An Autonomous, Self-Authenticating and Self-Contained Secure Boot Process for FPGAs

D. Heeger, W. Che+, F. Saqib*, Matt Areno^ and J. Plusquellic
University of New Mexico, Enthentica+, University of North Carolina, Charlotte*, Trusted and Secure Systems^, University of New Mexico

Abstract

Secure boot within an FPGA environment is traditionally implemented using hardwired embedded cryptographic primitives and NVM-based keys, whereby an encrypted bitstream is decrypted as it is loaded from an external storage medium, e.g., Flash memory. A novel technique is proposed in this paper that self-authenticates an unencrypted FPGA configuration bitstream loaded into the FPGA during the start-up. The ICAP interface is accessed to read-out configuration information of the unencrypted bitstream, which is then used as input to SHA-3 to generate a digest. In contrast to conventional authentication where the digest is computed and compared with a second pre-computed value, we use the digest as challenges to a hardware-embedded delay PUF called HELP. The delays of the paths sensitized by the challenges are used to generate a decryption key using the HELP algorithm. The decryption key is used in the second stage of the boot process to decrypt the application. Malicious tampering with the unencrypted bitstream changes the challenges, and the corresponding decryption key, resulting in key regeneration failure.

1 Introduction

SRAM-based FPGAs need to protect the programming bitstream against reverse engineering and bitstream manipulation (tamper) attacks. Fielded systems are often the targets of attack by adversaries seeking to steal intellectual property through reverse engineering, or attempting to disrupt operational systems through the insertion of kill switches known as hardware Trojans. Internet-of-things (IoT) systems are particularly vulnerable given the resource-constrained and unsupervised nature of the environments in which they operate.

FPGAs requiring secure boot usually store an encrypted version of the programming bitstream in an off-chip nonvolatile memory (NVM) as a countermeasure to these types of attacks. Modern FPGAs provide on-chip battery-backed RAM and/or fuses for storage of a decryption key, which is used by vendor-embedded encryption components within the FPGA to decrypt the bitstream as it is read from the external NVM during the boot process [1]. Recent attack mechanisms that are able to read out on-chip stored keys therefore threaten the security of the boot process [2].

In this paper, we propose a PUF-based key generation strategy that addresses the vulnerability of on-chip key storage. Moreover, the proposed secure boot technique is self-contained in that none of the FPGA-embedded security primitives or FPGA clocking resources are utilized. We refer to the system as Bullet-Proof Boot for FPGAs (**BulletProoF**). BulletProoF uses a PUF implemented in the programmable logic (PL) side of an FPGA to generate the decryption key at boot time, and then uses it for decrypting an off-chip NVM-stored second stage boot image. The second stage boot image contains PL components as well as software components such as an operating system and applications. BulletProoF decrypts and programs the PL components directly into those portions of the PL side that are not occupied by BulletProof using dynamic partial reconfiguration while the software components are loaded into DRAM for access by the processor system (PS). The decryption key is destroyed once this process completes, minimizing the time the decryption key is available.

BulletProoF is stored unencrypted in an off-chip NVM and is therefore vulnerable to manipulation by adversaries. However, the tamper-evident nature of BulletProoF prevents the system from booting the components present in the second stage boot image if tamper occurs because an incorrect decryption key is generated. In such cases, the encrypted bitstring is not decrypted and remains secure.

The hardware-embedded delay PUF (HELP) is leveraged in this paper as a component of the proposed tamper-evident, self-authenticating system implemented within BulletProoF. HELP measures path delays through a CAD-tool synthesized functional unit, in particular the combinational component of SHA-3 in the proposed system. Within-die variations that occur in path delays from one chip to another allow HELP to produce a device-specific key. Challenges for HELP are 2-vector sequences that are applied to the inputs of SHA-3. The timing engine within HELP measures the propagation delays of paths sensitized by the challenges at the primary outputs. The digitized timing values are used in the HELP bitstring processing algorithm to produce a device-specific key.

2 BulletProoF Enrollment Process

During enrollment when the key is generated for the first time, HELP generates the key internally and transfers helper data off of the FPGA. As shown in Fig. 1, the helper data is stored in the external NVM unencrypted. The internally generated key is then used to encrypt the other components of the external NVM by configuring AES in encryption mode.

BulletProoF embeds a configuration bit that determines whether it is operating in Enroll mode or Boot mode. The bit is labeled “Enroll/Boot config. bit” in Fig. 1. The trusted party configures this bit to Enroll mode to create the Helper data and to process the “UnEncrypted SSBI” to an “Encrypted SSBI” that are both stored and used by the fielded version to boot. Therefore, the Enroll and Boot versions are identical except for this bit.

2.1 BulletProoF Fielded Boot Process

The FSBL loads the unencrypted version of BulletProoF from the external NVM into the PL portion of the FPGA and hands over control to BulletProoF. BulletProoF utilizes a ring-oscillator as a clock source that cannot be disabled dur-

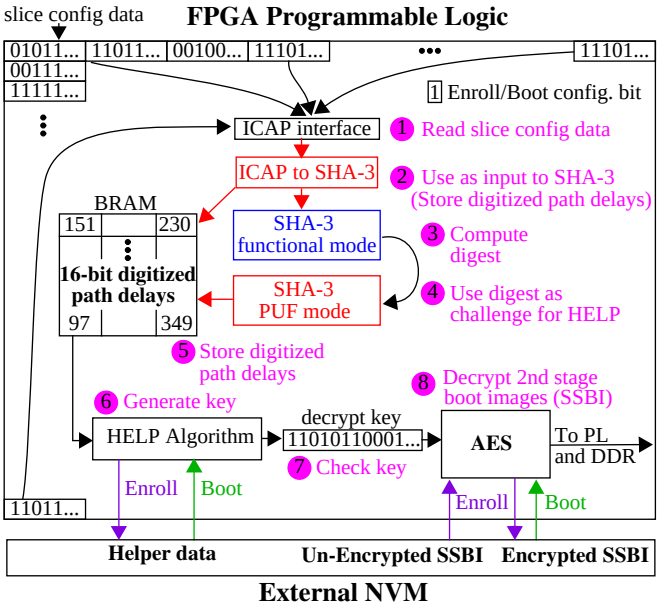

Fig. 1. Enrollment and in-field secure boot process.

ing the boot process once it is started. This prevents attacks that attempt to stop the boot process at an arbitrary point to reprogram portions of the PL using external interfaces, e.g., PCAP, SelectMap and JTAG PCAP.

1) BulletProoF reads slice configuration information using the ICAP interface and controller
2) The configuration information is applied to SHA-3 to compute a digest(s). Note that SHA-3 is configured in 'functional mode' during this step. The path delays between the ICAP interface and SHA-3 interface are timed, and the digitized delays stored in BRAM. This is done to prevent a reverse-engineering attack that we discuss below.
3) SHA-3 is used to compute one (or more) digests.
4) The digests are used as challenges to the SHA-3 combinational block with SHA-3 configured in PUF mode as they are generated.
5) The digitized timing values of sensitized paths are stored in a second on-chip BRAM.
6) The HELP algorithm processes the digitized timing values and Helper data which are stored in an External NVM into a decryption key.
7) BulletProoF runs an integrity check on the key.
8) BulletProoF reads the encrypted 2nd stage boot image (SSBI) from the external NVM. AES decrypts the image and transfers the software components into DDR and the hardware components into the unused portion of the PL using dynamic partial reconfiguration. Once completed, the system boots.

Note that the Enroll/Boot configuration bit is masked to zero when read from ICAP and used as input to SHA-3. This ensures that the same configuration data is used in either Enroll or Boot mode. Alternatively, it can be wired to an input (along with the start signal which is not shown).

2.2 Security Properties

The proposed system has the following security properties:

- The enrollment and regeneration process proposed for BulletProoF never reveals the key outside the FPGA. Therefore, physical, side-channel-based attacks are necessary in order to steal the key. We do not address side-channel attacks in this paper, but it is possible to design BulletProoF with side-channel attack resistance using circuit countermeasures proposed previously.
- Any type of tamper with the unencrypted BulletProoF bitstream or helper data by an adversary will only prevent the key from being regenerated and a subsequent failure of boot process. Note that it is always possible to attack a system in this fashion, i.e., by tampering with the contents stored in the external NVM, independent of whether it is encrypted or not.
- Any attempt to reverse engineer the unencrypted bitstream in an attempt to insert logic between the ICAP and SHA-3 input will change the timing characteristics of these paths, resulting in key regeneration failure. For example, the adversary my attempt to rewire the input to SHA-3 to allow external configuration data (constructed to exactly model the data that exists in the trusted version) to be used instead of the ICAP data.
- The paths from the ICAP interface to the SHA-3 inputs can be extended to run through the SHA-3 combinational block for additional resilience to reverse-engineering attacks. In other words, the blue and red instances of SHA-3 (which represent only one module) can be reversed in the figure, with functional mode run AFTER PUF mode.
- BulletProoF uses a ring oscillator as a clock source. Therefore, once BulletProoF is started, it cannot be stopped by the adversary as mechanism to steal the key.

2.3 Hardware Demonstration and Observables

We will build the secure boot system as described and carry out a live demonstration of the secure boot process by decrypting a HELP encrypted version. A graphical user interface will display the various stages of the boot process, with the assistance of 'hooks' inserted into the secure boot code for the purpose of the demonstration. A Xilinx Zynq 7020 SoC will be used as the demonstration platform. The unencrypted second stage boot image will consist of a PL side finite state machine that will be programmed into the fabric by BulletProoF using dynamic partial reconfiguration. The finite state machine will implement a simple design, e.g., a serial interface, that will allow confirmation that the system in fact booted.

3 References

[1] S. M. Trimberger, J. J. Moore, "FPGA Security: Motivations, Features, and Applications", Invited paper, *Proceedings of the IEEE*, Vol. 102, No. 8, 2014, pp. 1248-1265.

[2] S. Skorobogatov, "Flash Memory 'Bumping' Attacks", *Cryptographic Hardware and Embedded Systems*, 2010.

[3] J. Aarestad, P. Ortiz, D. Acharyya and J. Plusquellic, HELP: A Hardware-Embedded Delay-Based PUF, *Design and Test of Computers*, Mar., 2013, pp. 17-25.