Multilayer Camouflaged Secure Boot for SoCs

Conference Paper · December 2019

DOI: 10.1109/MTV48867.2019.00019

7 authors, including:

Ali Shuja Siddiqui
Cisco Systems, Inc
24 PUBLICATIONS **141** CITATIONS
SEE PROFILE

Geraldine shirley Nicholas
University of North Carolina at Charlotte
8 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Yutian Gui
University of North Carolina at Charlotte
19 PUBLICATIONS **71** CITATIONS
SEE PROFILE

J. Plusquellic
University of New Mexico
148 PUBLICATIONS **3,124** CITATIONS
SEE PROFILE

Multilayer camouflaged secure boot for SoCs

Ali Shuja Siddiqui, Geraldine Shirley Nicholas, Sam Reji Joseph, Yutian Gui, Jim Plusquellic*, Marten Van Dijk^, and Fareena Saqib,
Dept. of Electrical and Computer Engineering, University of North Carolina at Charlotte, Charlotte, North Carolina
* Dept. of Electrical and Computer Engineering, University of New Mexico, Albuquerque, New Mexico
^ Dept. of Electrical and Computer Engineering, University of Connecticut, Storrs, Connecticut
{asiddiq, gnichola, sjoseph8, ygui}@uncc.edu, marten.van_dijk@uconn.edu, jimp@ece.unm.edu and fasqib@uncc.edu

***Abstract*— Reconfigurable logic enables architectural updates for embedded devices by providing the ability to reprogram partial or entire device. However, this flexibility can be leveraged by the adversary to compromise the device boot process by modifying the bitstream or the boot process with physical or remote access of device placed in a remote field. We propose a novel multilayer secure boot mechanism for SoCs with a two-stage secure boot process. First stage uses device bound unique response as a key to decrypt application logic. The security function is extended at runtime by integrating intermittent architecture and application locking mechanism to reveal correct functionality.**

Keywords*— *Secure boot, Physical Unclonable Function, FPGA

I. Introduction

The capability of hardware reconfigurability in FPGAs can patch the hardware bugs in the IoT device with the ability to adapt the architectural needs with the changing requirements. The reconfigurable fabric is used by vendors to update the architecture on-the-fly with the changing requirements of the application. The reconfigurability makes a device adapt to new requirements over time, it also opens a new door for malicious physical and remote attacks.

In SRAM FPGAs, the configuration for the Programmable Logic (PL) fabric is performed by a bitstream. A bitstream consists of initialization values for all logic components, memory components and the interconnects that exist on the fabric. The bitstream is loaded onto the fabric at every boot-up. If an attacker has physical access to the device, it can copy the bitstream from the device onto another device of the same kind, thus performing IP cloning[1]. An attacker can also induce hardware trojan[2]. Being connected to the network creates possibilities for remote attacks, such as man-in-the-middle (MitM), and eavesdropping attacks that can lead to IP cloning. Additionally, remote attacks can also lead to spoofing attacks, where an attacker can pose as a backend server which can send a compromised bitstream to the device. To counter these threats, the secure boot process should be able to mutually authenticate the remote server and secure the boot process.

We propose a novel secure boot and runtime logic locking process for mitigating FPGA bitstream piracy. The bitstream is encrypted with a key that is only available to the device enrolled with the trusted content provider. The key is not stored on a non-volatile memory and is generated on fly using a Physical Unclonable Function (PUF). In this scheme, second stage of the boot process integrates logic locking, and logic camouflaging scheme to add runtime security to the bitstream. The scheme provides mutual authentication between the field device and the trusted server. The paper is organized in the following five sections. Section II describes the existing work; section III presents the proposed architecture. Section IV discusses the experimental setup and section V presents the conclusion.

This work is funded by NSF grant numbers: 1819687, 1819694 and 1814420

II. Background Studies

A. Secure Boot for FPGAs

Bitstream encryption is a common security mechanism provided by FPGA vendors[3][4]. Encrypted bitstream is stored on the storage medium. FPGAs have on-board eFuses or battery backed RAM (BBRAM) to hold encryption keys for decrypting the bitstream that is to be loaded onto the board. The limitations of this approach of key storage needs are expensive and in low power IoTs, it's not feasible as it requires constant power supply to hold the key values. On the other hand, eFuses are one time programmable and if in by anyway the key is leaked, the encryption is defeated, and there is no way to update the key.

The implementation for Secure Boot does not extend beyond the First Stage Boot Loader and the bitstream. [5] shows the second stage boot loader can be targeted to load any malicious code secured by this process. Additionally, the encryption block and other related security blocks are proprietary and therefore cannot be verified. Furthermore, non-invasive attacks have been proven to be effective against the memory elements used to store the secret keys[6][7]. Latest technology FPGAs [8] provide added security features, such as the use of Physical Unclonable Functions (PUF) to encrypt symmetric keys. However, the security comes at a premium

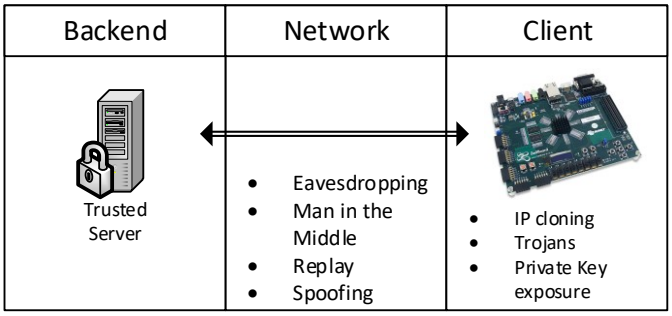

Fig. 1. Attack vectors for an in-field FPGA device.

cost, as these devices cost above thousands of dollars. There are other solutions which make use of self-authentication techniques[9][10]. The target bitstream consists of a static block and a reconfigurable block. The static block consists of a PUF and an encryption block. The response of the PUF block generates the encryption key for the reconfigurable partition. It limits portability of the reconfigurable block. The limitation of

giving resources to the security block has an area overhead attached to it, thus is limiting for resource constrained applications. Additionally, the private symmetric key exists on the same fabric as the application logic at the same time. This can lead to private key exposure to the application logic, as well as to any remote attacker having access to the device.

An alternate approach is used by[11], where a backend server is involved in the boot process. In this solution, the fabric consists of a similar configuration to [9], whereas the PUF challenge input is provided by the backend server. Once the server can attest the board by verifying the PUF output, the server pushes the entire bitstream for the reconfigurable partition that consists of the application logic. There is an area overhead associated with this approach. Additionally, this approach requires a higher bring up time for the board as the entire partition is transferred over the network.

Keeping in mind the state of security for reconfigurable IoT devices and its limitations, this paper proposes an architecture for providing secure boot and secure over-the-air-updates. This research focusses on providing secure boot mechanisms for FPGA bitstreams. Existing works divide the logic fabric into two, one for holding application logic and the other for holding security functions. This approach adds overhead on the fabric that takes away space from the application logic. In this work, we propose an architecture that mitigates the security overhead away from the application logic by proposing a two-stage secure boot mechanism that as its first stage uses device bound unique response as a key to decrypt application logic. The security functionality is extended at runtime by adding logic locking mechanisms in the application logic. To establish trust between the IoT device and the backend, the scheme implements a mutual authentication scheme.

B. Logic Locking

Logic Locking is a design for trust technique to lock the netlist by inserting key gates in the original design, such that the circuit functions correctly with the correct keys and when a different key is given to the circuit it results in a corrupted output[12]. The functionality of the circuit is hidden, and the key can be stored in the tamper proof memory to prevent access by attackers. Thus, an encrypted bitstream can be generated from the locked netlist. The application bitstream will be locked and only with run-time authentication, the bitstream can be unlocked by using the correct key. The application produces corrupted outputs if the adversary tries to access or modify the encrypted bitstream.

Stage two of multilayer secure boot consists of runtime authentication of the device, where the application logic is locked and verified by the device holding the key. Locking the functionality and generating the locked bitstream makes it difficult and resilient to physical attacks by an adversary. Different algorithms have been developed to decide which best location is suited to insert the XOR gates [13]. Logic locking techniques such as Random Logic Locking(RLL), Fault analysis-based Logic Locking(FLL), Strong Logic Locking (SLL), etc are vulnerable to SAT attack[14]. SAT attack is a key pruning attack that breaks combinational logic locking technique. It uses SAT solvers to search and compute DIPs on the locked netlist and eliminates the incorrect key. Anti-SAT countermeasure exponentially increases the total execution time of SAT iterations making it infeasible for SAT attack but is vulnerable to signal probability skew attack[15].

SARLock (SAT Attack Resistant Logic Locking) corrupts the output for an incorrect key and is resilient to SAT attack [16] where it maximizes the required number of distinguishing input patterns to recover the secret key. This technique uses a comparator and a mask block to protect the original circuit. The comparator generates a flip signal that is asserted for specific input and key combinations, and the flip signal is XORed with one of the primary outputs. Mask logic is used to prevent the flip signal from being asserted for the correct key. Hence, for a correct key, no error is injected in the circuit and for an incorrect key, an error is injected into the circuit leading to an incorrect output. Security analysis of SARLock implementation shows the one-point function (one-bit flip) can be provably obfuscated, thus providing the adversary no advantage beyond having a black-box access to the implemented netlist[17]. Therefore, the proposed scheme locks the encrypted application bitstream using SARLock technique.

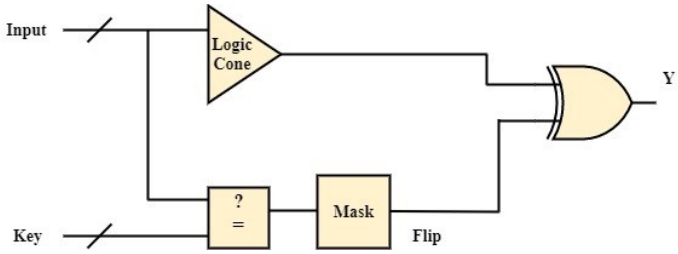

Fig. 2. Logic Locking using SARLock

In the proposed scheme, the original application netlist is locked by the SARLock technique to hide the functionality of the circuit which is shown in Figure 2. SARLock thwarts the SAT attack where a single bit flip can drastically corrupt the output bits resulting in a small Hamming distance(HD) for the incorrect keys. The critical logic cones are protected, and the netlist is resilient to reverse engineering.

III. PROPOSED SECURE BOOT SCHEME

The secure boot scheme consists of trusted first stage boot loader, and remote server for sharing the correct configuration of application bitstream referred as content provider. The backend content provider server is secure and is in trusted environment. The server authenticates and shares the correct logic implementation to legitimate client FPGA devices through mutual authentication. For every update occurring on the system, it provides a client device with an update bitstream. Each client has two kinds of non-volatile memories, a Read Only Memory (ROM) holds trusted pieces of code and design, and a separate application NVM (ANVM) to store all application logic and data. ANVM can be any form of mass storage, e.g. flash memory, SD card, QSPI memory, etc.

In the proposed scheme, there are two bitstreams, namely Attestation Bitstream (AUB) and the Application Bitstream (APB). Attestation bitstream AUB consists of the embedded PUF logic that is used for unique bitstream generation to attest a device on the field. APB consists of encrypted logic locked

and LUT camouflaged version of the application bitstream. There are multiple stages in this framework, device enrollment, attestation, logic unlocking and LUT configuration for selected frames. The secure boot process during enrollment and infield is shown in Figures 3a and 3b respectively, and is discussed in the later sections.

Device enrollment is done in a trusted environment to ensure

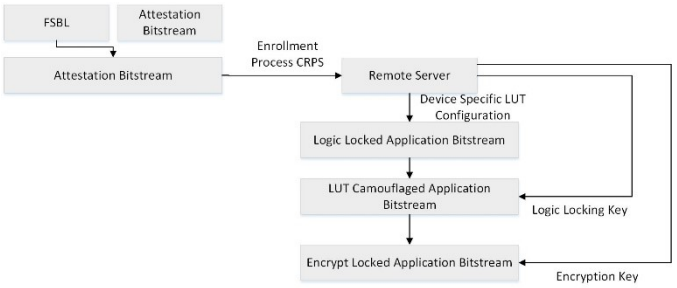

Fig. 3a. FPGA Secure Boot Enrollment Process.

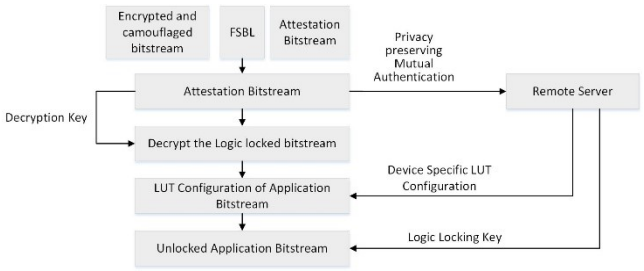

Fig. 3b. FPGA Secure Boot in field Process.

only legitimate enrolled devices can receive updates that are crafted for them individually. In the first stage, Authentication Bitstream (AUB) has a PUF to generate unique per device keys to decrypt the second stage application bitstream and a unique bitstream (APB) is generated per device which consists of sliced and corrupted frames at the LUT granularity. Finally, the correct LUT configuration for the device is sent by the server after successful light-weight device authentication.

A. Device Enrollment

Before an FPGA device can be deployed in the field, it is first enrolled with the server in a trusted environment. AUB design consists of an FPGA based PUF. For this demonstration, HELPUF implementation is used for attestation bitstream that is used for generating unique per device decryption key [18].

1) PUF Enrollment Process:

HELP PUF, shown in figure 4, use the manufacturing variations in the path delays of a functional unit. The functional unit in the paper is AES engine, where the paths have complex compositions consisting of fanouts and convergent paths with complex interconnection structure. The clock phase differences are used as soft path delays, that are used for bit generation using margining and dual helper data. The details of the bit generation and mutual authentication are discussed in [21].

HELPUF is based on path delay variations. Unlike other implementations of PUF, such as Arbiter and Ring Oscillator

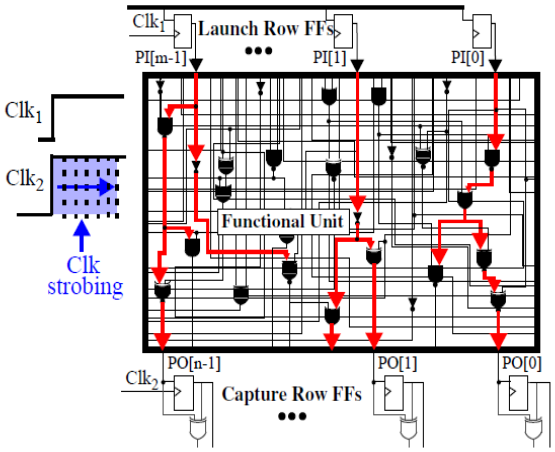

Fig 4: HELP PUF [21]

PUF designs, HELP PUF integrates itself into the existing hardware functions. In this design, HELP is integrated with AES functional unit. Input vectors are provided to the HELP controller. These vectors sensitize different paths on the chosen AES circuit and digitize the entropy of the path delays. The delays of the paths noted from the initial launch flip flop to the terminating capture flip-flop (called the Latch-Capture Interval (LCI)), provides the source of entropy to this PUF[19]. The PUF is strong and provides exponential challenge-response pairs (CRPs). This feature is used for mutual authentication, discussed in [20][21].

B. Multilayer Camouflaged Secure boot:

During enrollment, the server sends the AUB to a client and is stored on the ROM of the client node. The AUB is loaded onto the FPGA fabric during the boot process. The server stores responses from the client node for a large set of common input challenges. The server stores the set of challenge-response pairs (CRPs) in a database and are used in the attestation process. Once the AUB has been loaded on the target FPGA, the server sequentially sends the input challenges to the AUB design. Responses are gathered from the PUF and stored on the server. Server selects a unique challenge input string (c) and stores it on the device's ANVM. Challenge c is then used as input to the Physical unclonable function and generate response R_s. The response R_s is treated as a private decryption key by the device. The private key R_s is not stored on a non-volatile memory on the

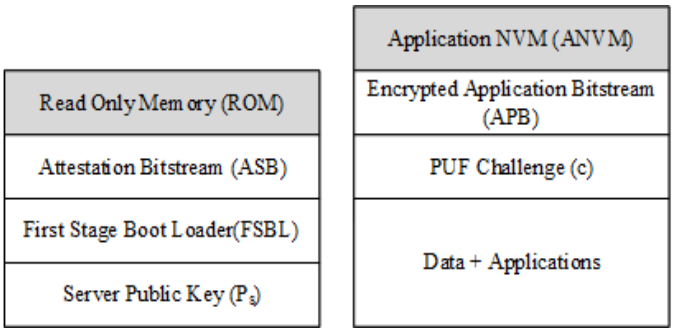

Fig 5: Client FPGA memory view after enrollment

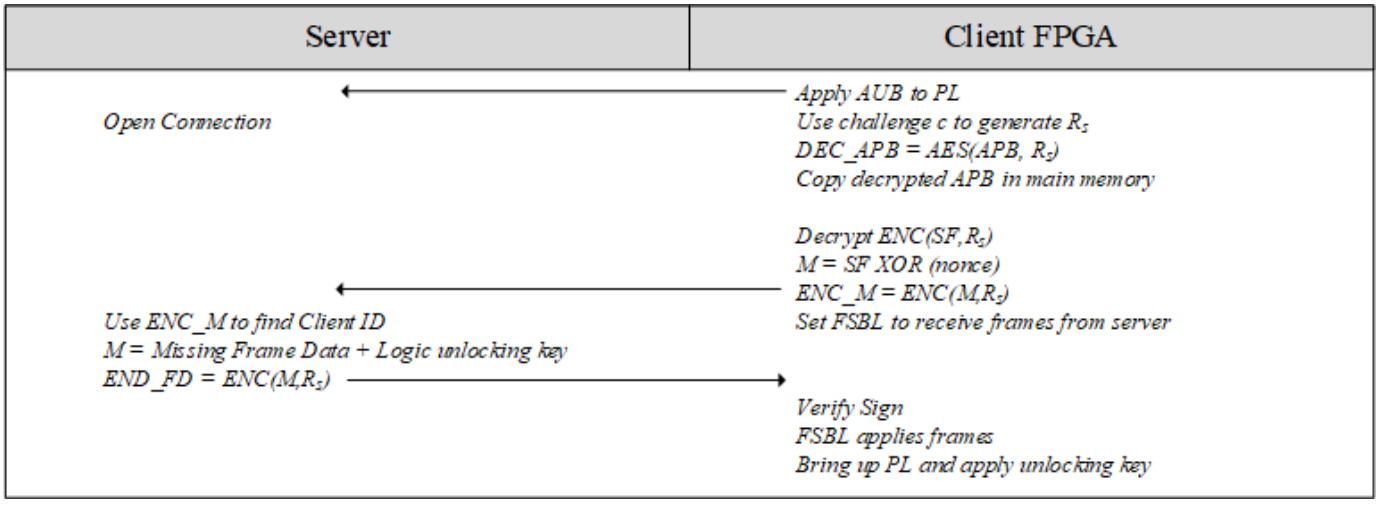

Fig. 6. Attestation and Application Bitstream programming

device throughout the lifecycle of its operation, to mitigate chances for private key exposure. The server uses R_s to encrypt APB. This encrypted APB is then stored onto the ANVM on the client FPGA device. The APB, that is logic locked and LUT camouflaged, has the associated keys stored on server in a secure database.

The First Stage Boot Loader (FSBL), along with AUB is stored on the ROM. It is used to load the bitstream onto the Programmable Logic (PL) fabric. The bitstream is loaded and the control is given to the next stage software, which can be a second stage boot loader (e.g. Uboot) or it can also be a baremetal application.

To implement mutual authentication, a device should also be able to authenticate the server. Asymmetric Public key of the server is stored on the ROM of a client device. We are using RSA based digital signatures in this system, however, any other signing scheme can also be used[22]. The applications and associated data are also stored in the ANVM, and that can be also encrypted, it is not presented in this paper. The final view of the on-board memory of a client device is given in Figure 5. After enrollment, the device can be placed in the untrusted field.

C. *Device Attestation and LUT Camouflage and Logic Unlocking*

Once the device is placed in the field, it is securely connected with the backend server. The device boots up, and the zeroth stage boot loader BootROM executes. BootROM code brings the device up and locates the FSBL code from the ROM. The FSBL code initializes the memory, peripherals connected to the device, and any onboard security features. Once the initialization is complete it copies the APB from the ROM over the main memory and then onto the Programmable Logic (PL) fabric.

The AUB consists of a PUF implementation, the PUF responses R_s are captured and copied onto the main memory. The FPGA device communicates with the server initiating attestation request. All messages originating at the server are asymmetrically signed using the private key pair. The server selects a random subset of challenges from the challenge subset and sends them to the client FPGA. Using public key Ps, the client device performs digital signature verification of all incoming traffic from the server. The client sends back the PUF responses to the server. Dual Helper Data (DHD) scheme discussed in [21], that is used to reduce bit-flip errors and for the mutual authentication using fuzzy matching

We propose a novel authentication protocol using the LUT camouflaged configuration that is unique per device. The client node and server share the list of devices specific LUTs that are modified/corrupted unique for the given device, this list is referred as sliced frame list (SF). The list is encrypted with the device PUF response R_s and is stored in a secure database on the server and stored on the client in ANVM. The client uses the stored challenge c to regenerate R_s and decrypts the list of sliced frames (SF) list. The timestamped SF list is encrypted again by R_s and is sent to the server for authentication. The client uses the PUF response R_s to decrypt APB using the onboard AES block, that is copied onto the main memory.

The server compares decrypted SF lists from the enrolled devices using the PUF responses for each client node stored in the enrolled database. The SF list and PUF response for a given challenge c is unique per device, the search process maps the encrypted SF list to a device. This process achieves mutual authentication. The server decrypts the SF, verifies the timestamp and compares the SF list with the SF list stored in the secure database is the same. After the client identification, the server sends the associated key bits and LUT configuration encrypted with R_s. The client decrypts the encrypted message from the server and programs the APB bitstream with the correct LUT configuration. The decrypted and correctly LUT configured APB is programmed onto the PL fabric. As a part of the cleanup process, the value of R_s, the keys and decrypted APB values are removed from the main memory. The scheme is illustrated in Fig. 6.

1) *LUT Configuration*

The decrypted bitstream is secured by camouflaged LUT level corrupted interconnections and frame configuration. These

LUT configurations are modifications that are unique per device. The Device Configuration Port (devcfg) is used to configure the corrupted/modified frames. A complete frame address list for the programmable logic PL is generated for the given bitstream, to target exact frame addresses. In each frame write request, an extra dummy frame is appended at the end of the request. Once all the frames have been committed to PL memory, the device is brought up again. Figure 7 shows the steps of LUT frames configuration updates on the programmable logic using devfg.

```
//Writing frame to PL
CmdIndex = 0;
CmdBuf[CmdIndex++] = 0xFFFFFFFF;    // Dummy Word
CmdBuf[CmdIndex++] = 0xAA995566;    // Sync Word
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x30018001;    //Write IDCODE
CmdBuf[CmdIndex++] = 0x03727093;    //Zedboard ID
CmdBuf[CmdIndex++] = 0x30002001;    // Type 1 Write 1 Word to FAR
CmdBuf[CmdIndex++] = 0x0000239b;    // FAR Address
CmdBuf[CmdIndex++] = 0x30008001;    // Type 1 Write 1 to CMD
CmdBuf[CmdIndex++] = 0x00000001;    // WCFG Command
CmdBuf[CmdIndex++] = 0x30004000;    // Write FDRI
CmdBuf[CmdIndex++] = 0x50000000 | 202; //Write 202 words
for (int count =0;count <202;count++){
    CmdBuf[CmdIndex++] = frame_update[count];
}
CmdBuf[CmdIndex++] = 0x30008001;    //Type 1 Write 1 Word to CMD
CmdBuf[CmdIndex++] = 0x0000000D;    //DESYNC Command
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0

XDcfg_InitiateDma(DcfgInstancePtr, (u32)(&CmdBuf[0]),
            XDCFG_DMA_INVALID_ADDRESS, CmdIndex, 0);

/* Poll IXR_DMA_DONE */
IntrStsReg = XDcfg_IntrGetStatus(DcfgInstancePtr);
while ((IntrStsReg & XDCFG_IXR_DMA_DONE_MASK) !=
        XDCFG_IXR_DMA_DONE_MASK) {
    IntrStsReg = XDcfg_IntrGetStatus(DcfgInstancePtr);
}

/* Poll IXR_D_P_DONE */
while ((IntrStsReg & XDCFG_IXR_D_P_DONE_MASK) !=
        XDCFG_IXR_D_P_DONE_MASK) {
    IntrStsReg = XDcfg_IntrGetStatus(DcfgInstancePtr);
```

Fig 7: Writing a frame addresses 0x0000239b

2) Runtime Locking

The next step after LUT configuration is the APB locking mechanism which reveals the correct functionality of the application. The application bitstream APB is programmed onto the PL is logic locked using SARLock to protect the original circuit from the adversary.

A logic locked bitstream is implemented on RTL synthesized to the netlist using Nangate open cell library. The python scripts read the netlist and integrated key gates and mask to lock the netlist, and the corresponding keys combination is shared with the server. The locked Verilog file is given to Vivado from Xilinx to obtain the locked application bitstream. Figure 7 shows the flow of the bitstream locking.

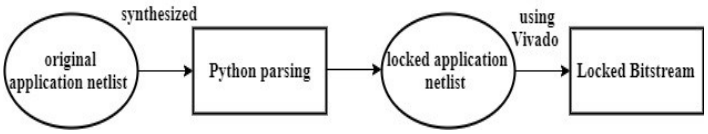

Fig. 8 Bitstream Locking

The keys generated from the locked circuit, are stored on the server. The key input is given to the logic locked application bitstream. After the device authentication is completed, the server sends the correct key to the client to successfully unlock the bitstream. If the system is physically accessed by the adversary to modify or to copy the bitstream, it produces a corrupted or wrong output which makes it unfeasible to clone the IP. Logic locked circuit provides secured bitstream protection. Multi-Level combinational circuit called Dedicated ALU from MCNC benchmark suite has been used for the demonstration of LUT configurations. It has 75 inputs and 16 outputs. The key insertion scheme SAT attack-resilient logic locking (SARLock) is used to lock the circuit. The obfuscation is done using insertion of XOR/XNOR gates. 190 Key bits are used.

The locked application bitstream is SAT resilient model and applicable for FPGA designs that extends the security of the device. During run time execution, until the correct key is provided as input from the server, the application's original functionality is unknown and difficult to break. This acts as an additional security layer between the client and the server. The programmed PL fabric is unlocked after the device attestation stage with the correct key from the server. The key combination is sent from the server to the device to unlock the APB.

IV. Experimental Results

This work has been implemented and tested on a Xilinx Zynq 7020 Zedboard. Key components for implementing security in this framework is the PUF component, the key locking mechanism implemented on the fabric, and LUT level design modification using corrupted LUT configuration. This scheme is tested using HELPUF. The HELP PUF implementation is based on the SBOX in an AES engine. The SBOXs are non-linear and complex and path delays provide large entropy.

The locked netlist and original netlist of the application circuit is tested for functionality on Vivado platform and the generated locked bitstream is flash programmed onto Zynq 7020 Zedboard. The VIO block is used to give inputs to the locked netlist. Switches act as the key inputs and the LEDs act as the outputs to test the functionality of the locked application bitstream. Experiment shows the experimental results of logic locking scheme implemented in the Zedboard with a correct 8-bit key (01101000)

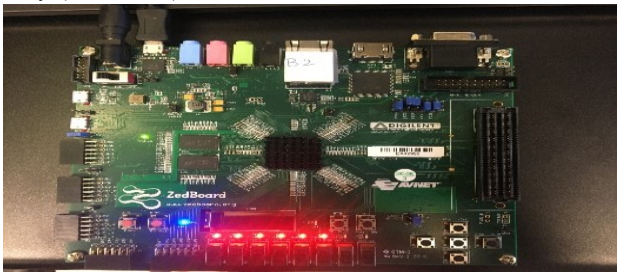

Fig. 9. Logic Locking implementation with correct key:

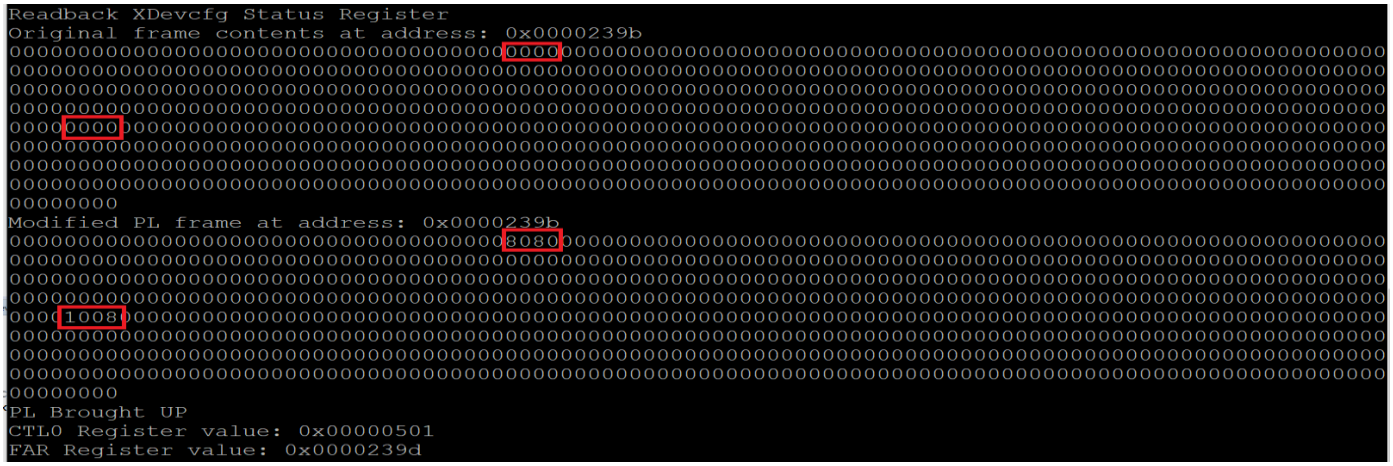

Fig. 10: LUT configuration of modified or corrupted and after verification.

The LUT configurations that are corrupted per device are unique addresses, and thus require the device identification before the correct configuration can be sent to the FPGA client node. The step of LUT configuration using devfg is shown in the figure 10, where the original frame content and the modified content for the PL is shown. This device level uniqe PL frame modification is a countermeasure for the IP cloning, as the same design will not work on any other device.

V. Conclusion

The paper proposes a multilayer secure boot process, that utilizes device level unique physical unclonable function for unlocking the design and updating the LUT frame unique per device to mitigate the security vulnerabilities of maliciously modifying the boot image of bitstream to program the programmable logic. This multilayer secure boot allows the remote attestation server to mutually authenticate, and verify the design running on the fabric with logic locking and LUT frame modification.

VI. References

[1] R. Maes, D. Schellekens, and I. Verbauwhede, “A pay-per-use licensing scheme for hardware IP cores in recent SRAM-based FPGAs,” *IEEE Trans. Inf. Forensics Secur.*, vol. 7, no. 1 PART 1, pp. 98–108, Feb. 2012.

[2] V. Jyothi and J. J. V. Rajendran, “Hardware Trojan Attacks in FPGA and protection approaches,” in *The Hardware Trojan War: Attacks, Myths, and Defenses*, Cham: Springer International Publishing, 2017, pp. 345–368.

[3] Xilinx Inc., “Zynq-7000 All Programmable SoC Secure Boot,” 2014. [Online].Available:https://www.xilinx.com/support/documentation/user_guides/ug1025-zynq-secure-boot-gsg.pdf. [Accessed: 11-Apr-2018].

[4] Intel, “Intel Arria 10 SoC Secure Boot User Guide.” [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/cru1452898171006.html. [Accessed: 07-Oct-2019].

[5] N. Jacob, J. Heyszl, A. Zankl, C. Rolfes, and G. Sigl, “How to Break Secure Boot on FPGA SoCs Through Malicious Hardware,” Springer, Cham, 2017, pp. 425–442.

[6] F. Courbon, “Challenges and examples of in-situ memory content extraction techniques,” in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2018, pp. 493–496.

[7] H. Lohrke, S. Tajik, T. Krachenfels, C. Boit, and J.-P. Seifert, “Key Extraction Using Thermal Laser Stimulation,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, pp. 573–595, Aug. 2018.

[8] Xilinx Inc., “Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices,” 2018. [Online]. Available:https://www.xilinx.com/support/documentation/application_notes/xapp1323-zynq-usp-tamper-resistant-designs.pdf. [Accessed: 11-Apr-2018].

[9] G. Pocklassery, W. Che, F. Saqib, M. Areno, and J. Plusquellic, “Self-authenticating secure boot for FPGAs,” in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2018, pp. 221–226.

[10] 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.

[11] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, “SACHa: Self-Attestation of Configurable Hardware,” in *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*, 2019, pp. 746–751.

[12] J. A. Roy, F. Koushanfar and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," *2008 Design, Automation and Test in Europe*, Munich, 2008, pp. 1069-1074.

[13] M. Yasin and O. Sinanoglu, "Evolution of logic locking," *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Abu Dhabi, 2017, pp. 1-6.

[14] Yasin M., Mazumdar B., Rajendran J., Sinanoglu O "Hardware Security and Trust: Logic Locking as a Design-for-Trust Solution" *In: Elfadel I., Ismail M. (eds) The IoT Physical Layer. Springer, Cham*, 2019.

[15] M. Yasin, B. Mazumdar, O. Sinanoglu and J. Rajendran, "Security analysis of Anti-SAT," *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Chiba, 2017, pp. 342-347.

[16] M. Yasin, B. Mazumdar, J. J. V. Rajendran and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, 2016, pp. 236-241.

[17] R. Canetti, “Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information,” in *Proc. Annual International Cryptology Conference*, 1997, pp. 455–469.

[18] J. Aarestad, P. Ortiz, D. Acharyya, and J. Plusquellic, “HELP: A Hardware-Embedded Delay PUF,” *IEEE Des. Test*, vol. 30, no. 2, pp. 17–25, Apr. 2013.

[19] W. Che *et al.*, “Analysis of Entropy in a Hardware-Embedded Delay PUF,” *Cryptography*, vol. 1, no. 1, p. 8, Jun. 2017.

[20] W. Che, F. Saqib, and J. Plusquellic, “PUF-based authentication,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 337–344.

[21] W. Che, M. Martin, G. Pocklassery, V. Kajuluri, F. Saqib, and J. Plusquellic, “A Privacy-Preserving, Mutual PUF-Based Authentication Protocol,” *Cryptography*, vol. 1, no. 1, p. 3, Nov. 2016.

[22] D. Johnson, A. Menezes, and S. Vanstone, “The Elliptic Curve Digital Signature Algorithm (ECDSA),” *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, Aug. 2001.