot-up or the instant the FPGA reconfiguration is triggered. Thus a self-authentication mechanism is required for FPGA based designs.

Once the system boots with self-authentication, the application code is prone to run time attacks. The malicious or untrusted data inputs can corrupt the return address, or the payload can inject codes to run different functions and fault can be injected. Additionally, leakage of information is hard to detect. Hence the information flow tracking technique is investigated to mitigate malicious attacks and can be further extended to detect trojan activations with minimal performance overhead. In this paper, we assume that the application code is vulnerable to the attackers to perform read/write operation, inject codes or leakage of information which causes memory corruption. In order to protect the integrity of the system, a new fine-grained identifier called as *tags* are attached to the spurious data inputs and these tags are propagated from the input operand to the output operand on an instruction. A straightforward, new fine-grained one-bit tag based IFT with minimum overhead has been implemented with RISC-V as the test bed.

IV. RUNTIME SELF AUTHENTICATION FOR RISC-V ON FPGAS

This work presents a self-authentication mechanism for FPGA based designs. In this paper, we demonstrate RISC-V FPGA implementation. The secure boot scheme consists of verifier and prover entities. The prover is the untrusted entity, which must prove its authenticity to the verifier. Whereas, the verifier is a trusted entity that sends a challenge to the prover. The prover responds to the challenges and if the prover cannot satisfy the verifier with its response, the verifier marks the prover system as compromised.

In different kinds of literature, the verifier is implemented either as a function of the programmable logic (PL) fabric or occupies a part of the fabric [3][4]. This approach has two limitations. PL Fabric based authentication frameworks require additional fabric for implementing cryptographic functions. Additionally, the implemented cryptographic functions use dedicated fabric-based access-controlled memory elements. Using configuration readback mechanisms, such as ICAP and PCAP, an adversary may be able to retrieve secret keys being used. This work instead offloads security to externally established cryptosystems and trusted modules (TM) such as the TPM.

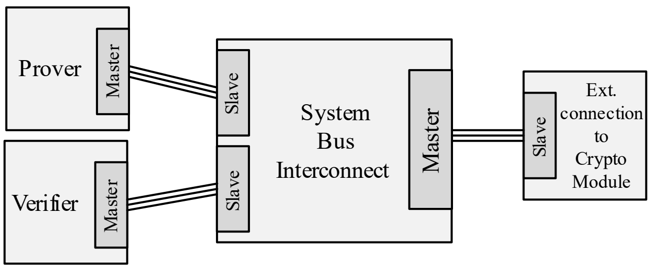


Figure 3. Secure boot attestation framework

The presented self-authentication design assumes that the verifier consists of a processor-based design implemented on an SoC that has trusted processing system and untrusted bitstreams programs the PL fabric. In order to mitigate the chances for spoofing attacks, this design assumes that PL fabric only uses on-chip Block RAM components and not any external memory components, e.g. DDR RAM. This limitation is practical realizing resource-constrained embedded system devices. The verifier can either be an off-chip processor or a hard-core processor sharing the system bus. The secure and trusted module (TM) is shared between the prover and the verifier on a shared bus interconnect or a secure network. Network security is not in the scope of this paper. The verifier-interconnect connection must not be visible to the verifier to mitigate eavesdropping. We assume that the verifier is secure and uses code stored on an isolated non-volatile Read Only Memory (ROM). The application bitstream is stored in a separate persistent storage medium. This bitstream is expected to be updated in case of an update. The verifier relates to a secure back-end update server through a secure network connection. An illustration of the design is given in Figure 3.

At boot-up, the verifier is booted up first. The verifier after initializing itself performs an initial attestation of the RISC-V design bitstream. The integrated/interfaced TM device is initialized by the verifier. In order to initialize the TM, for the demonstration we have integrated TPM with the verifier, the verifier performs the following sequence of actions:

- The TPM is sent a TPM_STARTUP structure. This command initializes the TPM. The drivers for the TPM are written to be able to access the security co-processor at the time of first stage boot loader is loaded.
- The verifier requests the backend for the expected SHA value of the bitstream. At each request, the backend computes a new SHA value. In order to ensure freshness, the SHA computation is initialized with a nonce. The computed cumulative SHA and the nonce is sent to the verifier.
- A TPM has five localities numbered 0 through 4. Each locality can use different functions and has access to separate memory locations on the TPM[5]. Locality 4 is used to computing Dynamic Root for Trusted measurement (DRTM) using PCRs. The verifier using locality 4, first sends TPM_HASH_START structure to the TPM. This readies the TPM to accept streaming data packets. The verifier now first pushes the nonce provided by the backend server into the TPM's TPM_HASH_DATA FIFO. The verifier now continues to push bitstream packets into the TPM. In this process, the bit locations where any memory element is expected is masked. This is done since the state of the memory cannot be guaranteed during execution.
- Once the bitstream has been pushed, TPM_HASH_END structure is sent to the verifier to confirm the end of data. PCR 17 on the TPM is populated with SHA value.
- The verifier compares the computed hash value with the SHA value sent by the back end. If the value is different, the verifier first notifies the backend of the change and then requests a new copy of the bitstream.
- When the bootup attestation is completed successfully, the verifier copies the bitstream into the PL fabric's configuration SRAM.
- The value of the PCR is discarded, and the PL fabric is brought up. Once this process is complete, the soft-processor system on the PL fabric becomes operational.

Once the PL fabric setup process completes the soft-processor initiates its boot up sequence and resumes application execution normally.

V. SECURING RISC-V WITH IFT

The RISC-V Rocket chip SoC contains a rocket Custom coprocessor Interface (RoCC) which provides communication between the rocket processor and attached co-processors. Most of these coprocessors are crypto units and vector processing units. The RISC-V processor is modified to incorporate a security feature which can be used to protect the device against run time attacks like buffer overflows and format strings along with self-authentication mechanism. In this implementation, we have also integrated the AES (Advanced Encryption Standard) in order to protect the sensitive data against any tampering or modifications and in order to provide data integrity and confidentiality supported at the architecture level. The AES crypto engine is a symmetric key algorithm where the same key is used to encrypt and decrypt the data. It has a fixed block size of 128 bits and a variable key size of 128, 192 and 256 bits. The key size of the AES engine specifies the number of rounds needed to transform the plain text to ciphertext or vice versa. As the key sizes used in AES are sufficiently large it can be used to protect classified information up to the top level. The AES integrated in RISC-V supports all three key sizes (128, 192, 256).

The IFT technique includes three main Tag management mechanisms namely Tag initialization, Tag propagation, and Tag check/update. In Tag initialization, tags are added to the incoming sensitive input data to mark it as spurious or authentic. Tag propagation follows a set of security policies where the tag propagates to the defined classes in which the authenticity of the operand is tested on the basis of the type of instructions being executed. The processor checks whether the spurious data is used in an unsafe manner in Tag check/ update mechanism resulting in a security exception.

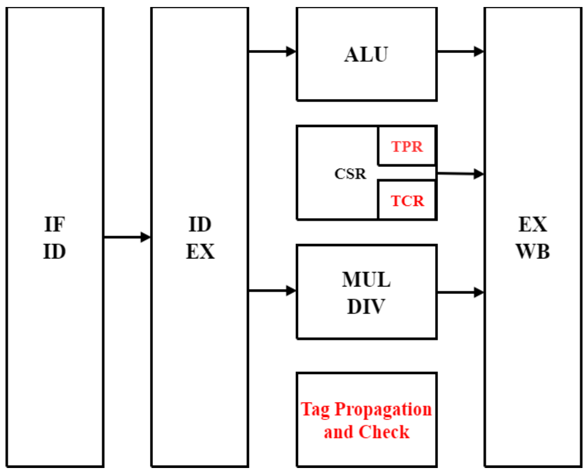


Figure 4. Modified execution stage of the processor pipeline

The IFT technique is implemented on the (SiFive) FE310 SoC which contains the E31 RISC-V core. This is done by modifying the execution phase of the E31 core. The E31 core is fully compliant with the RISC-V ISA specification. It is a high-performance implementation of RISC-V RV32IMAC architecture. Figure 4 illustrates the Modified Execution stage of the processor pipeline in RISC-V architecture. The RISC-V architecture consists of a collection of important Control and Status Registers (CSRs) to manage and access the system functionalities. Tagged memory has the ability to provide metadata, in the form of one-bit tags with each memory location [12]. The on-chip cache is extended to hold tags by adding a cache tag. To protect the data from memory corruption a one-bit tag is attached to the input data where the data is physically stored in the memory and the one-bit tag is stored in a tag cache.

Whenever the operands of an instruction are fetched, a 1-bit tag value is assigned to each operand depending on

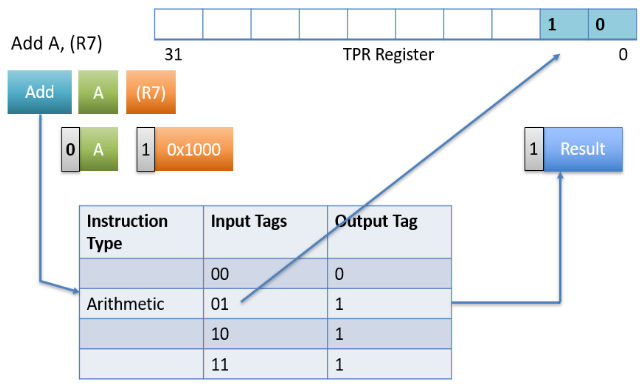


Figure 5. Tag Propagation Flow

whether they are coming from malicious I/O communication channels(1) or from a register file(0) present inside the processor. The propagation and check rules for the tags are stored in the Tag Propagation Register(TPR) and Tag Check Register(TCR) which are added in the CSRs on RISC-V. Figure 5 shows the Tag Propagation mechanism in which after assigning the tag bits, the type of instruction is checked: Arithmetic, Branch, Comparison, Logical, Shift, Jump, Branch. Depending on the type of instruction and the input operand tag value, the output tag bit value is determined. Different classes of instructions have been added to state the rules for tag propagation. When an instruction is identified the tag propagation rule sets the mode with respect to that class and under each mode, different conditions are specified for the tags to propagate from the input operand to the output operand.

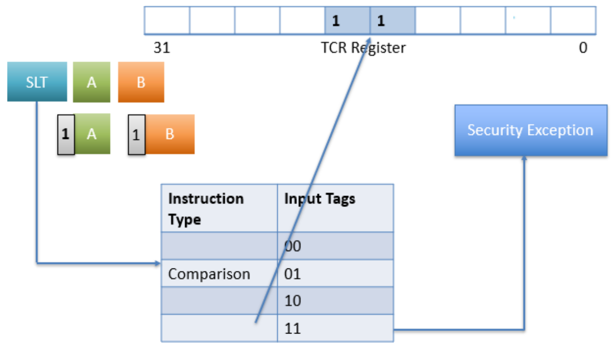


Figure 6. Tag Check Flow

The Tag Check rule checks for the source and destination operand tags and raises an exception when the condition is met. It restricts the number of operations which can be performed on spurious data. Figure 6 shows the Tag Check flow, in which the instruction type is checked for comparison or load/store. If it belongs to any of the two instruction types and if both the input operands tags of these instructions are marked as spurious, the Tag Check does not allow this instruction to be executed and raises a security exception. To assign the tag bits to the tag memory two new instructions have been added to the RISC-V architecture, namely *load_tag* and *store_tag*. The memory location is augmented with a one-bit tag that can be accessed in parallel with the data. Tagged memory is achieved by maintaining a shadow memory where the load and stores are implemented with additional codes.

VI. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The design is based on the lowRISC RISC-V FPGA implementation [6]. The lowRISC SoC is implemented on a Xilinx Zed board which uses a Xilinx Zynq 7000 xc7020clg484-1 FPGA. The FPGA is equipped with a hard dual-core ARM Cortex A9 (PS) processor. In the base implementation, the RISC-V processor is implemented on the PL fabric and has access to a serial console and a Serial Peripheral Interface (SPI) that are also implemented on the PL Fabric, and are a part of the bitstream. Zed board is equipped with an external DDR RAM which is directly connected to the PS. In order to access the RAM, the RISC-V processor instantiates the hard processor as AXI slave. The requests for the RAM are forwarded to the attached module by the hard processor. In this work, since we have assumed that FPGA implementation will not use any external memory module, no requests are made for it in the reference implementation code.

For physical Trusted Module (TM) integration TPM is interfaced with the setup, the hardware-based SPI port is configured on the PS. The TPM is interfaced using dedicated Multiplexed IO (MIO) port. This enables the TPM device to be available before the PL is programmed with a bitstream. This work uses the Infineon SLB 9670 TPM 2.0 module [7]. A custom driver is written for the PS and the RISC-V processor to access the TPM and to compute the hash of the bitstream. Additionally, a PCAP component is added to the RISC-V design to perform runtime attestation. The PS core also uses the onboard Ethernet port to communicate with the

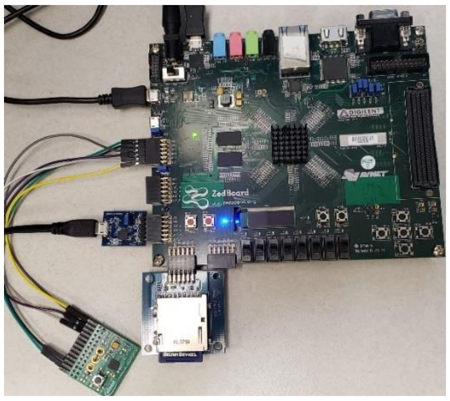
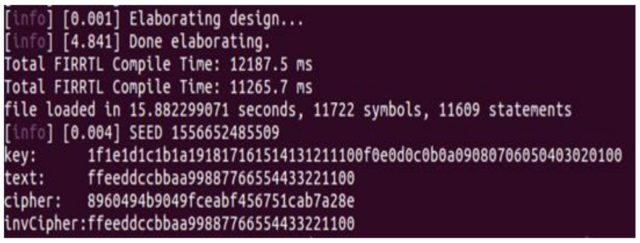


Figure 7. Experimental Test Bed

backend server. The setup of our implementation is shown in Figure 7.

The architectural level of the E31 core is modified for IFT and two registers are added in the CSR. Two new instructions are added and the RISC-V toolchain is extended. The TPR and TCR registers contain the rules for the tag propagation and the tag check respectively. Each element of the class indicates a field in the TPR and TCR register and depending on the rules the status of these registers is updated. Whenever an exception is raised, the offending instruction is not committed, and an exit routine is executed instead. In this implementation, the modifications are made in the execution stage of the processor pipeline. In order to incorporate the entire Information Flow

Tracking technique, these modifications will be integrated into the IF, ID and the WB stage of the pipeline. These structures can enhance security capability to mitigate the propagation of attacks during execution. AES crypto engine is integrated into the RISC-V processor. Figure 8 shows the results of the AES crypto engine integrated with the RISC-V core.



```
[info] [0.001] Elaborating design...
[info] [4.841] Done elaborating.
Total FIRRTL Compile Time: 12187.5 ms
Total FIRRTL Compile Time: 11265.7 ms
file loaded in 15.882299071 seconds, 11722 symbols, 11609 statements
[info] [0.004] SEED 1556652485509
key:      1f1e1d1c1b1a191817161514131211100f0e0d0c0b0a09080706050403020100
text:     ffeeddccbbaa99887766554433221100
cipher:   8960494b9049fceabf456751cab7a28e
invCipher:ffeeddccbbaa99887766554433221100
```

Figure 8. AES integrated in RISC-V Core

Control and data flow logic is customized in the RISC-V for the classes in the TPR and TCR rules and values are set according to the input tag operands and type of instruction. At the same time, the TPR register bits are also set according to the input operands tags for a class. The modified code flash programmed into the FPGA board.

VII. Performance Analysis

The overhead of the interface to the hardware includes additional rule check during the bitstream load process, that is integrated at the FSBL. The design requires custom drives to interface the Trusted Module and requires an addition of a PL based PCAP interface IP. The bitstream size for XC7020 is around 3.4 MB. On average, the total time to compute the hash for a bitstream was measured to be approximately 32 seconds. The area overhead of the information flow tracking is the integration of tag cache and tag registers, inputs, and outputs. Tag propagation register and tag check register and state machine to update these registers.

VIII. Acknowledgements

The research is funded by NSF awards 1814420, 1819687 and 1819694

IX. Conclusion

Considering the threat model and the presented architecture, the security solution put forward by our proposed architecture is realistic and has negligible performance overhead. The paper provides a solution for securing RISC-V processors by implementing a self-authenticated secure boot during startup and providing information flow tracking to detect and stop run-time memory corruption attacks. By using shared memory for IFT technique the performance overhead is considered negligible.

References

[1] "TPM Library Specification." [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_library_specification. [Accessed: 01-Feb-2018].

[2] Xilinx Inc., "Zynq 7000 SoC Technical Reference Manual," 2018. [Online].Available:https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. [Accessed: 13-Jul-2018].

[3] D. Owen Jr. et al., "An Autonomous, Self-Authenticating, and Self-Contained Secure Boot Process for Field-Programmable Gate Arrays," Cryptography, vol. 2, no. 3, p. 15, Jul. 2018.

[4] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, "SACHa: Self-Attestation of Configurable Hardware," in 2019 Design, Automation &amp; Test in Europe Conference &amp; Exhibition (DATE), 2019, pp. 746–751.

[5] Trusted Computing Group, "TCG PC Client Platform TPM Profile (PTP) Specification Family '2.0' TCG Public Review," 2017.

[6] "lowRISC·lowRISC."[Online].Available:https://www.lowrisc.org/. [Accessed: 31-May-2019].

[7] "SLB 9670VQ2.0 - Infineon Technologies." [Online]. Available:https://www.infineon.com/cms/en/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/slb-9670vq2.0/. [Accessed: 03-Nov-2018].

[8] Christian Palmiero, Giuseppe Di Guglielmo, Luciano Lavagno, Luca P.Carloni, "Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V core for IoT applications", 2018 IEEE.

[9] Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar and Ravishankar K. Iyer. "Non-control-data attacks are realistic threats". In In USENIX Security Symposium

[10] Shuo Chen, Jun Xu, Nithin Nakka, Zbigniew Kalbarczyk, Ravishankar K.Iyer, "Defeating memory corruption attacks via pointer taintedness detection. 2005 DSN

[11] M.Tiwari, X. Li, H. M. G. Wassel, B. Mazloom, S. Mysore, F. T. Chong and T. Sherwood. "Gate level Information Flow Tracking for Secure Architectures". IEEE Micro, 2010

[12] Chengyu Song, Hyungon Moon, Monjur Alam, Insu Yun, Byoungyoung Lee, Taesoo Kim, Wenke Lee and Yunheung Paek, "HDFI: hardware-assisted data flow isolation. 2016 IEEE Symposium.

[13] Armaiti Ardeshiricham, Wei Hu, Joshua Marxen, Ryan Kastner, "Register Transfer Level information flow tracking for provably secure hardware design", 2017 IEEE

[14] RISC-V Instruction Set Manual. Volume II: Privileged Architecture

[15] RISC-V Instruction Set Manual. Volume I: User-Level ISA