An Electronic Sealant for Secure Multi-chip Systems: Reducing Vulnerability to Malicious Alterations

Igor Markov, Kai Schramm and André Weimerskirch
imarkov@umich.edu,{kschramm,aweimerskirch}@escrypt.com

ABSTRACT

Abstract: electronic hardware security is rapidly gaining traction in academia and marketplace, given (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. Significant systems have already succumbed to malicious attacks. To this end, our work shows how to dramatically reduce vulnerability to Trojan-horse injection and in-field component replacement. Key ideas include delayed logic design, on-the-fly chip authentication, and electronic sealing of a multi-chip system, which automatically enforces the system configuration detected upon power-up and bans further modifications. Possible applications include systems-in-package and distributed multicomponent systems in automotive applications.

1. INTRODUCTION

The well-documented spread of fake goods has recently swept through the electronics industry. 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, several branches of US military, FAA and FBI. 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. Another member reported 19 cases involving 97,000 chips" [15]. According to EE Times, 61% of the goods shipped to the U.S. by one Chinese distributor in Shenzhen were found to be counterfeit. Counterfeit ICs typically mimick the originals, but lag in performance or embedded memory capacity, thus providing a cheaper and almost indistinguishable replacement. In some cases legitimate chips are "upgraded" to more expensive variants by misleading labeling.

Reportedly, the FBI is concerned that counterfeit equipment may be state-sponsored to aid in accessing otherwise secure systems [22]. To this end, *IEEE Spectrum* pointed out [1] 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. Bruce Schneier — a prominent cryptologist and security expert — recently articulated the potential impact of kill-switches that could remotely disable cars and airplanes, etc [20]. With semiconductor manufacturing increasingly outsourced, much of it overseas, almost any chip may be altered. On October 11, 2008, the *Wall Street Journal* reported [7] the discovery of "a highly sophisticated credit-card fraud ring that funnels account data to Pakistan from hundreds of grocery-store card machines across Europe... The account data have been used to make repeated bank withdrawals and Internet purchases... The scheme uses untraceable devices inserted into credit-card readers that were made in China. The devices selectively send account data by a wireless connection to computer servers in Lahore, Pakistan, and constantly change the pattern of theft so it is hard to detect, officials say." Another recent report describes two touch-screen voting computers in Arkansas that consistently allocated votes to a wrong election [27]. To this end, the DARPA-sponsored "Trust" program seeks techniques to detect and disable such alterations, and ensures rigorous evaluation by funding work on concealment of Trojan horses in ICs.

In the past, altering an integrated circuit required painstaking reverse engineering. However, in some cases such alteration can be greatly simplified. For instance, the "obligatory accreditation system for IT security products" in China [26] demands source code from hardware manufacturers starting in May 2009, making reverse engineering unnecessary.

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

Our proposal seeks to dramatically *reduce the vulnerability of electronic systems by decreasing the time interval during which successful component replacement is possible*. It encompasses

- *Delayed logic design* through the use of FPGAs, which prevents most malicious alterations before the system integrator receives the chip.
- *On-the-fly chip authentication* 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.

The remaining part of this paper is organized as follows. Section 2 covers necessary background. Our techniques for reducing the vulnerability of multi-chip systems to attacks are first introduced in Section 3 and described in more detail in Section 4, followed by the discussion of required hardware modifications in Section 5. Possible attacks and countermeasures are analyzed in Section 6. Concluding remarks are given in Section 7.

2. BACKGROUND

Until a few years ago, IT security research focused predominantly on full-fledged computers and networks. However, the increasing adoption of embedded systems raises security risks for cars, telephones, smartcards and other widespread applications. Techniques are required to not only secure data, but also provide safety and reliability, in a wide variety of industry and consumer application contexts. Contrary to assumptions made in traditional security research, an adversary trying to undermine an embedded system will in most scenarios have direct physical access to it. However, embedded systems, constrained by considerations of cost, size and battery life, rely on very limited computational resources and can therefore support only relatively simple cryptography. The use of techniques to provide tamper-resistance is limited by financial considerations as well.

Cryptographic communications can be classified into *symmetric* and *asymmetric*. Symmetric cryptography offers the ability to securely and confidentially exchange messages between parties over an untrusted channel, if the parties have previously shared a secret. Furthermore, symmetric cryptography can provide data integrity and authentication without non-repudiation. On the other hand, asymmetric (also called *public-key*) algorithms, provide additional services such as digital signatures and key distribution over unsecure channels. Asymmetric ciphers are often 2-3 orders of magnitude slower than symmetric ciphers, but the two types are usually combined in hybrid encryption and authentication protocols, such as implemented by SSL used in Web transactions.

2.1 Security Services

Cryptography offers the following security services.

- *Confidentiality* is a service used to keep the content of information accessible only to authorized entities. It includes both the protection (i.e., encryption) of data messages transmitted between nodes in a network as well as protection against traffic-flow analysis.
- *Integrity* is a service which thwarts unexpected manipulation of data (including the modification, deletion, creation, and delaying or replaying of transmitted messages).
- *Authentication* is a service that identifies the sender of a message. Data transferred over an insecure communication channel can be authenticated regarding the origin, time, date and location of origin, data content, etc.
- *Non-repudiation* is a property of a communication protocol which prevents both the sender and the receiver from denying previous commitments or actions.

2.2 Symmetric-Key Cryptography

Symmetric-key cryptographic protocols form basic building blocks of any security system which requires confidentiality. They normally encrypt bulk messages that are exchanged between two systems, using the same key for encryption and decryption. The exchange of symmetric keys among parties is typically performed in a secure environment, over a secure channel or achieved by using public-key distribution protocols. Block ciphers — a common type of symmetric ciphers — split messages into fixed-length blocks and encrypt each block using a fixed-length key. The algorithm Rijndael [3] developed by Daemen and Rijmen was selected in November 2001 from a large set of algorithms as the new Advanced Encryption Standard (AES) [23] that replaces the outdated Data Encryption Standard (DES). AES supports a block size of 128 bits and variable key sizes of 128-bits, 192-bits, and 256-bits, offering a choice of security level (and computation overhead) [23].

Mode	Application
ECB	Electronic Code Book: Message is divided into blocks and each block is encrypted separately.
CBC	Cipher-Block Chaining: Message is divided into blocks, each block is encrypted and each block of plaintext is XORed with the previous ciphertext block before being encrypted.
CBC-MAC	Cipher-Block Chaining Message Authentication Code: Message is divided into blocks and each block of plaintext is XORed with the previous ciphertext block before being encrypted. The ciphertext of a single block, i.e. an input plaintext of arbitrary has the length length is mapped to an output of fixed length. In particular, the CBC-MAC mode can be used to compute a fingerprint value.
OFB	Output Feedback: Block cipher is transformed into a stream cipher.

Table 1: Various block cipher modes.

2.3 Asymmetric-Key Cryptography

Symmetric cryptography algorithms are typically used to encrypt bulk data at high speed. However, a secret key must be known to the sender and receiver before messages can be exchanged. Moreover, if n nodes within a network need to establish pairwise secure communication links, then $\frac{n\cdot(n-1)}{2}$ keys will be necessary. Public-key cryptography, introduced in 1976 by Diffie and Hellman [4] addresses these shortcomings by (1) distinguishing the encryption key and making it public, (2) making the decryption key private. The private key is never transmitted but rather kept secret at all times, whereas the public key is available to everyone and can be shared. Public-key cryptography provides important services, such as key distribution protocols (e.g., Diffie–Hellman key exchange) and key transport protocols (e.g., via RSA) over insecure channels. Practical applications employ a combination of symmetric-key and public-key cryptosystems (*hybrid cryptosystems*), where a public-key algorithm performs initial key exchange, and symmetric-key algorithm performs high-speed encryption and authentication. Public-key algorithms also offer *digital signatures*, which are similar to handwritten signatures in identifying and confirming the sender (authentication), even if the sender tries to deny authorship (non-repudiation). A digital signature is typically a data block appended to a message. It is generated using the private key of the sender (which is not available to anyone else) and is verified by the receiver using the corresponding public key. Practical public-key algorithms can be divided into three families:

- Algorithms based on the *discrete logarithm problem (DLP)* over finite fields such as the Diffie–Hellman key exchange protocol, El Gamal encryption, and the Digital Signature Algorithm (DSA).
- Algorithms based on the *integer factorization problem* such as the RSA cryptosystem [18].
- Algorithms based on *elliptic curve cryptography (ECC).* Elliptic curve cryptosystems [14] are the most recent family of practical public-key algorithms which have gained wide acceptance including standardization due to short key lengths[11]. Elliptic curve cryptosystems are based on the DLP over elliptic curve groups.

Table 2 compares key lengths of a symmetric cipher, RSA and ECC for equal levels of security against exhaustive-search attacks. It is widely believed that ECC will substitute RSA in the future.

Symmetric	ECC	RSA	Strength
64 bit	128 bit	700 bit	short
80 bit	160 bit	1024 bit	medium
128 bit	256 bit	3072 bits	long
256 bit	521 bit	15360 bits	very long

Table 2: Strength levels of cryptographic keys.

2.4 Component Identification

Component identification (CI) provides protection against counterfeits and is intended for systems assembled from electronic components [24, 25]. Typical mechanisms either (1) use a central module that verifies the authenticity of all electronic components, or (2) arrange for components to verify each other. If a counterfeit is detected, components stop their service and/or output an error message. CI is implemented today in theft protection systems for automotive vehicles in order to prevent the removal and resale of navigation systems, airbags and other valuables. Many applications use similar techniques, e.g., SIM cards for mobile GSM phones are locked to a specific handset, and some printers only accepts cartridges from a single manufacturer. However, known CI mechanisms are not robust enough to counter malicious chip modification, and they often rely on application-specific mechanisms, such as locking cellular phone service.

2.5 FPGAs

Field-programmable gate arrays (FPGAs) are integrated circuits where the functions of individual logic gates and interconnect can be defined by the customer after the chip is manufactured and sold [13]. A traditional FPGA chip consists of identical programmable logic elements, linked by interconnect segments with programmable junctions. Each configuration bit is located next to a logic element or interconnect junction it controls. A blank FPGA is *programmed* by sending a configuration *bit-stream* to the chip through a dedicated interface. The bit-stream is formed by CAD tools based on circuit specifications (this process includes logic synthesis, placement, routing, etc) and *encrypted* before it is sent to the chip. Decryption is performed by a cryptographic module, typically AES.

The cost of FPGA chips ranges from single dollars to hundreds of dollars. Most FPGA chips (Xilinx, Altera) can be re-programmed many times, but FPGAs that can be programmed only once (Actel) are common in military applications because they are less susceptible to electromagnetic interference. FPGAs have traditionally lagged behind application-specific integrated circuits (ASICs) in performance and power consumption, but this gap is now decreasing due to architectural improvements in FPGAs and due to recent trends in semiconductor manufacturing [12, 13]. One particular improvement is the inclusion of discrete highly-optimized ASIC-style circuit modules, such as multipliers, that are used by many applications. As we point out in Section 6, these modules are particularly vulnerable to attacks. FPGAs have been successfully used in the design of secure hardware [6], often programmed to implement so-called *soft processors* that execute cryptographic primitives as software programs. Such embedded systems are discussed in detail in [10], including a variety of attacks and countermeasures.

3. REDUCING IC AND SYSTEM VULNERABILITY TO ATTACKS

A key reason for the dramatic rise in chip piracy and concerns about subversion is the recent separation of design and manufacturing, which were traditionally performed by the same company. More generally, individual companies now tend to specialize with narrow focus, which leads to longer supply chains, ending with OEM companies that integrate components into systems. Convincing solutions to security challenges must account for these specifics.

The general *security objective* pursued in our work is to prevent altered chips from successfully running in a computer system. Our techniques rely on certain hardware infrastructure for authentication and on initialization at a secure facility, then make sure that this infrastructure is not replaced or altered *afterwards*. We also consider the possibility of subversion *before* initialization, e.g., by means of Trojan horses, and make that difficult through the use of FPGAs, as explained below. We make the non-trivial assumption that CAD tools for FPGAs, used at the secure facility, are trusted and do not cause unintended side-effects [19].

3.1 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. In applications with small volumes, FPGAs are more economical than ASICs. Additionally, FPGA design tool-chain tends to require less effort and smaller professional expertise, reducing NRE costs. 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. *Pre-programming* attacks on blank FPGAs are discussed in Section 6, where countermeasures are offered.

3.2 On-the-fly Chip Authentication

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 (or actual configuration bits) and use this fingerprint to authenticate the chip. The fingerprint is checked periodically using a challenge-response protocol, to hamper replay attacks with chips reporting fake fingerprints. In principle, even a single modified wire should affect the FPGAs fingerprint, but challenge-response protocols may not detect every such change immediately if response to each challenge is limited in size.

3.3 Electronic Sealing

Component chips are field-programmed by a trusted system integrator, which defeats many replacement and Trojan-horse attacks prior to system integration. Further, to make use of chips that report their own identity, two integration challenges must be solved.

- The identity of each chip must be checked reliably, ruling out subversion by replacing the checking components.
- Since the identity of any one chip is known, one must rule out subversion by fake or maliciously altered chips that lie about their identity.

We address these challenges by a protocol for *electronically sealing* a multi-chip system at power-up. Rather than establish a separate hardware entity to periodically verify the identities of each chip, we task the chips themselves with verifying their neighbors. Since in most real-world systems chips communicate through busses and must collaborate to accomplish the system's mission, changes

in chip's identity should not go undetected. To respond to such an event, unaltered chips would lock up and paralize the entire system, thus refusing the newly altered component. Alternatively, if bus communications are encrypted, tying cryptographic keys to chip signatures would keep altered chips incommunicado.

We have assumed so far that each chip truthfully reports its identity. To overcome this assumption, we now strengthen the system-consistency requirement as follows. Not only each chip's identity is periodically checked by every other connected chip, but each chip must also prove that it still shares secrets that were exchanged at power-up (and never transmitted since).

To demonstrate the impact of the *stronger system consistency requirement*, consider two identical copies of a component chip. One chip is used in the system as it is assembled and initialized, and the other is left as a spare without being initialized. Upon power-up, the chip starts reporting its identity, and the entire system goes through the *electronic sealing* sequence which validates each chip's authenticity. Since the spare chip does not carry this information, using it as a replacement will violate strong consistency and disable the system (such *stronger system consistency* may or may not be useful in a specific system).

4. DETAILS OF PROPOSED PROTOCOLS

The three protocols introduced below are based on symmetric cryptography and support periodic, on-the-fly authentication of individual chips. They seek to detect changes in FPGA fingerprints due to manipulation of configuration bits. The first protocol presented in Section 4.1 relies on a central security module (CSM), while the second protocol in Section 4.2 is based on peer-to-peer authentication of a pair of FPGAs using a common global key. The third protocol outlined in Section 4.3 performs authentication in a ring topology without relying on a global key.

4.1 Symmetric Authentication with a Central Security Module (CSM)

The fingerprints over randomly chosen sections of the bitstreams of n FPGAs are to be verified during the normal operation of the system. The idea of challenge-response verification of memory contents based on randomly selected bits was proposed for embedded software by Seshadri et al. in [21] and is here applied to FPGAs configuration. As depicted in Figure 1, a central security module (CSM) is required which stores the individual keys of all FPGAs deployed and their bitstream. The CSM can be either 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 an online network interface. It is presumed that the CSM has sufficient memory to store the bitstreams of each FPGA used in the system. If the storage requiremens exceed the resources of the CSM, data compression can be used. FPGA fingerpints over randomly subsets of the bitstream will be requested by the CSM during authentication.

The protocol consists of two phases. The first *initialization* must occur within a secure environment, e.g., at the trusted vendor site before shipment for production use. Here n random keys K_i are generated by the CSM and transferred to the FPGAs as plaintext.

1. The CSM generates a random key K_i for each FPGA and stores it in a secure, non-volatile memory.
2. The CSM transfers the random key K_i as plaintext to each FPGA, where it is stored in a secure, non-volatile memory.
3. Each FPGA returns with its bit-stream code to the CSM, which stores the code in secure, non-volatile memory.

The CSM's internal key storage must not be read nor overwritten externally. Likewise each FPGA also stores its key in internal, secure memory which cannot be read nor overwritten. The following *lock-bit* functionality is optional: following a mutual authentication between CSM and FPGA, an unlock mechanism can overwrite the secure memory of an FPGA, by using an encrypted *unlock message* initiated by the CSM. This unlock message must also contain the new key in an encrypted form.

Once the keys K_i have been distributed, a mutual authentication scheme is periodically initiated by the CSM at runtime as part of the normal system operation. This *authenticated heartbeat* phase encompasses the following steps:

1. The CSM sends a random number R_1 to $FPGA_i$.
2. The FPGA generates a random number R_2 and responds with $ENC_{K_i}(ID_{CSM}, R_1, R_2)$, where ID_{CSM} denotes a target identifier of the CSM, e.g., a network ID.
3. The CSM first decrypts the message and checks if decrypted R_1 is correct. Then, it extracts R_2 and sends the message $ENC_{K_i}(ID_{FPGA_i}, R_1, R_2, \mathcal{S}_{FPGA_i})$ to the FPGA, where ID_{FPGA_i} denotes the target identifier of the FPGA and $\mathcal{S}_{FPGA_i}$ denotes a selected, random subset of the bitstream over which $FPGA_i$ computes a fingerprint.
4. The FPGA decrypts the message and checks decrypted R_2. It then sends the message $ENC_{K_i}(ID_{FPGA_i}, R_1, R_2, \mathcal{F}_{FPGA_i})$, where $\mathcal{F}_{FPGA_i}$ is the fingerprint over subset $\mathcal{S}_{FPGA_i}$ of the bitstream.
5. The CSM decrypts the message and checks the fingerprint.

The use of random numbers, called *nonces* in cryptography, defeat replay attacks in symmetric and bi-directional challenge-response protocols. Likewise randomly selected areas of the bitstream are fingerprinted to complicate replay attacks, since this would require that a faked or manipulated FPGA stores the entire bitstream of the original FPGA or at least a subset of the bitstream. Only after a successful mutual authentication of both parties the FPGA sends its encrypted fingerprint to the CSM. 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 and erasure of its bit-stream.

The removal of FPGAs can be easily realized by deleting the corresponding key and fingerpint in the internal memory of the CSM. To add a new FPGA, one must repeat initialization in a secure environment.

4.2 Symmetric Peer-to-Peer Authentication

The following protocol does not rely on a CSM. Instead, every FPGA stores an equal global key K_G which is used for mutual authentication. Hence, every FPGA is able to mutually authenticate any other FPGA in the system. Using a single global key K_G allows an efficient key deployment process during the initialization phase and requires only limited storage resources. On the other hand an adversary who extracts the global key out of an FPGA is able to replace any FPGA in the system.

The protocol consists of two phases. *Initialization* is performed within a secure environment, e.g., at the trusted vendor site. During this phase a server generates a single global key K_G and deploys this key in each FPGA, so that each FPGA can authenticate any other FPGA used in the system.

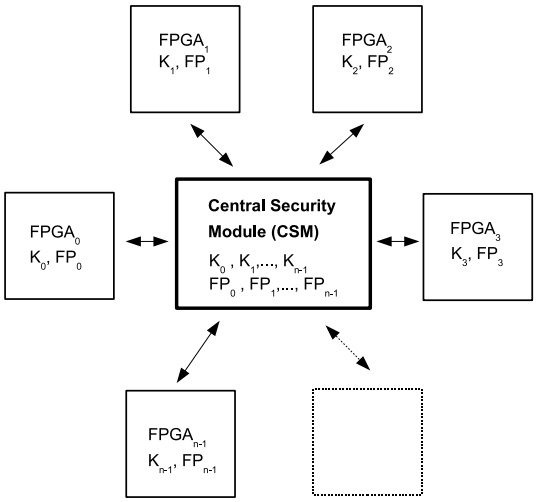

Figure 1: Mutual authentication scheme with FPGAs and a central security module (CSM).

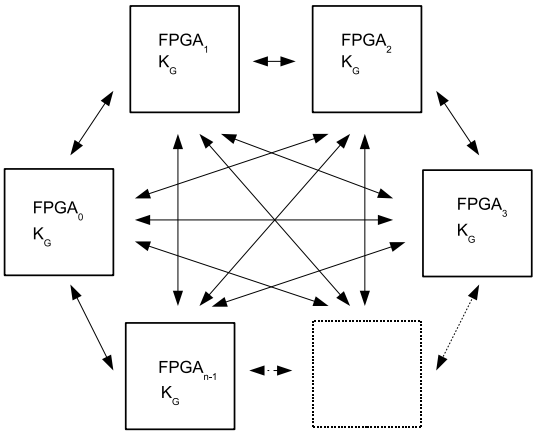

Figure 2: Peer-to-peer authentication scheme with FPGAs.

The secure memory where each FPGA stores the key K_G must not be read or overwritten externally. Once the key has been injected into every FPGA, every FPGA periodically engages a mutual authentication process as part of the normal system operation. This *authenticated heatbeat* phase encompasses the following steps:

1. $FPGA_i$ sends a random number R_1 to $FPGA_j$, with $i \neq j$.
2. $FPGA_j$ generates a random number R_2 and responds with $ENC_{K_G}(ID_{FPGA_i}, R_1, R_2)$, where K_G denotes the global symmetric key and ID_{FPGA_i} denotes the target identifier of $FPGA_i$, e.g., its network ID.
3. $FPGA_i$ decrypts R_1 and ID_{FPGA_i} and if they are correct, it responds with $ENC_{K_G}(ID_{FPGA_j}, R_1, R_2)$, where ID_{FPGA_j} denotes the target identifier of $FPGA_j$, e.g., its network ID.
4. $FPGA_j$ decrypts R_2 and ID_{FPGA_j} and checks them.

The system designer/integrator determines the appropriate response in case one FPGA chip fails authentication, e.g., an error message can be broadcast over the internal bus to prohibit communication with the FPGA that failed authentication. The topology of the peer-to-peer authentication scheme using a global key K_G is shown in Figure 2.

4.3 Authentication based on a Ring Topology

The following protocol also does not rely on a CSM. Compared to the the protocol presented in Section 4.2 FPAGs are arranged in a ring topology, so that each FPGA only engages a symmetric authentication scheme with its two nearest neighbors (see Figure 3). Hence, this protocol does not require that a global key or a single individual key is stored 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. Hence, an adversary who has successfully extracted a pair of keys K_i and K_{i+1} out of an FPGA may be able to counterfeit this particular FPGA but not any other FPGA in the ring.

The protocol consists of two phases. *Initialization* is performed within a secure environment, e.g., at the trusted vendor site. During this phase a server generates n random keys K_i with $i \in \{0, ..., n-1\}$ and deploys a pair of individual keys K_i and K_{i+1} in each FPGA, so that each FPGA can authenticate its two neighbors.

The secure memory where each FPGA stores the key pair K_i and K_{i+1} must not be read or overwritten externally. Once the keys have been distributed, every pair of FPGAs periodically (at runtime) engages a mutual authentication process as part of normal system operation. This *authenticated heatbeat* phase encompasses the following steps:

1. $FPGA_i$ sends a random number R_1 to $FPGA_{i+1}$, with $0 \leq i < n-1$ (only $FPGA_{n-1}$ sends a random number R_1 to $FPGA_0$ to close the ring).
2. $FPGA_{i+1}$ generates a random number R_2 and responds with $ENC_{K_i}(ID_{FPGA_i}, R_1, R_2)$, where K_i denotes a pre-shared symmetric key known to $FPGA_i$ and $FPGA_{i+1}$. The term ID_{FPGA_i} denotes the target identifier of $FPGA_i$, e.g., its network ID.
3. $FPGA_i$ decrypts R_1 and ID_{FPGA_i} and if they are correct, it responds with $ENC_{K_i}(ID_{FPGA_{i+1}}, R_1, R_2)$, where $ID_{FPGA_{i+1}}$ denotes the target identifier of $FPGA_{i+1}$, e.g., its network ID.
4. $FPGA_{i+1}$ decrypts R_2 and $ID_{FPGA_{i+1}}$ and checks them.

The main advantage of this authentication scheme compared to the scheme discussed in Section 4.2 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. Furthermore, the authenticity of the entire ring of FPGAs can be determined by sequentially verifying two neighboring FPGAs until all FPGAs have passed the mutual authentication scheme.

5. HARDWARE REQUIREMENTS

The CSM-based protocol we described may be more straightforward conceptually, but would imply greater overhead if CSM is implemented as a separate chip. Alternatively, CSM can be bundled with one of the chips considered indispensable for the system

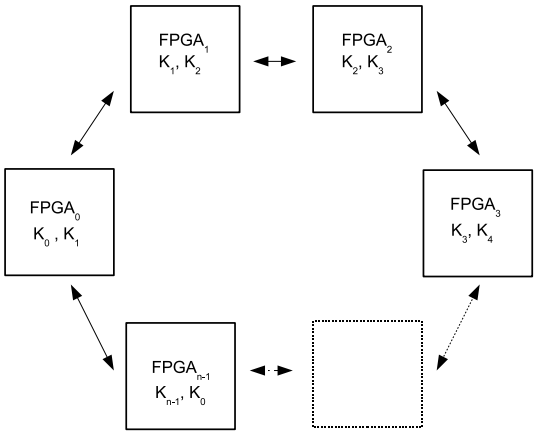

Figure 3: Mutual authentication scheme with FPGAs arranged in a ring topology.

in question, in which case the cumulative overhead for the entire system can be smaller due to centralized non-volatile memory.

Our proposed symmetric protocols can use AES. To this end, high-end FPGAs from Altera and Xilinx already include optimized AES cores next to the programmable fabric, so as to decrypt a bitstream. For example, the new Cyclone III LS family by Altera features an AES-256 encryption engine, the ability to deactivate its JTAG debugging interface and the detection of an intentional or unintentional change of configuration random access memory (CRAM) bits [28].

A complete AES core supporting encryption and decryption at medium speed requires approximately 3000 logic gates (medium speed is acceptable in our protocols, which do not encrypt large amounts of data). In terms of chip area, this would be on the order of 1% for a modern ASIC or embedded CPU, and less than 0.1% for SoC chips and high-end CPUs. We note that AES can play multiple roles in our protocols, e.g., its CBC mode should be used during the challenge-response protocols. Its AES-CBC-MAC mode can compute fingerprints, and the counter mode can be used as a cryptographically secure PRNG, by encrypting an increasing counter and using a random, secret key. Such reuse of the AES core helps keeping area (and cost) overhead low. Power overhead in percent often matches area overhead, but we can lower it further by observing that AES is not used continually, but rather during short, periodically repeating intervals. This calls for *clock gating*, which would decrease both clock power and dynamic power in signal wires by 1-2 orders of magnitude. Unfortunately, relying on pre-existing AES cores in our protocols would not be compatible with the *delayed logic design* principle of Section 3.1 and erode hardware security by exposing such FPGAs to pre-programming attacks discussed in Section 6. Therefore, stronger security requirements call independent implementations of cryptographic ciphers in reconfigurable logic, even if this increases overhead. Advantages include the ability to change the symmetric block cipher and its implementation on demand. Table 3 lists a variety of alternative ciphers with compact implementations. Furthermore, they can be implemented as software programs for soft processors executed by the FPGA [6, 10]. FPGA vendors provide such processors (Nios, Nios2, MicroBlaze) as intellectual property to their customers. If the IC in question already integrates such a processor for other uses, then area and power overhead of our protocols will be minimal, and no circuit-level features (such as clock gating) will be required.

Block cipher	Gate count	Ref.
AES	3400	[5]
DES	2309	[17]
PRESENT-128	1886	[2]
PRESENT-80	1570	[2]
Grain (stream cipher)	1294	[9]

Table 3: Various block ciphers and their known minimum chip size requirements in terms of logic gate counts. LUT counts for gate-level FPGA-based implementations are 3-4 times lower.

Another contributor to area overhead is non-volatile memory used to store symmetric keys of the FPGAs. While specific numbers depend on the type of non-volatile memory used, limiting the maximum number of FPGAs (to, say, 20) and choosing a medium-grade key length (say, 80-128 bits) can keep the overhead low.

A True Random Number Generator (TRNG) can be useful instead of an AES-based PRNG. Such facilities are already available in recent chips, such as Sun Niagara2, and exhibit very low overhead — less than 0.1% of the chip area [16].

Some of our proposed protocols rely on access to the configuration bits of an FPGA. Ideally, one would like random access to SRAM bits (or anti-fuses) of a running FPGA, or at least a cryptographically secure hash function of those bits. Since any hash value can be faked by tampering with the chip, random access would provide better verification. But random access would also undermine bitstream encryption and intellectual property protection. While it is theoretically possible to design FPGAs which would offer encrypted random access to their configuration bits, such a feature may require significant area and power overhead. Instead, it is more practical to query the EEPROM from which the FPGA reads its configuration upon power-up. Since EEPROMs already provide random access, such a feature would require little more than interconnect and several multiplexors. The downside is that tampering with the FPGA itself does not affect EEPROM and will not be detected when EEPROM is queried. Therefore, most of our protocols do not require this feature.

Implementations of our protocols will require I/O pins for inter-chip communication. Given their bandwidth, all communications can be serialized through 1-2 pins on each chip. For FPGAs with hundreds of I/O pins, the overhead is going to be low. Another I/O optimization takes into account the regular schedule of authentication messages to multiplex authentication traffic through pins used by the digital system for other purposes.

6. ATTACKS AND COUNTERMEASURES

We now discuss the robustness of proposed protocols and explain how they address common cryptographic attacks.

Traffic analysis records spikes in communication activity through insecure channels and other communication patterns, such as repetitions. In our protocols, traffic intensity of *authenticated heartbeat* does not change at runtime. Repetitions are hidden through the use of nonces and PRNGs, as discussed below. The initialization phase of our protocols is performed at a secure facility, which should prevent the recording of any information for further analysis.

In **replay attacks**, messages are recorded and played back at later time, possibly without decryption. For example, a ground control system may send a command to a satellite requesting that its orbit be lowered by 1km. No matter how well such a message is encrypted, a simple replay attack could crash the satellite by replaying the radio transmission many times. Replay attacks are easily defeated by randomized changes of context, which can

be implemented using nonces (as discussed in earlier sections) or PRNGs. This way, semantically identical messages must look different when encrypted; and published analyses of existing protocols show no easy attacks. Such cryptographic enhancements also complicate traffic attacks.

Man-in-the-middle attacks typically involve an impostor with access to the communication medium, who acts as one of the intended parties so as to perform key exchange and subsequently decrypt all data intended for someone else. Such attacks are typically addressed through authentication with digital signatures, and this solution is compatible with out protocols. Additionally, if such an attack targets our protocols, it must be perpetrated during the initialization phase, which would be difficult because we require initialization to be performed at a secure facility.

Side-channel attacks observe power usage, heat dissipation, EM radiation, or other unintended emissions during system operation, then deduce cryptographic keys or perform traffic analysis. Since our protocols communicate only through standard cryptographic primitives, such as RSA and AES, their susceptibility to side-channel attacks is similar to that of other applications that rely on these primitives (and we will benefit from any improvements in RSA, AES implementations). Furthermore, we perform key exchanges at a secure facility, which physically prevents collection of side-channel information.

Attacks before programming seek to alter a blank FPGA, e.g., by embedding a Trojan horse, so as to expose the programmed chip to further attacks. We distinguish the following categories of pre-programming attacks.

(a) interfering with FPGA programming, e.g., by setting and locking some of the configuration bits, or altering the wires used for programming,

(b) adding side channels,

(c) modifying discrete circuit modules enclosed with the FPGA, such as high-speed multipliers and encryption modules.

One countermeasure against pre-programming attacks is to test each blank FPGA before use. Circuit testing for discrete modules is well-known, but focuses on manufacturing faults rather than on subtle subversion. For example, a high-speed multiplier could be sabotaged to produce incorrect results only for some rare inputs, that can be sent during a later attack. Therefore, sensitive calculations should not rely on discrete modules, but implement the same functionality in programmable logic, which can be protected as shown below. Alternatively, one can continually check the results produced by a high-speed module using a slower computation implemented in programmable logic. For example, every 32-bit arithmetic operation can be verified modulo $p = 13$, $p = 19$, etc (with the modulus selected at runtime at random).

Reconfigurable logic can be tested by programming it with randomized test configurations and then checking responses (assuming SRAM-based, rather than antifuse-based, FPGAs). Fooling this procedure without knowing the tests would be difficult, except that it does not check for added side channels, e.g., connecting a flip-flop to a radio transmitter for eavesdropping. Fortunately, this and other attacks directed at reconfigurable logic do not survive randomized circuit layout.

As explained in Section 2, the values of specific configuration bits for FPGA programming are determined by CAD tools during logic synthesis, circuit placement and wire routing. Given the regularity of FPGA circuit-fabrics, and the fact that their logic elements and wire segments are interchangeable, the same functionality can be implemented in many different ways. In particular, by randomizing placement and routing algorithms for FPGA, one makes it impossible to predict the configuration of individual logic elements of interconnect junctions, or their role in a given circuit.[1]

In a pre-programming attack, the perpetrator may have access to a language-based description of the integrated circuit design that will be mapped to the FPGA. He or she will then want to modify the blank FPGA to plant a Trojan horse. However, not knowing the mapping of the IC design into specific logic elements and wire segments makes it challenging to plant a Trojan horse, since specific signals cannot be accessed.[2] This mapping cannot be known in advance because it is created *after* the blank FPGAs are delivered. Running randomized CAD tools for each newly delivered batch of blank FPGAs will make pre-programming attacks of type (a) and (b) above very difficult. We emphasize that there is a very large number of possible configurations to implement the same functionality, in part because there is usually a significant fraction (10% and higher) of unused logic elements and wire segments, giving us the freedom to reassign those elements and segments that are used.

Fake fingerprints could help an attacker replace an existing chip with their own. This is why we use them only in the first of our three proposed cryptographic protocols. Since we trust a built-in feature of an FPGA to truthfully report a hash value of its bitstream, it is fairly easy to alter a blank FPGA to produce any given signature that does not correspond to the actual configuration, as long as the attacker knows the desired signature. A straightforward countermeasure is to exchange full fingerprints only during secure initialization, and otherwise transmit them encrypted. Additionally, the secure initialization step can include a secret-sharing protocol, so that the secrets are not transmitted during runtime but rather encrypt communications, defeating chip replacement. Another countermeasure is to verify the contents of FPGAs using a challenge-response mechanism, such that knowing previous responses is not sufficient to predict future responses.

While the attacker may gain physical access to the chips to read their secure memories, technologies currently exist for tamper-proof packaging of ICs, such that any attempt to open the package will result in the IC being dissolved in acid. Tamper-proof packaging can also prevent a range of other attacks, and is therefore recommended. However, in practice, one would buy tamper-proof FPGAs from the supplier, which would expose them to potential attacks before programming.

A potential attack could proceed as follows. The attacker manages to secretly undermine tamper-proof packaging in blank FPGAs *before* they are shipped and programmed, and then obtains physical access to running chips *after* programming and initialization. With appropriate equipment, the attacker may then read chip signatures and shared secrets, and prepare a replacement chip that would report just the right signatures and secrets. The latter requires altering the FPGA design and manufacturing new chips – a difficult task, which may require infiltrating the fabrication plant.

Depending on the application, we could further complicate such attacks by requiring that the electronic system in question run non-stop. In other words, any interruption in communications or power supply will be interpreted as an attack, shutting the system down, and requiring a new round of secure initialization. This feature is trivial to implement by replacing secure non-volatile key storage by conventional SRAM shielded from inspection by X-rays.

By making technical attacks very difficult, we may nudge the

[1] In practice, FPGA placement often uses graph partitioning algorithms and simulated annealing — both heavily randomized. As long as their pseudo-random number generators start with different seeds on every run, the results will be very different.

[2] I/O ports can be easily accessed, but one can encode, encrypt and randomize I/O communications to defeat I/O-based attacks.

attackers to focus on human factors. At that point, those responsible for the electronic system in question should be more concerned about their employees, maintenance procedures, and physical security of facilities, than about increasingly arcane scenarios for purely electronic attacks.

7. CONCLUSIONS

We proposed a comprehensive solution to guard against counterfeit integrated circuits in individual chips and in multi-chip electronic systems. The chip-level solution relies on key properties of FPGAs and uses delayed programming as a means to thwart typical Trojan horse attacks. However, FPGAs make it easier to create replacement chips, therefore we propose an electronic sealant mechanism that thwarts such replacement.

As discussed in Section 4 we presented three protocols based on symmetric key cryptography which can be implemented in order to validate the authenticity of the FPGAs used in the system. The first protocol presented in Section 4.1 relies on a central security module (CSM), which verifies the authenticity of the individual secret key of each FPGA and the authenticity of its bitstream. Hence, an adversary who knows the secret key of an FPGA in the system and who tries to substitute the FPGA with a counterfeit variant would still have to store the entire bitstream of the original FPGA. The second protocol presented in Section 4.2 is based on a random, mutual authentication of a pair of FPGAs using a common global key. Both the initial key deployment and the authentication messages during run-time are very efficient. However, an adversary able to extract the global key can counterfeit any FPGA used in the authentication scheme. Finally, the third protocol presented in Section 4.3 uses an authentication scheme based on ring topology which does not rely a global key, but on a pair of individual keys in each FPGA. Hence, an adversary able to extract the pair of keys out of a particular FPGA can only counterfeit this FPGA but not others.

Our work can be compared to the recent major initiative in the PC industry known as Trusted Platform Module (TPM). TPM is currently implemented in many PCs as a separate chip that includes secure memory to store software keys. TPM also checks the integrity of the system and would block Windows OS if the hardware configuration changes significantly. However, TPM may not detect subtle changes in electronic components, which is the goal of our work. The TPM design focuses on PC software security (viruses, Trojans), but not hardware.

In current implementations, the TPM is detached from the CPU and relies on the CPU to deliver trustworthy data. However, it offers no mechanisms to ensure a trustworthy channel between CPU and TPM, and is vulnerable to man-in-the-middle attacks, for example. It is expected that next-generation CPUs will include TPM, which would boost the level of security by far. Such an extension would bring TPM closer to our techniques, although TPM will remain focused on software and overall PC configuration. In contrast, our methods offer stronger protection to the integrity of distributed electronic hardware and can be used in a variety of applications, such as automotive and building-control systems.

8. REFERENCES

[1] S. Adee, "The Hunt for the Kill Switch," *IEEE Spectrum*, May 2008. HTTP://WWW.SPECTRUM.IEEE.ORG/MAY08/6171

[2] A. Bogdanov et al., "PRESENT: An Ultra-Lightweight Block Cipher", in CHES 2007.

[3] J. Daemen and V. Rijmen,"AES Proposal Rijndael", in *First Advanced Encryption Standard (AES) Conf.* 1998.

[4] W. Diffie and M. E. Hellman, "New Directions in Cryptography", *IEEE Trans. on Information Theory* IT-22 (1976), 644-654.

[5] M. Feldhofer, S. Dominikus, and J.Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES algorithm", in CHES 2004, LNCS, vol. 3156, pp. 357Ű370.

[6] G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, R. Vaslin. "Reconfigurable Hardware for High-Security/ High-Performance Embedded Systems: The SAFES Perspective," *IEEE Trans. on VLSI*, vol. 16, no. 2, pp. 144-155, February 2008.

[7] S. Gorman, "Fraud Ring Funnels Data From Cards to Pakistan," *the Wall Street Journal*, 10/11/2008.

[8] G. Gross, "US, Canadian agencies seize counterfeit Cisco gear," *The Industry Standard*, 02/29/2008. HTTP:// SLASHDOT.ORG/ARTICLE.PL?SID=08/02/29/1642221

[9] M. Hell, T. Johansson, W. Meier, "Grain - A Stream Cipher for Constrained Environments", eSTREAM, 2006, WWW.ECRYPT.EU.ORG/STREAM/CIPHERS/GRAIN/ GRAIN.PDF

[10] T. Huffmire, B. Brotherton, N. Callegari, J. Valamehr, J. White, R. Kastner, T. Sherwood, "Designing Secure Systems on Reconfigurable Hardware," *ACM Trans. on Design Autom. of Elec. Syst.*, vol. 13, no. 3, article 44, July 2008.

[11] IEEE. IEEE P1363-2000: IEEE Standard Specifications for Public Key Cryptography, 2000. HTTP://STANDARDS. IEEE.ORG/CATALOG/OLIS/BUSARCH.HTML

[12] I. Kuon, R. Tessier, J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation* 2(2): 135-253 (2007).

[13] I. Kuon, J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Trans. on CAD of Integrated Circuits and Systems* 26(2): 203-215 (2007)

[14] Miller, V., "Uses of Elliptic Curves in Cryptography", in *Advances in Cryptology*, CRYPTO '85, H. C. Williams, Ed., LNCS 218, Springer-Verlag, pp. 417-426.

[15] R. Merritt, "Web site aims to limit channels for fake chips," *EE Times*, 04/21/2008. HTTP://WWW.EETIMES.COM/ SHOWARTICLE.JHTML?ARTICLEID=207401126

[16] U. M. Nawathe et al., "An 8-Core 64-thread 64b powerefficient SPARC SoC," In *ISSCC*, pp. 108Ű611, 2007. http://www.opensparc.net/opensparc-t2/index.html

[17] A. Poschmann, G. Leander, K. Schramm, C. Paar, "A Family of Light-Weight Block Ciphers Based on DES Suited for RFID Applications", Workshop on RFID Security 2006.

[18] Rivest, R. L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Comm. ACM* 21, (February 1978), 120-126.

[19] J. A. Roy, F. Koushanfar, I. L. Markov, "Circuit CAD Tools as a Security Threat," *HOST* 2008, pp. 65-66.

[20] B. Schneier, "I've Seen the Future, and It Has a Kill Switch," *Wired*, 05/26/08. HTTP://TECH.SLASHDOT.ORG/ ARTICLE.PL?SID=08/06/29/1147247

[21] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: SoftWare-based ATTestation for Embedded Devices", Proceedings of the IEEE Symposium on Security and Privacy, 2004, HTTP://SPARROW.ECE.CMU.EDU/ ~ADRIAN/PROJECTS/ SWATT.PDF

[22] "FBI Concerned About Implications of Counterfeit Cisco Gear," *Slashdot*, 04/22/2008. HTTP://HARDWARE. SLASHDOT.ORG/ ARTICLE.PL?SID=08/04/22/1317212

[23] U.S. Department of Commercea and the National Institute of Standard and Technology, "FIPS PUB 197, Specification for the Advanced Encryption Standard (AES)", 11/2001. HTTP://CSRC.NIST.GOV/ENCRYPTION/AES/

[24] A. Weimerskirch, C. Paar, and M. Wolf, "Cryptographic Component Identification: Enabler for Secure Inter-vehicular Networks", 62nd IEEE Vehicular Technology Conference, September 25-28, 2005.

[25] A. Weimerskirch, K. Höper, C. Paar, and M. Wolf, "Component Identification: Enabler for Secure Networks of

Complex Systems“, Applied Cryptography and Network Security (ACNS) 2005, June 7-10, 2005.

[26] “China to make foreign firms reveal secret info,” *Yomiuri Shimbun*, 09/19/2008 HTTP://WWW.YOMIURI.CO.JP/DY/BUSINESS/20080919TDY01306.HTM

[27] K. Zetter, “Arkansas Election Officials Baffled by Machines That Flipped Race”, *Wired News*, 05/29/08. HTTP://BLOG.WIRED.COM/27BSTROKE6/2008/05/ARKANSAS-VOTING.HTML

[28] Altera, “Cyclone III FPGA Family”, HTTP://WWW.ALTERA.COM/PRODUCTS/DEVICES/CYCLONE3/CY3-INDEX.JSP?WT.MC_ID=CL_PR_AL_NE_TX_J_271/