Securely Sealing Multi-FPGA Systems

Tim Güneysu[1], Igor Markov[2], and André Weimerskirch[3]

[1] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
[2] University of Michigan, 2260 Hayward St., Ann Arbor, MI 48109-2121
[3] ESCRYPT Inc, 315 E. Eisenhower Parkway, Ann Arbor, MI 48108

Abstract. The importance of hardware security of electronic systems is rapidly increasing due to (1) the increasing reliance of mass-produced and mission-critical systems on embedded electronics, and (2) the ever-growing supply chains that disentangle chip designers and manufacturers from OEMs. Our work shows how to dramatically reduce vulnerability to Trojan-horse injection and in-field component replacement. We propose methods to verify the authenticity and integrity of an FPGA configuration during startup and at runtime. We also developed efficient protocols for electronic sealing of a multi-FPGA system, which automatically enforces the system configuration detected upon power-up and bans further modifications.

1 Introduction

The proliferation of fake goods in the electronics industry is well documented. Between 2005 and 2008, US and Canadian authorities seized $78M worth of counterfeit Cisco hardware from China, including network modules, WAN interface cards, gigabit interface converters, and less expensive routers [8]. An even larger amount of fake networking hardware had previously been sold to customers, including financial institutions, universities, the FAA, the FBI and several branches of the US military. The Anti-Counterfeiting Task Force established by the Semiconductor Industry Association reports that "one company has seen fakes of 100 separate parts in three years".

Reportedly, the FBI is concerned that widely seen counterfeit equipment may be state-sponsored, to facilitate access to otherwise secure systems [2, 8]. *IEEE Spectrum* pointed out that a Syrian air-defense radar may have been sabotaged during an Israeli bombing raid in September 2007 through a "kill-switch" planted by a European chipmaker [3]. With semiconductor manufacturing increasingly outsourced overseas, the DARPA-sponsored "Trust" program is addressing potential subversion of military electronics by seeking techniques to detect and disable such alterations. It ensures rigorous evaluation by funding work on concealing Trojan horses in ICs and identifying such Trojans.

Altering an IC often requires painstaking reverse engineering, but in some cases such alteration can be easy. For instance, the proposed "obligatory accreditation system for IT security products" in China [1] demands source code from hardware manufacturers, making reverse engineering unnecessary.

The challenges addressed in our work include (1) defeating Trojan horses, (2) component authentication in single- and multi-chip systems, and (3) countering unauthorized component replacement with deficient or compromised variants.

Contribution: Our proposal signficantly reduces the vulnerability of electronic systems by reducing the time interval when component replacement can be successful. It encompasses

– *Delayed logic design* via the use of FPGAs, which defeats many malicious alterations before system integration.
– *Chip integrity verification* required to prevent replacement of an existing chip.
– A protocol for *electronic sealing* of a multi-chip system that continually authenticates individual chips and prevents unauthorized replacement after the first power-up.

Outline: The remaining part of this paper is organized as follows. Section 2 covers our security objectives for secure (FPGA-based) hardware systems. Here we rely on security infrastructure available in recent FPGA devices, reviewed in Section 3. Section 4 outlines assumptions and requirements necessary to establish the methods and protocols proposed in Sections 5 and 6. Implementation details are discussed in Section 7, and concluding remarks are given in Section 8.

2 Security Objectives for Hardware Systems

The dramatic rise in chip piracy and concerns about subversion are due to the recent separation of design and manufacturing, traditionally performed by one company. Companies now tend to narrow their focus and fit into long supply chains that end with OEM system integrators. Convincing solutions to security challenges must account for this context.
Delayed Logic Design. It would be difficult to subvert a chip *before* its functionality is available to the attacker in any form — hardware or software. Therefore, *delaying logic design until after manufacturing and shipment* can dramatically reduce the time period during which a chip is vulnerable to replacement or alteration. Such delay is made possible by modern FPGA chips, which are sufficiently large, fast and cheap for many applications. Given their regular structure, blank FPGA chips can be tested quickly, after which a system integrator can program each chip with a precomputed trusted bit-stream.
Chip Integrity Verification. Along with ASICs and custom-designed chips, FPGA-based chips used in larger systems run the risk of being replaced by knock-offs or maliciously altered designs. Unfortunately, the low cost of FPGAs and the simplicity of FPGA design tools facilitates single and mass-produced knock-offs. However, such replacements are easier to detect because FPGA configurations are completely determined by discrete parameters (configuration bits). Therefore, one can compute a cryptographic hash of FPGA bit-streams and use this fingerprint to periodically authenticate the chip using a challenge-response protocol.
Electronic Sealing. Field-programming of component chips by a trusted system integrator defeats many replacement and Trojan-horse attacks performed *prior to system integration*. To guard against attacks *after system integration*, we use chips that report their own identity. The identity of each chip must be checked reliably, so that

– faking any part of the chip, such as communication circuits, would not allow passing identity checks.
– chips that lie about their identity (e.g., trying to replay overheard communications) do not pass the checks.

We address these challenges by several protocols for *electronically sealing* multi-chip systems. One option is for a dedicated hardware entity to periodically verify the identity of each FPGA by checking if it retains a shared secret. These checks can be performed between normal bus-level communications and would detect changes in a chip's identity.

3 State of the Art in FPGA Security

To detect chip tampering after start-up, our protocols for authenticating chip configurations use recent FPGA technology. We therefore review security features and techniques in recent FPGAs and in academic literature.

3.1 Security Features of Modern FPGAs

The introduction of bitstream encryption was the first feature that significantly improved FPGA security. Today, many (volatile) FPGAs support encrypted bitstreams to cryptographically protect configuration stored in an external memory – eliminating the threat of inspection and theft by unauthorized parties. Besides *confidentiality* of the Intellectual Property (IP) contained in the FPGA configuration, the installation of a secret key on each device can be also used as an effective tool to prevent *overbuilding* and *device cloning*. However, bitstream encryption does not solve all security issues. For example, attacks on the integrity of an FPGA configuration (which performs a security-related function) are countered by a non-cryptographic CRC checksum on most FPGAs, and CRC is easy to manipulate. Finally, some FPGA manufactures (such as Xilinx) install a device-specific identification number (ID) on their FPGAs with which FPGAs can be distinguished. For example, Xilinx' DeviceDNA is a 57-bit static device ID (of which effectively only 55-bit are used) that could be employed for device identification in cryptographic binding or sealing protocols. We classify modern FPGA devices as follows:
C1. FPGAs with authenticated encryption. Xilinx Virtex-6 FPGAs offer an AES-256 bitstream encryption and HMAC (based on a SHA-256 hash) function that can be used to ensure configuration confidentiality, authentication and integrity during power-up. Authentication/identification of multi-chip systems and (cryptographic) integrity checks at runtime are not supported.
C2. FPGAs with bitstream encryption only. Many high-performance FPGA (such as Xilinx Virtex II/4/5/6 FPGAs, larger Spartan-6, Altera Stratix II/III/IV/V, Cyclone III LS, Actel ProASIC3, LatticeECP) devices allow the use of encrypted bitstreams, mostly using AES-128 or AES-256 encryption. However, cryptographically strong authentication and integrity checks for the bitstream are not supported.
C3. FPGA without cryptographic features. Low-cost FPGA devices typically do not provide any cryptographic protection mechanisms for their configurations, except for an identification number (e.g., in Xilinx Spartan-3 A/AN devices). Since these devices do not provide any cryptographic trust anchor, an FPGA developer needs to include security features within the configuration bitstream. The security of this approach is bounded by the difficulty of reverse-engineerng the bitstream that contains the cryptographic secrets. Therefore, adapting our techniques to low-security FPGAs would require more work and incur a greater overhead.

3.2 Prior Academic Work

In [6, 21], the authors compiled a comprehensive list of security risks of FPGA-based systems. Further guidelines on trusted FPGAs can be found in [18, 20]. Several proposals perform bitstream authentication by extending the bitstream decryptor core within the FPGA's static configuration logic with an authentication function [5, 16]. Further work discusses the challenges of establishing and storing secret keys. Instead of plain memory cells that store the secret key, one can use Physically Unclonable Functions [15], which can be configured to derive cryptographic keys from chip-specific hardware fluctuations. For example, key establishment schemes based on (rather complex) PUF-based protocols were proposed in [9, 19]. Keys can be established through public-key cryptography [10].

4 Threat Model, Assumptions and Requirements

Electronic hardware is produced and employed by several actors. The *Hardware Manufacturer* (HM) produces FPGA devices. The *Intellectual Property Owner* (IPO) owns application-specific designs, synthesizes them into configuration bitfiles and licenses for a fee, usually on a per-volume basis. The *System Integrator* (SI) embeds these designs in FPGA devices. The *End User* (EU) physically possesses the product, using it as intended or abusing it. The *Trustworthy Owner* (TO) owns the entire electronic system[4].

Threat model. We assume that the IPO is trustworthy and seeks protection from design subversion. Our proposal relies on SI, also assumed trustworthy. Given that FPGAs are easy to program, SI and IPO may be the same entity. [5] The EU may alter the system, and any malicious third party is included in the EU group. The HM includes the entire development chain, and we distinguish several cases in terms of trustworthyness.

The HM (or any part of the design and manufacturing chain) can introduce Trojans, malicious programs, and backdoors in the FPGA. To defeat some of these attacks, we assume that SI can randomly sample FPGAs chips from all FPGAs produced, e.g., by purchasing them through proxies/anonymously. This undermines targeted manipulation of FPGAs for use by the HM. Furthermore, we assume that the SI *randomizes FPGA place-and-route*, so that each maps FPGA differently, making it impossible to predict where each LUT/gate will go in the FPGA (note that ASICs and custom-made chips cannot afford this approach). Thus, HM's Trojans or malicious routines cannot connect to specific internal signals of the chip. However, if the HM includes a general trigger (e.g., time-based), the issue will be detected by other users of the FPGA, very likely before it can cause major damage to TO's systems. The HM may be able to inject a backdoor to read out the FPGA configuration. However, this issue is in the domain of protection of intellectual property rather than our objective. Furthermore, our analysis assumes that HM already obtained the FPGA configuration and can modify it maliciously.

The EU is able to add/remove one or several FPGA(s) to/from the system. The added FPGAs may be under attacker's control. The EU is able to tap into and alter

[4] The TO does not always run FPGAs devices. For instance, military systems are owned by the government but run by specific detachments.

[5] If FPGAs are shipped by an untrusted party, the receiver can check FPGA's integrity using proposed techniques. In this case, the initial receiver is included in the SI group and assumed trustworthy.

the communication between FPGAs. The attacker is also able to inject messages to the communication channel between FPGAs, but not probe an FPGA or extract security credentials while the system is powered. When the system is powered down, the attacker is not able to physically extract security credentials in a short time period (say one hour). We assume that the attacker is not able to replace the entire system (including all FPGAs) at once. We assume that the TO regularly checks the system. For instance, one of the FPGAs might be located in a physically secure compartment that is only accessible to the TO and that regularly provides a status notification to the trustworthy TO. Our proposed mechanisms will defeat all attackers that are located within the described attacker model.
Further assumptions and requirements. The general *security objective* pursued in our work is to prevent altered chips from successfully running in a computer system. Therefore we propose cryptographic authentication and identification services for single and multi-chip systems. We rely on trustworthy hardware infrastructure for authentication and on initialization at a physically-secure facility by a trusted party. We also assume that each FPGA passes an *enrollment phase* performed by a trusted party (e.g., the system integrator) during which secrets are installed in the device or system. During system initialization in this environment, keys K_i for bitstream encryption are generated and loaded in the secure key store of each FPGA (either by programming eFuses or the battery-backed-up RAM). We assume that bitstreams are authenticated and that there is a mechanism in place to counter loading old versions (e.g., each bitstream version uses an individual bitstream encryption key). We further make the non-trivial assumption that CAD tools for FPGAs, used at the secure facility, are trusted and do not cause unintended side-effects [17]. Note that the possibility of subversion *before* initialization and enrollment, e.g., by injecting Trojan horses into the FPGA fabric, is inherently difficult since the FPGA configuration may be randomized at the last minute (the principle of *delayed logic design*) and multiple FPGA configurations, loaded one after another, can use the same logic elements.

Based on our assumptions, we now present mechanisms to integrate cryptographic configuration integrity checking and electronic sealing for FPGA devices of classes C1,C2. The availability of bitstream encryption in these two device classes can be regarded as basic trust anchor that enables the implementation of upper-layer security objectives on top (i.e., cryptographic authentication and integrity checks). Note that our proposals are solely based on symmetric cryptography that can be implemented in the fabric of an FPGA. This involves the standardized AES block cipher for encryption/decryption, an authentication function such as HMAC-SHA-256, and a method to distinguish FPGA devices (e.g., using DeviceDNA or PUFs). We further assume that these cryptographic standard primitives are sufficiently protected against implementation attacks, such as side-channel or fault-injection attacks.

5 Configuration Integrity Checking

Integrity checks of FPGA configurations (especially for security-critical component) are essential to hamper replacement attacks. We now present two possible methods to check the integrity of a single device beyond automatically provided CRC integrity tests (1) at startup time and (2) periodically during operation to detect Single Event Upsets (SEU) as well as a malicious tampering at runtime. Our proposal for (1) described in Section 5.1 is more lightweight, does not interfere with operations and resource requirements of the main application and only demands for a slightly larger external memory to provide the storage for a multi-boot FPGA configuration. The proposal (2) discussed in Section 5.2 performs periodical checks while the main application is running at the cost of augmenting the main application with relevant cryptographic functions (possibly requiring a larger FPGA device).

5.1 Integrity Check at System Startup

Since devices of class C1 already include a facility to authenticate FPGA configuration at startup, we focus on FPGA class C2, illustrated by a Xilinx Spartan-6 XC6SLX75. In particular, this device supports encrypted multi-boot configurations, which can be loaded in sequence. For devices that do not support multi-boot, a similar behavior can be achieved by using the partial reconfiguration of FPGAs and internal configuration interfaces such as Xilinx' Internal Configuration Access Port (ICAP).

The first configuration (A) contains a cryptographic authentication and decryption (and optionally an identification) function that directly decrypts and authenticates the multi-boot image (B) right after configuration. The authentication is based on standardized symmetric authentication mechanisms, such as MACs involving standardized block ciphers or keyed hash functions like HMAC. Note that the bitstream must be decrypted before authentication to prevent simple substitution attacks [5]. To this end, AES decryption engine is implemented in the fabric of the FPGA (additionally to the bitstream decryptor in the FPGA control logic which is not directly accessible to user logic). Secret keys and/or checksums required for authentication and integrity check can be stored in the configuration bitstream, which must be encrypted and inaccessible in decrypted form, so as to ensure cryptographic security. Upon passing the authentication and integrity test of the configuration image, configuration (A) issues a command (e.g., for Spartan-6 FPGAs, the IPROG command sent via ICAP) to load the second part of multi-boot bitstream (B) into the FPGA which contains the main application. Figure 1 illustrates the FPGA multi-boot configuration with configuration (A) currently loaded into the FPGA.

Configuration (A) can also be extended to perform security functions beyond authentication and integrity tests. For example, (A) can implement a PUF instead of or in addition to the DeviceDNA device identification function (see Section 6.1). It might be also beneficial to implement or derive a chip-specific secret value (e.g., obtained from a PUF or other complex secret function) in configuration (A) but actually use it in the main application (B) (e.g., to perform periodical integrity checks as described in Section 5.2). This requires passing a secret value K_c between these two FPGA configurations (preserving confidentiality) that are loaded one after the other on the same FPGA. An possible solution is to store a common global secret K_G in both encrypted configuration (A) and (B) that masks the derived secret value K_c from configuration (A) by xor using $c = K_G \oplus K_c$. The masked value c can then be stored temporarily off-chip until configuration (B) is loaded decrypting the off-chip secret via $K_c = K_G \oplus c$ again. Depending on the FPGA, it may also be possible to transfer the secret K_c on-chip between configurations. This can be done with read/write registers of an FPGA's control logic which remain unchanged during FPGA configuration. For example, 5×16-bit GENERAL and READBACK registers

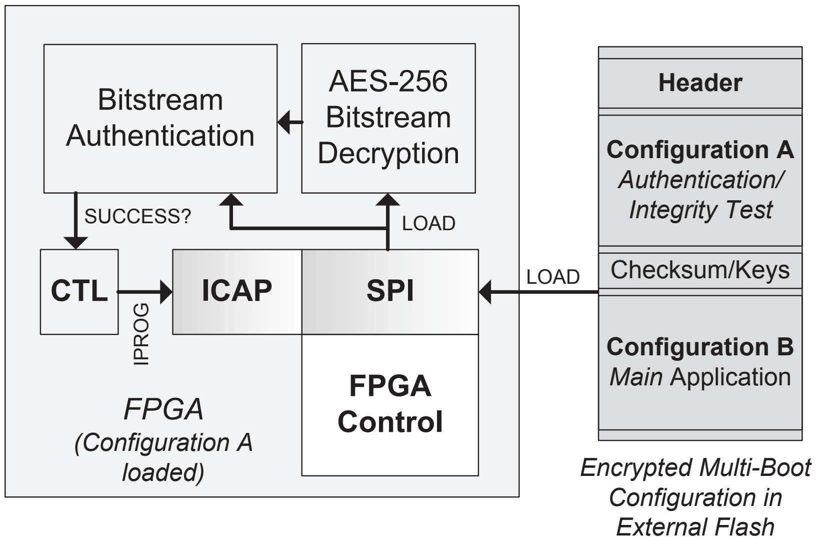

Fig. 1. Multi-boot configuration for bitstream authentication & integrity test.

can be used (*i*) to store individual words $K_{c,i}$ (or masked c_i) of a temporary 80-bit secret in configuration A and (*ii*) read them back after loading configuration B.

5.2 Periodic Integrity Check at Runtime

To prevent attacks that are applied *during* system operation, the one-time configuration authentication at system startup presented in Section 5.1 is not sufficient. The solution presented in this Section applies to FPGAs of classes C1,C2 and is related to the proposal by Drimer [5] but our solution is counter-less and thus supports asynchronous authentication. Most devices of these two classes allow a readback of the currently used configuration using external and internal ports from the fabric. On many FPGAs (such as Spartan-6), a readback using internal ports remains possible even if bitstream encryption is used (using the ICAP interface) and also does not interrupt a continuous system operation. Thus, bitstream authentication can run in parallel to the main application. It can be periodically called to read the current FPGA configuration using the ICAP interface, verifying its integrity and authenticity (see Figure 2).

6 Electronic Sealing of Multi-Chip Systems

Given a multi-chip systems, we want to prevent the injection of Trojan horses and replacement of individual components with knock-offs. Such tampering can be detected using mutual identification of individual chip-level components, performed within our proposed *electronic sealing* protocols. A generic solution to protect a single FPGA is provided by Altera's MAX II reference design [4] in which a CPLD is programmed to exchange cryptographic handshaking tokens with the FPGA device. However, this solution leaves unclear the storage of secret cryptographic key(s) and the applicability to multi-FPGA systems.

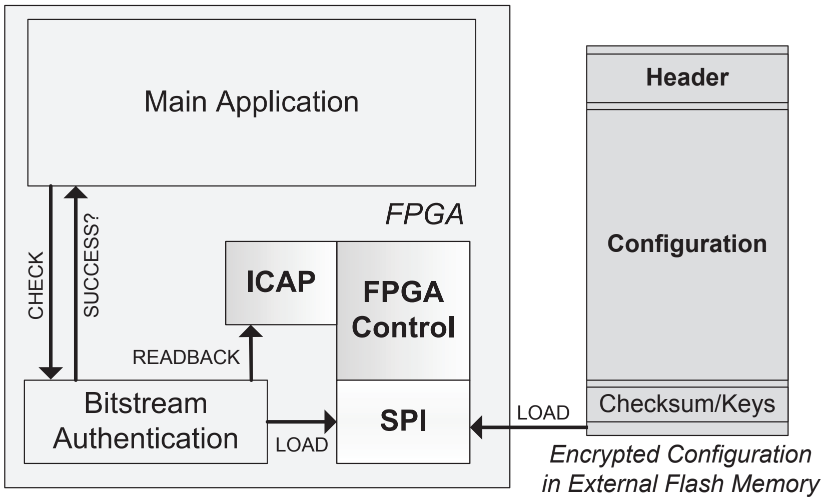

Fig. 2. Periodic integrity & authenticity checks of an FPGA configuration.

In this section, we propose three protocols for use with FPGAs of classes C1,C2 to enable electronic sealing based on an authenticated heartbeat for multi-FPGA systems[6] The first protocol (Section 6.1) employs a central chip or component to store cryptographic identifiers for all chips in the system, similar to Altera's solution [4] but more versatile. In Section 6.2, we further establish a decentralized method with which FPGAs perform a peer-to-peer authentication using a common global key without an additional security component. The third protocol (Section 6.3) performs electronic sealing in a ring topology without relying on a global key.

6.1 Symmetric Authentication & Central Security Module

For this protocol a central security module (CSM) is used to store individual keys K_i and identities ID_i of all deployed FPGAs. The ID_i of an FPGA can be derived using the DeviceDNA feature, if available. Alternatively, PUFs on FPGAs have evolved to a suitable method to derive device-specific identification numbers when no vendor support to distinguish devices is available. The individual (or global) secret keys K_i of each FPGA can be stored as encrypted part of the configuration in the system Flash or can be generated by another PUF. To reduce the number of logic resources occupied by secret key generation, a multi-boot configuration as shown in Section 5.1 can be used to generate secret keys beforehand in a separate configuration step. The CSM that manages all secrets can be either a CPLD located on-board, i.e., directly connected to the FPGAs via an internal bus, or it can be located at a remote back-end and connected via a network interface. The CSM needs to be initialized by a trusted party during system initialization

[6] While large FPGAs can already accommodate entire systems, multi-FPGA systems remain attractive in many cases, in part due to pricing of FPGA parts. For example, it is still cheaper to buy three largest Spartan-6 XC6SLX150 devices compared the cheapest, single Virtex-6 FPGA.

where all keys K_i as well as the corresponding identifier ID_i for all FPGAs are stored in a permanent, internal key storage. The CSM's internal key storage must not be read nor overwritten externally. Updating or removing a key-identifier pair within the CSM is an optional feature but can be useful to allow later chip updates or replacements due to defects. This can be realized by authenticating the updating party against the CSM using a master-key after which overwriting (parts of) the key storage with external data is enabled for a limited amount of time.

Once the keys K_i have been stored within the CSM, a mutual authentication scheme is periodically initiated by the CSM at runtime as part of the normal system operation. This *authenticated heartbeat* phase draws on [5] and enables electronic sealing at system runtime as follows:

1. The CSM sends a random number R_1 to FPGA_i after a globally specified time interval.
2. The FPGA generates a random number R_2 and responds with $Enc_{K_i}(R_1, R_2, \text{ID}_{\text{CSM}})$, where ID_{CSM} denotes a target identifier of the CSM and $Enc_{K_i}(X, Y)$ denotes the encryption (e.g. AES-CBC) of X concatenated Y using key K_i.
3. The CSM first decrypts the message and checks if decrypted R_1 and ID_{CSM} is correct. Then, it extracts R_2 and sends the message $Enc_{K_i}(R_1, R_2, \text{ID}_{\text{FPGA}_i})$ to the FPGA, where $\text{ID}_{\text{FPGA}_i}$ denotes the target identifier of the FPGA.
4. The FPGA decrypts the message and checks R_2 and $\text{ID}_{\text{FPGA}_i}$.

After such a successful mutual authentication of both parties, both chips continue operation. If the CSM fails to authenticate an FPGA chip, it may permanently deactivate the entire system, broadcast a warning message, etc. Likewise, the system designer/integrator determines the appropriate response if an FPGA chip fails to authenticate the CSM, e.g., self-deactivation of the FPGA by erasing its bit-stream.

6.2 Pairwise Authentication

The following protocol does not rely on a CSM. Instead, all FPGAs use a global key K_G for mutual authentication using identifiers ID_i. The global key K_G can either stored directly in an encrypted part of the FPGA configuration or it can be realized as follows: during system initialization in a trusted environment, a PUF on each FPGA is used to derive a chip-specific secret key K_{PUF_i}. Then a global key K_G is selected randomly and a mask value M_i is computed for each FPGA i as $M_i = K_{\text{PUF}_i} \oplus K_G$. This mask value is stored (unencrypted) in permanent memory. Only by combining the correct secret K_{PUF_i} (obtained individually per device from the PUF) with the device-specific mask value, the global key can be recovered during system runtime: $K_G = M_i \oplus K_{\text{PUF}_i}$.

A single global key K_G allows efficient key deployment during initialization and requires only limited storage (a single key/mask). However, by extracting the global key from an FPGA, an adversary can replace any FPGA in the system.

After initialization by system integrator, every FPGA periodically invokes a mutual authentication process as part of the normal system operation. This *authenticated heartbeat* phase draws on [5] and encompasses the following steps:

1. FPGA_i sends a random number R_1 to FPGA_j, $i < j$ over a shared link using a multi-master communication protocol (for synchronization, arbitration and data collision detection).
2. FPGA_j generates a random number R_2 and responds with $Enc_{K_G}(R_1, R_2, \text{ID}_{\text{FPGA}_i})$, where $\text{ID}_{\text{FPGA}_i}$ denotes the target identifier of FPGA_i.
3. FPGA_i decrypts R_1 and $\text{ID}_{\text{FPGA}_i}$ and if they are correct, it responds with $Enc_{K_G}(R_1, R_2, \text{ID}_{\text{FPGA}_j})$, where $\text{ID}_{\text{FPGA}_j}$ denotes the target identifier of FPGA_j.
4. FPGA_j decrypts and checks R_2 and $\text{ID}_{\text{FPGA}_j}$.

The system designer/integrator determines the appropriate response in case an FPGA fails authentication, e.g., an error message can be broadcast over the internal bus to prohibit communication with the FPGA that failed authentication.

6.3 Authentication based on a Ring Topology

Compared to the protocol presented in Section 6.2, FPGAs are arranged in a ring topology so that each FPGA only engages a symmetric authentication scheme with its two nearest neighbors. Hence, this protocol does not require storing a global key or a single individual key in each FPGA, but only the two individual keys K_i and K_{i+1} with $i \in 0, ..., n-1$ are stored in each of the n FPGAs. These keys can be stored or generated similarly as described in Section 6.2. Note that extracting a pair of keys K_i and K_{i+1} from an FPGA does not, in itself, allow faking another FPGA in the ring.

The main advantage of this authentication scheme compared to the scheme discussed in Section 6.2 is that only two individual keys K_i and K_{i+1} are stored in every FPGA. Hence, an adversary is not able to replace an arbitrary FPGA in the system after he has extracted the keys out of another FPGA. Further, the entire FPGA ring can be authenticated by sequentially verifying every pair of neighboring FPGAs.

7 Implementation Considerations

Our protocols and techniques draw on features common in modern FPGAs (e.g., JTAG TAP, ICAP, DeviceDNA), as well as standard cryptography available in both academic literature and industrial IP portfolios. Our solution in Section 5 requires an additional core for bitstream decryption (AES-CBC) and authentication (e.g., HMAC as a standardized keyed hash function), implemented in the programmable fabric. Similarly, electronic sealing requires encryption (AES) and a device-specific identifier (based on the DeviceDNA feature or PUFs). Rather than review well-known implementation details of state-of-the-art implementations, we cite relevant sources and provide an overview in Table 1. It is not our intent to provide an exhaustive list, but rather to show feasibility of our proposals and limit implementation overhead, even for small FPGA devices of classes C1,C2. For standard encryption and decryption, Drimer et al. describe small and fast AES cores with different modes of operation (including authenticated encryption) on Virtex-5 FPGAs [7]. Their smallest implementation requires 212 slices and 2 BRAMs using a fixed secret key. Logic slices in our reference FPGA (Spartan-6) are quite similar to those on Virtex-5, and implementation results generally agree with those reported by Drimer et al.

Table 1. Implementation cost (SLC = FPGA slices) and performance of required cryptographic primitives. Note that FPGA slices of Spartan-6 and Virtex-5 devices are very similar.

Primitive	**Device**	**Resources**	**Throughput**
AES [7]	Virtex-5	212 SLC/2 BRAM	1.7 GBit/s
HMAC/SHA-256 [12]	Spartan-6	110 SLC/1 BRAM	1.2 GBit/s
PUF (50 bit ID) [13]	Virtex-5	130 SLC	-

Optimized keyed hash functions such as HMAC-SHA-256 on FPGAs were described by McEvoy et al. in [14] and are also commercially available from Helion [12] taking 110 slices and 1 BRAM of a Spartan-6 FPGA. To implement the integrity check based on a multi-boot configuration and/or periodic runtime check described in Section 5, we combine the cryptographic primitives with an ICAP interface and a small-footprint Microblaze microprocessor soft core with fixed timer support which occupies 613 slices and 4 BRAMs on a Spartan-6. The complete solution requires less than 1000 Spartan-6 slices and 7 BRAMs. In a medium-sized Spartan-6 XC6SLX75 (see Section 5.1), this corresponds to a slice and BRAM utilization of 8.6% and 4.1%, respectively.

Protocols in Section 6 require fewer resources. To reliably distinguish FPGA devices (Section 6), one can use Xilinx DeviceDNA or any of FPGA-compatible PUF techniques [9, 11, 13]. Butterfly PUFs, for example, require 130 Virtex-5 slices to generate a 50-bit identifier [13]. Including overhead for control logic and the required encryption function [7], the resource consumption for electronic sealing is upper-bounded by 500 slices and 2 BRAMs (i.e., 4.3% of the slices and 1.2% of the BRAMs contained in the Spartan-6 XC6SLX75).

8 Conclusions

We proposed a comprehensive solution to guard against counterfeit integrated circuits in single- and multi-chip systems. In particular, *delayed logic design* in terms of FPGAs limits the time interval during which successful attacks can be mounted against a hardware system. Since modern FPGAs already provide bitstream encryption as a basic feature, we proposed in this work how to enable configuration authentication during system startup and runtime as well how to establish electronic sealing in multi-chip systems using only symmetric cryptography. We presented two reasonable techniques to verify the integrity and authenticity of FPGA configuration for devices that provide bitstream encryption. First, we proposed a multi-boot configuration that cryptographically verifies its own integrity before the actual main application is loaded. The second technique performs periodic configuration readback within the FPGA to prevent manipulations of security functions during runtime. A second major contribution deals with electronic sealing of multi-chip systems. We developed three efficient protocols based on a central security module, peer-to-peer- and ring-based communication that enable mutual authentication of system components to detect tampering or unauthorized replacement of protected devices. Finally, we evaluate the implementation overhead for common FPGA parts and show that it is moderate.

References

1. China to make foreign firms reveal secret info. Yomiuri Shimbun (September 2008), http://www.yomiuri.co.jp/dy/business/20080919TDY01306.htm
2. FBI Concerned About Implications of Counterfeit Cisco Gear. Slashdot (April 2008), http://hardware.slashdot.org/article.pl?sid=08/04/22/1317212
3. Adee, S.: The Hunt For The Kill Switch. IEEE Spectrum 45(5), 34–39 (2008)
4. Altera Corporation: FPGA Design Security Solution Using MAX II Devices. White Paper (September 2004), ver. 1.0 at http://www.altera.com/literature/wp/wp_m2dsgn.pdf
5. Drimer, S.: Authentication of FPGA bitstreams: why and how. In: Applied Reconfigurable Computing. vol. 4419, pp. 73–84 (March 2007)
6. Drimer, S.: Volatile FPGA design security – a survey (v0.96) (April 2008), http://www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf
7. Drimer, S., Güneysu, T., Paar, C.: DSPs, BRAMs, and a pinch of logic: Extended recipes for AES on FPGAs. ACM Trans. Reconfigurable Technol. Syst. 3, 1–27 (January 2010), http://doi.acm.org/10.1145/1661438.1661441
8. Gross, G.: US, Canadian agencies seize counterfeit Cisco gear. The Industry Standard (2008), http://slashdot.org/article.pl?sid=08/02/29/1642221
9. Guajardo, J., Kumar, S., Schrijen, G., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: CHES. vol. 4727, p. 63. Springer (2007)
10. Güneysu, T., Möller, B., Paar, C.: Dynamic Intellectual Property Protection for Reconfigurable Devices. In: ICFPT 2007. pp. 169–176 (2007)
11. Güneysu, T.: Using Data Contention in Dual-ported Memories for Security Applications. Journal of Signal Processing Systems 30 December 2010, 1–15 (2010), dOI 10.1007/s11265-010-0560-z
12. Helion Technology: Tiny Hash Core Family for Xilinx FPGA. Data Sheet (2010), http://www.heliontech.com/downloads/tiny_hash_xilinx_datasheet.pdf
13. Kumar, S., Guajardo, J., Maes, R., Schrijen, G., Tuyls, P.: Extended abstract: The butterfly PUF protecting IP on every FPGA. In: IEEE International Workshop on Hardware-Oriented Security and Trust (HOST 2008). pp. 67–70 (2008)
14. McEvoy, R.P., Crowe, F.M., Murphy, C.C., Marnane, W.P.: Optimisation of the SHA-2 family of hash functions on FPGAs. In: Emerging VLSI Technologies and Architectures. vol. 00 (2006)
15. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science 297(5589), 2026–2030 (2002)
16. Parelkar, M.M.: FPGA security – bitstream authentication. Tech. rep., George Mason University (2005), http://ece.gmu.edu/courses/Crypto_resources/web_resources/theses/GMU_theses/Parelkar/Parelkar_Fall_2005.pdf
17. Roy, J.A., Koushanfar, F., Markov, I.L.: Extended abstract: Circuit CAD tools as a security threat. In: HOST. pp. 65–66 (2008)
18. Seamann, G.: FPGA bitstreams and open designs (April 2000), http://web.archive.org/web/20050831135514/http://www.opencollector.org/news/Bitstream/
19. Simpson, E., Schaumont, P.: Offline hardware/software authentication for reconfigurable platforms. In: CHES. vol. 4249, pp. 311–323 (2006)
20. Trimberger, S.: Trusted design in FPGAs. In: Design Automation Conference (June 2007), http://videos.dac.com/44th/papers/1_2.pdf
21. Wollinger, T., Guajardo, J., Paar, C.: Security on FPGAs; state of the art implementation and attacks. In: ACM Trans. Embedded Comp. Sys. (TECS) (2004)