SAND2022-5063C

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

D. Heeger, D. Owen, W. Che[1], F. Saqib[2], M. Areno[3], and J. Plusquellic

University of New Mexico, Enthentica[1], University of Carolina Charlotte[2], Trusted and Secure Systems[3]

Introduction

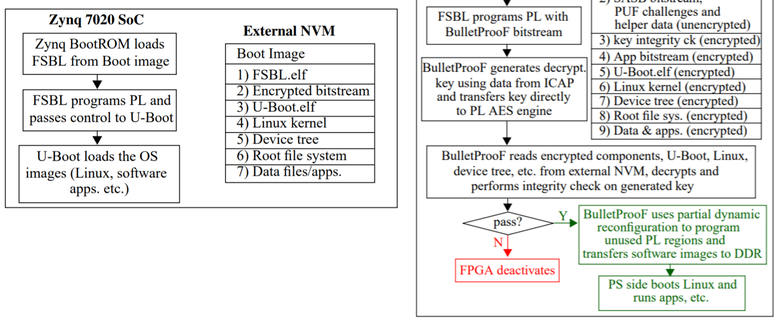
- Secure boot of an FPGA involves decrypting an encrypted bit stream
- Traditionally, decryption keys are stored in NVM
- This work proposes a new technique that uses a PUF challenged by the device configuration information as the decryption key
- The design self authenticates its own boot mechanism to detect trojans
- This technique is more robust against malicious tampering than traditional techniques

Background

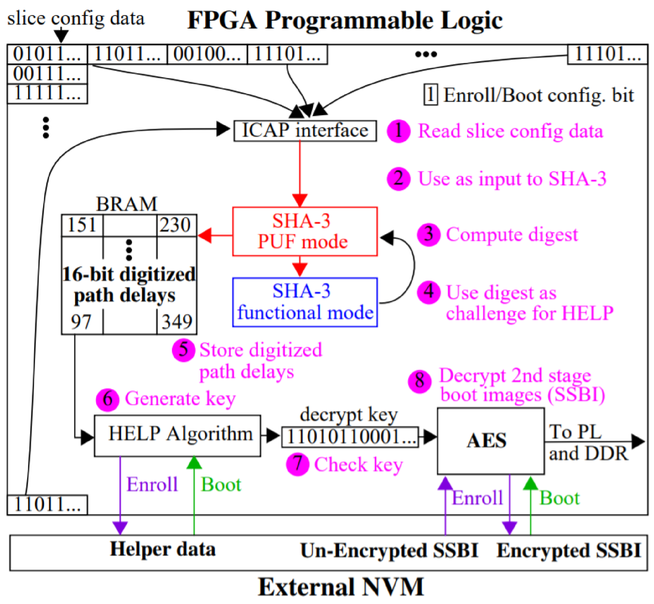
- Standard FPGA Boot Process uses an encrypted bit stream decrypted by a stored key to decrypt the data.
- Traditionally boot keys are stored in BBRAM or eFUSE
  - BBRAM has complicated maintenance since it requires a battery
  - eFUSE keys can be vulnerable to extraction using SEM
- HELP PUF
  - Physically Uncloneable Functions (PUFs) use device parasitics to generate random data
  - The HELP PUF uses variations in path delays as the source of randomness
  - A comparison of path delays generates the bits required to digitize the within-die variations

BulletProof Operations

Xilinx Zynq SoC Boot Process compared to the proposed Zynq SoC Boot Process



Program Logic Operations

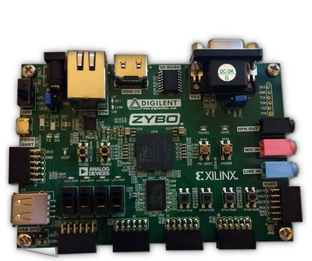


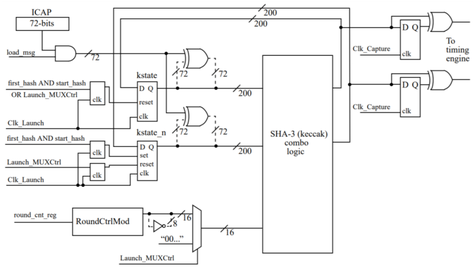
Security Considerations

- The bit stream decryption key is never exposed outside the FPGA
- Physical tamper results in the key permanently being destroyed
- A ring oscillator is used for the clock to prevent the device from being “frozen” to try to reveal keys
- External interfaces are disabled immediately (PCAP, JTAG,
- The decryption key and timing data are deleted as soon as the bit stream is decrypted

Concept Development

- A Zybo board using the Zynq-7000 was used for testing
- The diagram on the right shows the detailed logic implemented in the FPGA





Future Work

- Modifying the unencrypted bit stream and assessing the PUF performance relative to the amount of changes
- Inserting leakage trojans to the design and verifying it properly detects them