FPGA SoC Architecture and Runtime to Prevent Hardware Trojans from Leaking Secrets

Gedare Bloom, Bhagirath Narahari, Rahul Simha
George Washington University
Washington, DC
{gedare,narahari,simha}@gwu.edu

Ali Namazi, Renato Levy
Intelligent Automation, Inc.
Rockville, MD
{anamazi,rlevy}@i-a-i.com

***Abstract*—Hardware Trojans compromise security by invalidating the assumption that hardware provides a root-of-trust for secure systems. We propose a novel approach for an FPGA system-on-chip (SoC) to ensure confidentiality of trusted software despite hardware Trojan attacks. Our approach employs defensive techniques that feature morphing on-chip resources for moving target defense against fabrication-time Trojans, onion-encryption for confidentiality, and replication of functionally-equivalent variants of processing elements with arbitrated voting for resilience to design-time Trojans. These techniques are enabled by partial runtime reconfiguration (PRR) and are managed by a hardware abstraction layer (HAL) that reduces developer burden. We call our approach the *Morph Onion-encryption Replication PRR HAL*, or MORPH. MORPH aims to provide a stable interface for embedded systems developers to use in deploying applications that are resilient to hardware Trojans.**

I. Introduction

During integrated circuit (IC or chip) manufacturing, untrusted entities along the supply chain have ample opportunity to add malicious inclusions, or hardware Trojans, to chips. Once active, the malicious circuitry can leak cryptographic keys to allow unprivileged access to information including proprietary code and sensitive data. At the same time, embedded systems increasingly rely on field programmable gate arrays (FPGAs) for economic reasons such as shorter development-release cycles and economy of scale. Library implementations of soft core processors have become readily available and FPGAs have achieved commoditization with software support and size sufficient for system-on-chip (SoC) designs. An FPGA-based soft SoC has flexibility advantages over ASICs, albeit at decreased performance, and commercial availability of partially runtime reconfiguration (PRR) has further increased the flexibility appeal of FPGAs. Unfortunately, the widespread reuse of circuit designs creates the possibility that untrusted third parties can inject malicious code into soft IP cores. Furthermore, hard IP cores remain vulnerable to hardware Trojans injected into FPGA circuitry at fabrication-time in an untrusted foundry.

We propose a framework for current FPGA devices that offers security against design- and fabrication-time hardware Trojans. This framework combines moving target defense using PRR, encrypted execution using onion-encryption, and a stable hardware abstraction layer (HAL) that isolates software from the complexity of the underlying hardware; see Figure 1. We call our approach *Morph Onion-encryption Replication PRR HAL*, or MORPH. Applications running on MORPH are minimally impacted, because they execute on the SoC's main CPU in parallel to and independently from the HAL, which manages the PRR. Our initial results indicate that MORPH incurs low size and time overhead (below 1% each on state-of-the-art FPGA devices) after device boot completes and SoC execution begins. The combined effect of MORPH is to shift the root of trust from the hardware into the HAL so that attacks on the hardware do not compromise security of the software.

II. Threats and Attacks

There are two widely-accepted threat models for hardware Trojans: untrusted foundry and untrusted designer. We assume both models, and consider (possibly colluding) hardware Trojans. MORPH's security objective is to prevent leakage of the application's code and data that executes in the SoC.

The strongest attack in our threat model is one that introduces hardware Trojans in IP cores that collude with hardware Trojans added during fabrication. Trojans injected at the foundry will target fixed structures on the FPGA device, or, with a known expected configuration, specific locations of the reconfigurable logic. Note that attacking such fixed locations is not simple, because the FPGA device vendor tests these areas prior to shipping devices to consumers. Furthermore, the area of fixed units is small and therefore amenable to visual inspection and sampling-based destructive testing. Trojans in IP cores could rely on the functionality of Trojans added by the foundry, thus any channel between untrusted third party IP and fixed FPGA structures must be treated with caution.

III. MORPH: Architecture for FPGA Security

We named the MORPH platform for the five capabilities it relies upon, which we describe after explaining the bootstrap process and cryptographic keys that MORPH uses.

The configuration bitstream of MORPH comes in two parts: the boot configuration with an initial HAL configuration we call the BOOT HAL, and the runtime configuration, which contains another HAL configuration and soft IP cores for MORPH's SoC. The bootloader contains a decryption module with access to the key (see below) for the boot configuration. The bootloader places the BOOT HAL in reconfigurable logic, begins its execution, and then halts. The runtime configuration

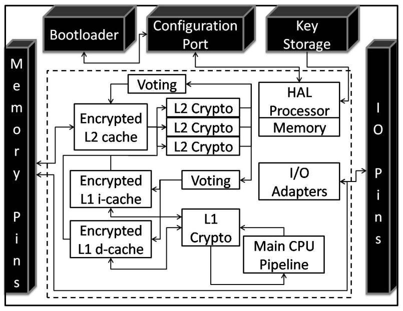
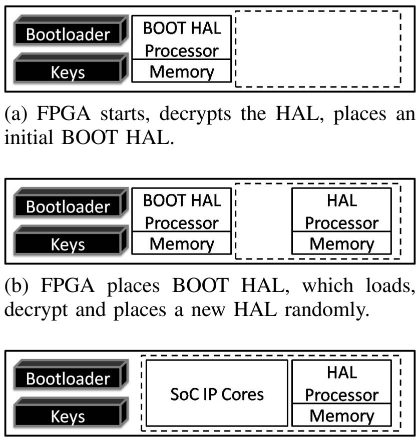


Fig. 1. Architecture of Morph Onion-encryption Replication PRR HAL (MORPH). Black-filled cubes represent hard IP cores. The dashed area holds the SoC and HAL, soft IP cores that are subject to morph operations.



(a) FPGA starts, decrypts the HAL, places an initial BOOT HAL.

(b) FPGA places BOOT HAL, which loads, decrypt and places a new HAL randomly.

(c) HAL loads, decrypts and places SoC IP cores randomly.

Fig. 2. MORPH uses standard FPGA bootstrapping techniques to load the BOOT HAL, which loads and starts the HAL processor. The HAL builds the SoC, begins morphing it, and starts application execution.

is encrypted by a key that the BOOT HAL reads from non-volatile storage. The BOOT HAL loads the HAL processor from the runtime configuration, configures it to a random location in the FPGA, starts it, then halts. The HAL processor builds the SoC and starts the application while periodically morphing SoC resources. Figure 2 demonstrates the boot process of MORPH.

MORPH uses at least four cryptographic keys, which are loaded in on-chip non-volatile storage at a trusted site before deployment. The *boot-key* is used by the bootloader to decrypt the configuration containing the BOOT HAL, as described above. The *HAL-key* is used by the BOOT HAL to read the configuration of the HAL processor and its program, and then transferred to the HAL when it starts. The HAL uses the HAL-key to decrypt the soft IP cores composing the SoC. Two additional keys, the *L1-key* and *L2-key*, are used for off-chip I/O including memory access. These latter keys are named after the levels of cache at which they are applied. The L1-key encrypts the stores (respectively decrypts the loads) between the CPU in the SoC and the L1 caches. The L2-key does the same between the L1 caches and the L2 cache. (Uncached accesses still go through the cryptographic units.)

M. Morph operation

Morph is an operation conducted by the HAL to rearrange the hardware resources in use by the executing system by replacing and binding resources in a new location. Resources are available at the granularity of soft IP cores from a library as commonly used in FPGA-based development. The soft IP cores are relocatable designs of functional units that can be hierarchically composed to create complex configurations up to the a complete SoC. The use of third-party IP cores raises the possibility of malicious code in the design level, which we address with onion-encryption and replication. The security principle behind morphing is moving target defense, which is the idea that variance in the attack surface at runtime increases the difficulty of attack.

O. Onion-Encryption

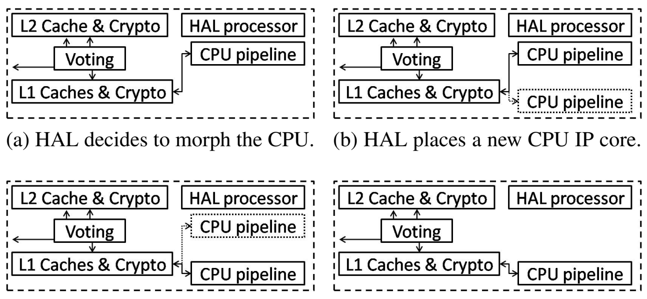
We adapt dual-encryption from prior art [1], but now deployed in a single-chip SoC with a traditional memory hierarchy. For multi-layer encryption, we prefer the term onion-encryption. We use onion-encryption to protect against leakage of code and data by adding layers of encryption along the path memory takes in the system. All of memory begins onion-encrypted and remains that way throughout execution. Peripherals only ever see onion-encrypted memory. The outer layer of an onion is decrypted (encrypted) in between the L2 and L1 caches, and the inner layer is decrypted (encrypted) in between the L1 caches and the main processor. The security principle for onion-encryption is defense-in-depth: any single compromise of cryptography does not lead to total loss of confidentiality.

R. Replication

Replication of functionally-equivalent IP cores obtained from different vendors is one way to prevent design-time hardware Trojans from successfully modifying the behavior of a chip. With more than two IP cores, a voting protocol can determine which output to select in case of disagreement. Beaumont et al. [2] pioneered this approach, along with fragmented execution to limit access to code and data. We use a similar scheme as the prior work, except we limit replication to the outer-most cryptographic modules used in onion-encryption, which we suggest is sufficient for ensuring confidentiality. MORPH uses cryptography rather than fragmented execution to protect from leakage, the HAL implements replication, and a simple circuit arbitrates voting of the outer layer onion-encryption modules.

P. PRR

PRR support enables a reconfigurable device to modify part of its configuration while the device runs. The HAL uses PRR to relocate IP cores in available FPGA resources dynamically by placing a new IP core, stopping the original component, transferring its state and connections, and resuming normal execution, all without rebooting the FPGA. Some parts of an FPGA design are not reconfigurable, such as the clock circuitry and I/O pins. (Figure 1 shows those fixed parts as 3-D



(a) HAL decides to morph the CPU. (b) HAL places a new CPU IP core.

(c) HAL stops the FPGA and updates routing lines. (d) HAL starts the FPGA and erases the old CPU.

Fig. 3. MORPH uses PRR to relocate parts of an SoC. Here, the HAL replaces a processor pipeline.

black-filled boxes.) MORPH uses a soft IP core on the FPGA to implement PRR by reading the encrypted configuration data from a non-volatile memory and loading it into the reconfigurable block. During PRR the SoC remains running except for a brief interruption to update routing lines from an old block to its replacement. Figure 3 shows an example of using PRR to replace a processor pipeline stage.

H. HAL

Operating systems use HALs to operate on third party hardware. The HAL in MORPH is more sophisticated than that found in an OS, and instead is an active component that bootstraps the SoC, manages cryptographic keys, and randomly—using a true random number generator (TRNG) circuit—morphs itself and the SoC IP cores on-the-fly. Application software and IP cores execute without any knowledge of the HAL's operations.

MORPH's HAL consists of two parts, a soft IP core with a simple processor and internal memory, and a software runtime. When the HAL boots it immediately begins morph operations by relocating itself. It also zeroes-out the bootloader, decrypts the SoC IP cores, and starts to morph them into the reconfigurable fabric. After the HAL instantiates all the IP cores, it starts execution of the application on the CPU while continuing to morph the IP cores.

IV. Security Analysis

The morph operation prevents an attacker at fabrication-time from knowing where to place hardware Trojans. Even if the attacker knows the configuration, the location of any particular IP core cannot be predicted after the HAL boots (subject to the security of the TRNG), thus limiting fabrication-time Trojans to attacking fixed structures on the FPGA (hard IP). As we noted earlier, the bootloader structure is small and verifiable by sampling, imaging, and traditional circuit extraction techniques. The layered encryption of all outgoing data ensures that any information available at I/O pins is encrypted. Since all data are onion-encrypted coming out of the reconfigurable logic, Trojans in the fixed structures cannot successfully leak unencrypted data.

Onion-encryption in combination with replication also prevents leakage stemming from design-time Trojans that circumvent cryptographic units. The burden for the attacker increases as a majority of the cryptographic cores must be compromised to disable onion-encryption. More modules can be added to increase the burden further. Furthermore, cryptographic modules with access to both the L1-key and L2-key would need to be compromised in order to leak both cryptographic keys and thus be able to read the encrypted main memory. Thus, the security guarantee holds probabilistically when more than half of the cryptographic modules are trustworthy.

The trusted computing base (TCB) includes the BOOT HAL, HAL, and cryptographic key storage. Successful compromises of the TCB leads to loss of security for the system. Note that the bootloader is not in the TCB, but if confidentiality of soft IP cores in the SoC is required, then the bootloader must be trusted and therefore verified. We assume fabrication-time Trojans in the key storage and initial location of the BOOT HAL are detectable. After boot, only the HAL and key storage remains in use, as the bootloader and BOOT HAL hand-off control of the device to the HAL processor, which erases the BOOT HAL and starts morphing itself randomly across the chip.

V. Preliminary Results

MORPH consumes FPGA resources, incurs performance loss, and increases energy consumption. FPGA resources, which include configurable logic (slices composed of flip-flops and look-up tables) and on-chip memory (block RAM or BRAM). Performance is affected by added boot time, cryptographic operations, and reconfiguration. Other considerations when evaluating MORPH are energy dissipation and size requirements, i.e. the usage of FPGA resources. In terms of measuring MORPH's impact on performance, we considered the added boot costs and the cost for the HAL to morph one module periodically. The baseline we use is a statically-configured FPGA SoC that lacks all of MORPH's features.

We model MORPH using a Xilinx PicoBlaze 8-bit processor with 16K of on-chip block RAM (BRAM) for the HAL processor, and a simple AES-128 cryptographic module. In our evaluations, we target two FPGAs from Xilinx: the Virtex-5 110T and the Virtex-7 485T. At most, the HAL processor uses 96 FPGA slices and 16 BRAMs, and each AES module uses 74 slices and no BRAM. MORPH requires five AES modules (that should be independently sourced), one attached to the HAL processor for decrypting configurations, one attached to the main CPU within the SoC, and three between the SoC and off-chip; the last three have their outputs routed through a simple majority voting circuit.

We use a partial initial configuration for booting, thus the overhead for booting is driven primarily by loading the BOOT HAL and HAL. The cost for the HAL to load the runtime configuration is approximately the same as the cost to configure the FPGA device with the same SoC. Thus, the overhead for boot is the time needed to load the BOOT HAL and HAL configuration. We estimate these overheads by supposing the BOOT HAL and HAL both fit in PRR IP blocks that use 5% each of the FPGA (a conservative estimate, see

Process	Time Overhead
Boot	10%
PRR (1s period)	1%
PRR (5s period)	0.2%
PRR (10s period)	0.1%

TABLE I
MORPH TIME COST ESTIMATES.

FPGA Resources			MORPH Usage	
Device	Slices	BRAMs	Slices	BRAMs
Virtex 5 110T	17,280	296	2.70%	5.4%
Virtex 7 485T	75,900	2940	0.61%	0.54%

TABLE II
RECONFIGURABLE AREA REQUIREMENTS FOR MORPH

size costs below), thus the time overhead for boot is an extra 10% to configure these blocks.

Under normal processing (not during reconfiguration), the HAL executes in parallel to the SoC, therefore contributing no time overhead. Similarly, onion-encryption operates in parallel to the memory path, and is not expected to contribute to the critical path. The static power overhead for the HAL and AES modules is less than 1% of overall power. We have not yet quantified the dynamic power overhead, which is application-dependent.

During the reconfiguration portion of relocation, the processing clock stops for about 10 ms, which determines the bulk of the time overhead for morphing. During this time only static (quiescent) power is consumed. Note that PRR costs are linearly inversely-proportional to the period.

We calculated costs for boot and reconfiguration at periods of 1, 5, and 10 seconds, and report results in Table I. The costs for booting seem high as a percentage, but the costs are reasonable due to the short duration and infrequency of booting. For example, the Virtex-7 device takes 405 ms to load a full configuration, so 10% overhead is about 40 ms longer. We did not account for other costs associated with morphing, which include loading and decrypting bitstreams into the HAL, finding an unused random location, and erasing the old functional unit after it has been replaced.

MORPH's size overhead is the extra area required for the HAL processor, its memory, and the five AES modules. Table II shows the size overhead when using MORPH in two popular devices from Xilinx.

VI. Related Work

We build on our past work with dual-encryption [1] and architecture-OS protocols for detecting hardware Trojans [3]. Waksman and Sethumadhavan [4] use cryptography at the granularity of functional units in a processor to defend against value-triggered Trojans, but only against the untrusted designer. Replication using functionally-equivalent variants was first proposed by McIntyre et al. [5] at the software-level to detect hardware Trojans, and Beaumont et al. [2] propose an architectural solution using replication, voting, and fragmented execution. These works focus on untrusted design, and attacks from untrusted foundry or end-user void security guarantees. Kim and Villasenor [6] propose to replace functions in an ASIC with FPGA implementations when a hardware Trojan is detected in the ASIC, which is functionally correct but requires a Trojan detector and Trojan-free FPGA—MORPH is proactively defensive and assumes Trojans are present in the FPGA. Mentens et al. [7] add temporal and spatial jitter in cryptographic hardware similar to the morph operation, but the lack of variation in functional blocks means design-time Trojans are not defended against.

VII. Conclusion

The MORPH framework provides the ability to design embedded software systems resistant to hardware Trojan attacks without increased development time and at low runtime cost. Our plans for future work include defending against physical attacks (malicious end user), exploring the design-space for security-performance tradeoffs, and ensuring integrity as a security objective.

Acknowledgment

This work is supported in part by ONR N00014-14-1-0386.

References

[1] G. Bloom, B. Narahari, R. Simha, and J. Zambreno, "Providing secure execution environments with a last line of defense against trojan circuit attacks," *Computers & Security*, vol. 28, no. 7, pp. 660–669, Oct. 2009.
[2] M. Beaumont, B. Hopkins, and T. Newby, "SAFER PATH: Security architecture using fragmented execution and replication for protection against trojaned hardware," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, Mar. 2012, pp. 1000–1005.
[3] G. Bloom, B. Narahari, and R. Simha, "OS support for detecting trojan circuit attacks," in *Hardware-Oriented Security and Trust, IEEE International Workshop on*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 100–103.
[4] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *2011 IEEE Symposium on Security and Privacy (SP)*, May 2011, pp. 49–63.
[5] D. McIntyre, F. Wolff, C. Papachristou, S. Bhunia, and D. Weyer, "Dynamic evaluation of hardware trust," in *Proceedings of the 2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, ser. HST '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 108–111.
[6] L.-W. Kim and J. Villasenor, "Dynamic function replacement for system-on-chip security in the presence of hardware-based attacks," *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 661–675, Jun. 2014.
[7] N. Mentens, B. Gierlichs, and I. Verbauwhede, "Power and fault analysis resistance in hardware through dynamic reconfiguration," in *Cryptographic Hardware and Embedded Systems CHES 2008*, ser. Lecture Notes in Computer Science, E. Oswald and P. Rohatgi, Eds. Springer Berlin Heidelberg, Jan. 2008, no. 5154, pp. 346–362.