Offline Hardware/Software Authentication for Reconfigurable Platforms

Eric Simpson and Patrick Schaumont

Virginia Tech, Blacksburg VA 24060, USA
{esimpson, schaum}@vt.edu

Abstract. Many Field-Programmable Gate Array (FPGA) based systems utilize third-party intellectual property (IP) in their development. When they are deployed in non-networked environments, the question raises how this IP can be protected against non-authorized use. We describe an offline authentication scheme for IP modules. The scheme implements mutual authentication of the IP modules and the hardware platform, and enables us to provide authentication and integrity assurances to both the system developer and IP provider. Compared to the Trusted Computing Platform's approach to hardware, software authentication, our solution is more lightweight and tightly integrates with existing FPGA security features. We are able to demonstrate an implementation of the authentication scheme that requires a symmetric cipher and a Physically Unclonable Function (PUF). In addition to the low hardware requirements, our implementation does not require any on-chip, non-volatile storage.

1 Introduction

The latest generation of Field Programmable Gate Arrays (FPGAs) can accommodate complex systems containing embedded hardware and software. While they are often used in a constrained, non-networked environment, their configuration presents a valuable piece of intellectual property that merits protection. Our contribution is an offline mutual authentication scheme for both the hardware and software configuration of a reconfigurable platform. The mutual authentication involves the FPGA chip manufacturers, who provide a standard security module in each of their FPGAs, and the IP providers, who commit to an identity for each release of their software. An FPGA system developer combines chips and IP components in their product. Using the hardware identity provided by the chip manufacturers and software identity committed to by the IP providers, they are able to construct a product where hardware and software components can authenticate each other.

In this section, we briefly review the roles that the chip manufacturers, IP providers, and system developers play in the authentication scheme, along with what it means to have a software or hardware identity. Also discussed is the meaning of software in an FPGA and the role it plays in modern FPGA design.

use the direct method instead

Fig. 1. Parties involved in modern FPGA design

1.1 Securing Intellectual Property in Modern FPGA Design

The rapidly increasing design capacity of FPGAs enables more complex and bigger designs than ever before. Many current models of FPGAs not only support traditional hardware design, but also have the ability to run embedded software. The software in an FPGA executes in either an embedded hardcore processor like the PowerPC, or in a softcore [1] that is synthesized with the rest of the hardware design. These software modules are often developed and distributed by third-party IP Providers. While design protection in FPGAs has been available for some time in the form of configuration encryption [2], this technique is ineffective at protecting third-party intellectual property and software modules. We point out two key issues.

The first issue is that IP-methodologies require additional authentication. As shown in Fig. 1, system developers design their product with a plug-and-play methodology in which they adopt third-party intellectual-property components (IP) for integration onto a chip. This IP can possibly come from multiple vendors in the case of so-called System-on-Chip design (SoC). The result is an intriguing multi-level authentication problem. At one level, system developers would like to authenticate the IP they are running (Fig. 1, A), and at another level the IP providers would like to authenticate the system into which they are integrated (Fig. 1, B). In this paper, we will specifically consider the integration of software IPs onto an FPGA platform, but it is understood that the need for authentication in IP-methodologies is generic.

The second issue is that current FPGA security mechanisms have a limited scope, focused on the hardware configuration. Bitstream encryption [3] for example will enforce bitstream privacy and integrity, but it will not protect the software running on the processors configured in the FPGA. In the case of third-party IP, and specifically in the case of software IPs, additional protection mechanisms are required.

We present a solution to the above to problems, in the form of a protocol and an architectural extension for FPGA-based design. We also demonstrate a sample implementation and report on the complexity and performance of this implementation.

The system setup that we are considering for our protocol development is shown in Fig. 2. A hardware platform, designed by a System Developer, will be configured into an FPGA. The System Developer will also use third-party software IPs that execute on top of the platform. The System Developer can apply bitstream encryption to protect the hardware configuration in the FPGA, but an additional hardware-software authentication mechanism is needed to protect the software IPs.

use the direct method instead

Fig. 2. Levels of Authentication

The outline of this paper is as follows. In Section 2 we describe the protocol for enrolling the hardware and software identities in the authentication scheme. Then, in Section 3 we discuss the protocol for distributing mutually authenticated IP to system developers. After discussing the authentication protocol in Sections 2 and 3, we analyze the security properties in Section 4. Finally, a low-cost implementation of the offline, hardware, software authentication scheme is presented in Section 5.

2 Enrollment Protocol

The enrollment phase involves three parties: the chip manufacturers, IP providers and a trusted third-party that is used to store and communicate identity information among the participants. As shown in Fig. 3, the enrollment phase is composed of two communication channels. One channel is used by the chip manufacturer to communicate hardware identities to the trusted third-party, and the other channel is used by the IP provider to communicate software identities. The definition of hardware and software identities is given in the following sections.

use the direct method instead

Fig. 3. Enrollment Phase

2.1 Hardware Enrollment

In this phase, the manufacturer would like to distribute FPGAs that have the ability to securely run third-party IP. To enable their customers to securely run third-party IP, the FPGA manufacturers implement a standard security module in each chip. This security module contains two distinct hardware blocks:

1. PUF - Used for hardware authentication and key generation
2. Block Cipher - Used for symmetric encryption and software authentication

The PUF is a device that maps inputs (challenges) to outputs (responses). The mapping from a challenge to a response is determined by the physical properties of the chip it is implemented in. Therefore, an identical PUF circuit implemented on two different chips will result in different responses for the same challenge. Several implementations for PUFs have been reported in literature [4, 5].

After building their FPGAs, the manufacturer enrolls them in the authentication scheme by sending each chip's identification information to the trusted third-party. The identity is composed of two data items:

$HW\#$: Public, unique 128-bit value that identifies the chip

$\overrightarrow{CRP}$: Private list of challenge, response pairs produced by the chip

To communicate the identity for each chip, the manufacturer opens an authenticated and secure link to the trusted third-party (over SSL, SSH, etc.). Over the authenticated and secure link, the manufacturer sends:

$$\mathit{Manufacturer} \longrightarrow \mathit{TTP} : HW\#, \overrightarrow{CRP}$$

2.2 Software/IP Enrollment

The enrollment of IP providers in the authentication scheme allows system developers to verify the integrity and authenticity of the software they are running. The identification information the IP provider sends to the trusted third-party is composed of two data items:

$IP\#$:	Public, unique 128-bit value that identifies the name and version of the intellectual property
$Hash(SW, IP\#)$:	Public hash of the IP# and software that the IP is composed of

Like the chip manufacturer, the IP provider opens a secure and authenticated link to the trusted third-party. For each version or release of their software, the IP provider sends:

$$\mathit{IPP} \longrightarrow \mathit{TTP} : IP\# \ , Hash(SW, IP\#)$$

Since the IP Provider only has to commit to a hash of their IP, they don't have to trust the third-party with the actual software. Also, the IP provider doesn't have to make any changes to their development process to enroll in the authentication scheme. They simply commit to a version and hash for each software release. There is no need to embed watermarks [6], or any other identification information in their software.

3 Authenticated IP Request and Distribution

Once the system developer has purchased FPGAs that have been enrolled in the authentication scheme, the developer can request authenticated IP from the trusted third-party. The request and distribution of an authenticated hardware-software configuration requires four messages per IP module. The first three messages, involving the trusted third-party, form the online phase of the protocol. The fourth message does not require the trusted third-party and forms the offline phase of the protocol.

The messages of the online phase are exchanged over a standard secure, and authenticated link.

First, some definitions of the symbols used in the protocol:

$Nonce$:	Number used once, a unique token used to ensure the freshness of a message
C_{ttp}, R_{ttp} :	Challenge, response pair used by the trusted third-party to communicate the IP authentication and integrity data to the system developer
R_{ip} :	Response used by the IP provider to encrypt and package their software for the target hardware platform
C_{ip} :	Challenge that the target hardware can use to generate the R_{ip} used to encrypt the software.

3.1 Request and Distribution Messages

$$(1)\ SYS \longrightarrow TTP : IP\#, HW\#, Nonce$$

$$(2)\ TTP \longrightarrow SYS : IP\#, HW\#, C_{ttp}, \{IP\#, Hash\left(SW, IP\#\right), C_{ip}, Nonce\}_{R_{ttp}}$$

$$(3)\ TTP \longrightarrow IPP : IP\#, HW\#, Nonce, R_{ip}$$

$$(4)\ IPP \longrightarrow SYS : IP\#, HW\#, \{length, Nonce, SW\}_{R_{ip}}$$

use the direct method instead

Fig. 4. Distribution Phase

3.2 Messages Explained

As shown in Fig. 4, the request and distribution of an authenticated hardware-software configuration requires four messages between the system developer, trusted third-party and the IP provider.

1. Message #1 is the system developer IP request to the trusted third-party.

2. Message #2 is sent by the trusted third-party to the system developer. The message is encrypted using a CRP that the security module can generate and use to decrypt the message. The message contains the requested IP's identity and integrity information. Also contained in this message is the challenge that can be used by the security module to generate the response used by the IP provider to encrypt their software.

3. Message #3 is the trusted third-party forwarding the system developer's request for IP to the IP provider. This message contains the response that the IP provider will use as the key to encrypt their software.
   After message #3, the trusted third-party is no longer involved in the mutual authentication, and the protocol becomes off-line. At this point, the IP provider can now securely package their software for a unique hardware identity. While this package could be sent over the network, it could also be in the form of a ROM chip that is given to the system developer.

4. Once the system developer has received message #4 from the IP provider, this data can be merged with authentication information contained in message #2. At this point, the system developer has the following information:

   (a) $C_{ttp}, \{IP\#, Hash\,(SW, IP\#)\,, C_{ip}, Nonce\}_{R_{ttp}}$
   (b) $\{length, Nonce, SW\}_{R_{ip}}$

   Part (a) of the message contains the necessary information to validate the authenticity and integrity of the software, and the software is assured that only the target hardware can decrypt the IP contained in part (b). Since the security module is the only one that can generate the required responses to decrypt the data, the merged message can then be saved to insecure storage by the system developer. This IP containing message can then be validated, loaded, and run by the offline FPGA indefinitely.

While the initial request and distribution of the messages involve active communication, the last stage of verification is able to be performed in an offline context. It is important that the last authentication stage can be performed offline because many systems are deployed in non-networked environments. This is an important distinction from protocols that require interactive zero-knowledge proofs [7, 8], or active dialogue to perform authentication [9].

4 Analysis

The protocol is secure against cheating attempts by either the system developer, or IP provider.

4.1 Tampering with Data from the TTP

Since the response from the TTP in message #2 is encrypted with a random response, only the developer who possesses the target hardware with a valid security module, will be able to decrypt the message. Also, in order to create a fake encrypted portion of the message, the attacker needs to know the mapping from C_{ttp} to R_{ttp}, which is infeasible due to the properties of the PUF. Invalid messages in this step can be detected when the plaintext $IP\#$ doesn't match the $IP\#$ contained in the encrypted portion of the message.

4.2 Tampering with Data from the IP Provider

There are two hurdles an attacker must overcome to tamper with the software received by the system developer in Message #4. The first hurdle is that the attacker must not only know the C_{ip} the TTP sent in Message #2, but also the mapping from C_{ip} to R_{ip}. In addition, any modifications to the data will be detected when the $Hash(SW, IP\#)$ doesn't match the expected data, or the $Nonces$ don't agree.

4.3 Collusion Scenarios

Using an example from today's marketplace, it is interesting to look at the possible sources of cheating and fraud between the various parties. The parties are defined as follows:

System Developer:	Customer designing a product or prototype
TTP:	Fabless company that designs the actual FPGAs
IP Developer:	Value-added seller that produces IP for a specific company's FPGAs
Chip Manufacturer:	Third-party company that manufacturers the TTP designed chips

The company that designs the actual FPGAs is the trusted third-party in this scenario, because they have an incentive to be trustful to both the system developers that purchase their chips, and the IP providers that develop for their FPGAs. Since the FPGA designer is in the business of selling chips, it is desirable to have a a complete design portfolio of IP components available for their FPGAs. By designing the security module into their chips, they can assure IP providers that their chips provide a secure environment from IP piracy. Therefore, if the FPGA designing company wants to stay in business, they must stay trustworthy to both their chip purchasing customers and IP providers. For example, if they distributed a CRP that enabled a system developer to pirate an IP module, the FPGA designer would lose all trust and likely see an exodus of IP providers from their platform.

By having the trusted third-party directly generate the list of CRPs, the authentication system also protects the FPGA designer from a chip manufacturer who overbuilds and directly sells the FPGA designer's product. Since the overbuilt chips will not be in the TTP database, these chips will not be authorized to run authenticated, third-party IP. In addition, when a system developer uses a counterfeit chip, the FPGA designer will directly notice the counterfeits.

4.4 Implementation Practicality

One system issue is the ability of the TTP to store the authentication data. As an example, for each chip the TTP must store the hardware ID along with a CRP list. Making a rough estimate of 1,000 CRPs for each chip, this results in a storage requirement of 250KB per chip. Therefore, a TTP would be able to store the authentication data for a 1,000,000 chips on a single 250GB disk. Given that many of today's workstations have 250GB worth of storage, this is certainly a reasonable storage requirement.

Also, implementing IP authentication into the development process is a reasonable consideration as well. Given the fact that FPGA designers already distribute development software to system developers, implementing IP authentication into these tools would be of similar complexity. Therefore, integrating an authenticated IP distribution scheme into the development process does not appear to be an unreasonable task.

5 Results

The security module for the reconfigurable platform was developed on a Xilinx Spartan-3 FPGA. The security module block diagram is presented in Fig. 5. The authenticated IP is stored in an off-chip memory module and loaded into the FPGA on power-up. The security module's Protocol Controller is responsible for detecting the presence of authenticated IP on the input lines and coordinating the load of authenticated IP. The transfer from the external store into the security module is done over an 8-bit bus, with full-handshaking. Once the data is inside the security module, all data is passed over a shared 128-bit bus between the Protocol Controller, AES module and the PUF. All processing is done in a fully parallel manner, such that the AES block, PUF and IO can overlap execution. Therefore, after one IP block has completed authentication and is being loaded into FPGA for execution, another can be undergoing authentication, while the next IP block is simultaneously being loaded into the security module.

use the direct method instead

Fig. 5. Security Module block diagram

Whether the security module is loading authenticated IP or generating CRPs is determined by a leading opcode. Currently the opcode is one byte, with two predefined opcodes. One opcode is used to instruct the security module to generate a CRP, while the other causes IP to be loaded through the security module. Therefore, authenticated IP is stored in the following three-part format:

1. $Opcode_{load}$
2. $C_{ttp}, \{IP\#, Hash\,(SW, IP\#)\,, C_{ip}, Nonce\}_{R_{ttp}}$
3. $\{length, Nonce, SW\}_{R_{ip}}$

An important note is that the authentication scheme is not limited to loading a single authenticated IP module. The FPGA can load an arbitrary number of IP components through the security module. Even while running, the system can be configured to load new IP modules, or to swap old ones out of the system.

Generating a $\overrightarrow{CRP}$ requires the following message:

1. $Opcode_{CRP}, seed, \#$ of pairs to generate

Where the *seed* is a random number used to seed the PUF, and the number of pairs to generate is a 64-bit integer. The CRP list is generated by the following:

$$C_0 = PUF\,(PUF\,(seed)) \tag{1}$$
$$R_0 = PUF(C_0) \tag{2}$$
For $i = 1$ to $\#$ of pairs to generate
$$C_i = R_{i-1} \tag{3}$$
$$R_i = PUF(C_i) \tag{4}$$

The system was designed and simulated using the the GEZEL language and environment [10]. In addition to decrypting incoming IP modules, the AES cipher is also used to compute hashes as in Cohen's AES-hash NIST proposal [11]. We have not yet built a PUF implementation, but have simulated its behavior using another AES block with a fixed key.

Since GEZEL is a cycle-based hardware description language, most of the work and simulation was completed before translating the GEZEL code to VHDL. The translation from GEZEL to VHDL was done by the `fdlvhd` tool provided by the GEZEL environment. After translation, the VHDL was synthesized and mapped to a Spartan-3 FPGA using Xilinx's toolchain. The results are summarized in Table 1.

Table 1. Security Module Synthesis

Component	**HW**	**Slices**	**Speed**
Protocol Controller (Input)	Spartan-3	169	202 MHz
Protocol Controller (Output)	Spartan-3	142	187 MHz
AES	Spartan-3	2046	124 MHz
Simulated PUF	Spartan-3	2025	124 MHz

As expected, the hardware requirements are dominated by the AES and simulated PUF. The important note though, is the low complexity and requirements of the Protocol Controller. Couple this protocol model with some of literature's low cost symmetric cipher and PUF implementations and this offline mutual authentication scheme not only fills a needed security gap in modern FPGA design, but is also low cost.

5.1 Related Work

While AES was chosen as the symmetric cipher, and a simulated PUF was used, it's important to note that our offline hardware, software authentication scheme only requires a single symmetric cipher, and a single PUF. This is in contrast to the Trusted Computing Group's (TCG) authentication scheme that requires the following components be implemented in their trusted platform module (TPM) [12, 13]:

Non-volatile storage for storing various keys and authorization data

True random-bit generator for key and nonce generation

SHA-1 Engine for computing signatures

RSA Key Generation for at least a 2048-bit modulus

RSA Engine for digital signatures and encryption/decryption

Compared to the recommended components for the TPM, our security module is more practical to implement as a standard module in FPGAs. Also, the TPM requirement for secure on-chip non-volatile storage isn't likely to be met by current reconfigurable platforms.

In [14] the authors present a solution to the shortcomings of the TCG specifications with regards to sealed data. Our work is orthogonal to [14] because their focus is primarily on data that has been sealed to a particular TPM. This results in a different problem domain because the data we are protecting is not unique. While the data that an individual produces can be unique and irreplaceable, the protected software delivered to our security module is neither unique, nor irreplaceable. Instead, the software is able to duplicated by the IP provider on demand. Therefore, when switching to a new FPGA, the system developer can simply make a request to the TTP for another copy of the IP. The previously distributed software doesn't have to be exported in a secure way because it can be duplicated by the IP provider. If our authentication scheme was to be extended to protecting not only hardware and software, but also unique data, the ideas from [14] would be good addition for managing the secure data.

Other relevant work was done in [15] where the authors implemented a secure and cost-optimized verson of a TPM for hand-held devices. The work describes an architecture to cleanly implement the Trusted Computing Group specifications [12] in a hand-held context. By focusing on the ability to implement the TCG specifications though, this implementation has the same requirements as a standard TPM. A notable exception to the standard TPM implementation is that the authors discuss the possibility of using a PUF based system to avoid the need for onchip non-volatile memory to store secrets. Other relevant work that our security module could benefit from is in the [15] authors work on secure debugging interfaces and methods.

6 Conclusions

The use of intellectual-property components in FPGA-based design leads to new and particular security requirements. The protection of the configuration bitstream itself is insufficient to cope with multiple IP originators, and moreover it does not offer adequate guarantees with respect to IP protection. Our results show that a protocol can be designed that offers the required protection while meeting the constraints of a small embedded and offline implementation. We also don't require a major modification of the design process. In fact, our protocol can be made backward compatible with existing approaches for downloading FPGA bitstreams.

We believe that our scheme is applicable to situations outside of FPGA design, and are presently investigating its use in the context of other implementation technologies, as well as in the context of different forms of IP, including data and hardware IP blocks.

References

1. Moyer, B.: Using softcore-based FPGAs to balance hardware/software needs in a multicore design. Embedded System Design Magazine (2006)
2. Feng, J.: FPGA design security. ECN Magazine (2006) 23–24
3. Inc., X.: Using bitstream encryption. Handbook of the Virtex II Platform (2003)
4. Gassend, B.: Physical Random Functions. Master's thesis, Massachusetts Institute of Technology (2003)
5. Suh, G.E., O'Donnell, C.W., Sachdev, I., Devadas, S.: Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions. SIGARCH Comput. Archit. News **33**(2) (2005) 25–36
6. Kahng, A.B., Lach, J., Mangione-Smith, W.H., Mantik, S., Markov, I.L., Potkonjak, M., Tucker, P., Wang, H., Wolfe, G.: Watermarking techniques for intellectual property protection. In: Design Automation Conference. (1998) 776–781

7. Feige, U., Fiat, A., Shamir, A.: Zero-knowledge proofs of identity. J. Cryptol. **1**(2) (1988) 77–94
8. Bellare, M., Palacio, A.: Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In: CRYPTO. (2002) 162–177
9. Otway, D., Rees, O.: Efficient and timely mutual authentication. Operating Systems Review **21**(1) (1987) 8–10
10. Schaumont, P., Ching, D.: GEZEL homepage. http://rijndael.ece.vt.edu/gezel2 (2006)
11. Cohen, B., Laurie, B.: AES-Hash. NIST: Modes of Operation for Symmetric Key Block Ciphers (2001)
12. Group, T.C.: TCG Specification Architecture Overview. (2004)
13. DoCoMo, N., IBM, Corporation, I.: Trusted Mobile Platform Hardware Architecture Description. (2004)
14. Kuhn, U., Kursawe, K., Lucks, S., Sadeghi, A.R., Stuble, C.: Secure Data Management in Trusted Computing. In: Cryptographic Hardware for Embedded Systems (CHES 2005). (2005)
15. Khan, M., Seifert, J., Wheeler, D.M., Brizek, J.P.: A platform-level trust-architecture for hand-held devices. In: Cryptographic Advances in Secure Hardware (CRASH 2005). (2005)