Breaking the Trust Dependence on Third Party Processes for Reconfigurable Secure Hardware

Aimee Coughlin, Greg Cusack, Jack Wampler, Eric Keller, Eric Wustrow
What is secure hardware?

Hardware that protects against other parts of the system
• Implemented such that you can trust its functionality

• Useful for defending against untrusted software

• Can defend against some kinds of physical threats
Trusted Platform Modules

• Secure coprocessor that provides cryptographic functions and key storage

• Most popularly used for disk encryption
Secure Boot

• Ensures that only trusted system software can boot by checking the signature of the software before it boots.
Trusted Execution Environments

arm
TRUSTZONE

Isolated App
Operating System

TEE
CPU

Isolated Memory
Memory

App 1
App 2
Problem: Only Manufacturers Make Decisions

- What features to include
- When and if patches are available
What if Developers Could Make These Decisions?

- Application
- Operating System
- Processor
- Secure Hardware

define
Enter FPGAs

• Leverage programmability of FPGAs to enable reconfigurable secure hardware

• Expose programmability to developers
The Downside of Programmability

- Immutable nature of silicon is a basis for the guarantees of secure hardware

- Programmability compromises security properties
The Downside of Programmability

• Immutable nature of silicon is a basis for the guarantees of secure hardware

• Programmability compromises security properties
Bitstream Protection (widely available)

Desired Config

Encrypt Bitstream

Decrypt bitstream

Secure HW Function

FPGA
Bitstream Protection – can’t attack the device

Desired Config

Encrypt Bitstream

Decrypted bitstream

Secure HW Function

FPGA

Attacker
Bitstream Protection – CAN attack the process

- Encrypt Bitstream
- Decrypt bitstream
- Insecure HW Function
- Desired Config

Attacker

Encrypt Bitstream

FPGA
Breaking the Trust Dependence on Third Party Processes for Reconfigurable Secure Hardware
High-level Idea

- Self-provisioning
  - Key creation and maintenance is internal to device

- Policy-controlled update system
Defining some Roles

• FPGA Manufacturer
• System Manufacturer
• System Provisioner  <= loads an initial FPGA configuration
• Application Developer <= loads secure hardware configuration
• User                  <= operates the device

(roles may be overlap)
Self-provisioning

• Start with empty device

- Secure Storage
- Secure Boot

FPGA
Self-provisioning

- Start with empty device

- Secure Storage
  - Can only be read by specific hardware

- Secure Boot

FPGA
Self-provisioning

- Start with empty device

- Stores a public key
- the device will only load an image generated with the corresponding key
Self-provisioning (1) – Generate Key Pair

- Special self-provisioning config used by system provisioner

1. Generate a key pair
2. Store the secret key in secure storage.
3. Program the public key to the secure boot system on the device.

**Initial config**

**Self Provision config**

**Secure Storage**

**Secure Boot**

**FPGA**
Self-provisioning (1) – Generate Key Pair

- Special self-provisioning config used by system provisioner

1. Generate a key pair
2. Store the secret key in secure storage.
3. Program the public key to the secure boot system on the device.
4. Sign/encrypt the initial FPGA configuration with secret key.
Self-provisioning (2) – Load Initial Config

• Only this specific configuration can be loaded onto the FPGA (next power cycle)

Initial Config (Signed)

Secure Storage

Secure Boot

FPGA

Update System (w/ INITIAL policy)

Initial policy could include a one-time use key
• The user initiates the loading of a new config

(1) Receive an update.
(2) Verify that the update conforms to the update security policy.
Loading Secure HW App (1) – verify policy

• The user initiates the loading of a new config

1. Receive an update.
2. Verify that the update conforms to the update security policy.
3. Use secret key to sign/encrypt the update (using embedded encryption hw)
Loading Secure HW App (2) – Load Desired Config

- The secure HW App is then loaded

![Diagram](image)
Update Policies: Implemented by the loaded config

- Flexibility to use a variety of means to protect the update (including multiple-factors)
  - User inputs, key maintained by trusted dev, key maintained by user, etc.

- Flexibility to implement a variety of policies
  - Trust once (initially unprotected)
  - One time key (protected by shared key)
  - 2 factor (signed by trusted dev and user input PIN)
  - No updates allowed
Implementation

• Self-provisioning, secure update, secure storage on Xilinx Zynq Ultrascale+
• Application: TEE (like SGX but with custom root of trust)
SDK for the TEE

Enclave Applications:
- SHA512
- Password manager
- Just copy-in / copy-out
- Contact Matcher (like Signal)
Conclusion

Main Take Away:
• We don’t need to trust the system provisioner to maintain keys

Our system
• protects the most important key (self-provisioning)
• provides flexibility to determine how updates happen (policy)
Thank you

Eric.Keller@Colorado.edu