Secure Update Mechanism for Remote Update of FPGA-Based System

Benoît Badrignans[1,2], Reouven Elbaz[3] and Lionel Torres[1]
[1]LIRMM UMR University of Montpellier 2- CNRS C5506, Montpellier, FRANCE, name@lirmm.fr
[2]SAS NETHEOS, Montpellier, FRANCE, b.badrignans@netheos.net
[3]Dept of Electrical Engineering, Princeton University, USA, relbaz@princeton.edu

Abstract— **Remote update of hardware systems is a convenient service enabled by Field Programmable Gate Arrays (FPGA) based systems. This service turns out to be essential in applications like Space-based FPGA systems or Set-top Boxes. However, the remote characteristic allows for a set of attacks that may challenge the confidentiality and the integrity of the FPGA configuration, the bitstream. Existing schemes propose to encrypt and to authenticate the bitstream to thwart those attacks. However we show in this paper that they do not prevent the replay of old bitstream versions, and thus give the opportunity to an adversary to downgrade the system. We propose a new technique that ensures bitstream confidentiality and integrity and detects replay of old bitstreams.**

Keywords- FPGA security; bistream; integrity; confidentiality; replay attack;

I. Introduction

FPGA-based systems are widely spread in embedded systems. The capability of remotely configuring their hardware configuration (i.e. the bitstream) is an attractive property for applications requiring regular update/upgrade of the hardware (e.g. FPGA-based cryptographic co-processors, Space-based FPGA [1]). For instance such property allows for repairing eventual security vulnerabilities while the device is already deployed on the field. However, while being extremely convenient, remote update generates security issues: an adversary spying on this communication channel can retrieve the hardware IP. In addition, an adversary can tamper with the bitstream on its way to the FPGA. Those attacks challenge respectively the confidentiality and the integrity of the bitstream.

Our FPGA usage model is inspired from [2]. FPGA vendor is the entity that designs and produces FPGA chips. IP designers provide hardware description of components that a System Designer (SD) assembles to produce an FPGA-based system. Finally the system owner is the end user that exploits the system, the SDs and IP designers do not necessarily trust him.

IP designers are mainly interested in the protection of the confidentiality of their know-how and IPs while a SD wants his design to behave as he specified. Therefore, the SD needs to ensure that the running system configuration is genuine. Research efforts [3,4] as well as FPGA vendors [5,6,7,8] have proposed solutions to provide the bitstream protection at power up or upon update of the FPGA configuration. However we show in this paper that they fail in detecting system downgrade in which an adversary prevents a system upgrade / update by replaying an old FPGA configuration. This attack highlights the need of a secure management of the FPGA configuration version.

In this paper, we introduce a new technique to provide bitstream encryption and authentication upon a system update performed remotely by the SD. We describe the corresponding protocol the SD has to follow to perform this task and the required hardware support, the Secure Update Mechanism (SUM) architecture, which FPGA vendors need to provide for SDs to implement our solution.

The outline of this paper is as follows. Section 2 presents our security model. Section 3 gives an overview of past work. Section 4 presents our solution to ensure bitstream confidentiality and integrity on a remote update of a FPGA-based system. Finally section 5 evaluates the cost of our solution and section 6 concludes the paper.

II. Security Model

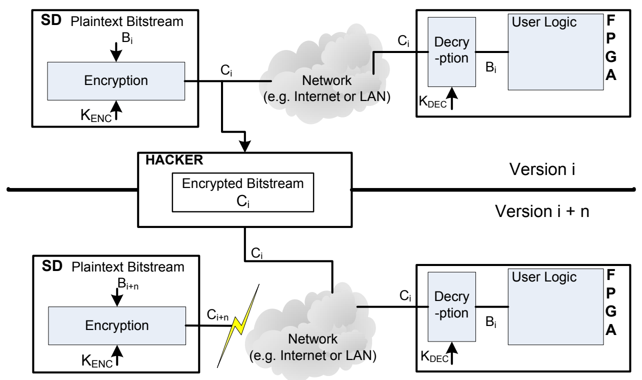


Figure.1. Replay Attack

The FPGA-based system is supposed resistant to physical attacks. We assume that the SD is trusted and that the FPGA platform is initialized in a trusted area.

In this work we focus on man-in-the-middle attacks performed over the network while the FPGA-based system is reconfigured. In such an attack, the adversary retrieves the bitstream by snooping on the communication channel (passive attack). Then, the second stage of this attack consists of an active attack where the adversary replaces a configuration update by one of his choice. The first stage of a man-in-the-middle attack challenges confidentiality of the bitstream while the second stage challenges its integrity. The objective of the adversary can be to make illegal copy of the underlying hardware IP or to downgrade the system. In case of active attacks, we call spoofing, the replacement of a genuine bitstream by a fake one chosen by the adversary and replay (Figure 1) its replacement by a FPGA configuration previously recorded over the network by the adversary. A replay is particularly dangerous for the system security because, even if bitstream encryption is enabled by the FPGA static logic, it allows for system downgrading.

III. Past Work

Nowadays high-performance [5][6] and some low-cost [7][8][11] FPGAs include hardwired mechanisms that ensure the *bitstream confidentiality* (called here the Secure Configuration Module : SCM). The configuration stream is encrypted with a symmetric key shared between the FPGA circuit and the system designer. Key set up is performed in a secure area by the system designer before the system is shipped.

Most of FPGA vendors propose solutions based on Cyclic Redundancy Checks (CRC) [5] to ensure *the integrity of the bitstream*. However the purpose of CRC is to detect transmission errors and not to check the integrity of data in the cryptographic sense. CRC codes can be easily broken with a brute force attack since they are only 16 or 32-bit long depending on the FPGA model. That is why [3] and [4] suggest using respectively an authenticated encryption mode or a Message Authentication Code (MAC) function to ensure the integrity of the bitstream. Some Actel FPGAs [8] include an AES-based Message Authentication Code engine that allows the SD to append a keyed-hash to the configuration stream. Integrity checking is done by the FPGA configuration logic that verifies this keyed-hash before embedded configuration flash loading.

Existing bitstream integrity solutions prevent spoofing of the bitstream but are powerless to prevent a *replay attack* and thus *system downgrade* threats. In [3,4,8] the keyed-hash is computed only over the received bitstream. Therefore the FPGA configuration logic is not able to make the difference between different keyed-hash/configuration pairs legitimately generated by the SD in the past. An adversary that replays a bitstream with its keyed-hash will succeed in his attacks.

Recent works on reconfigurable trusted computing propose FPGA-based implementation of the Trust Plateform Module (TPM) [9]. They address the issue of the secure update of FPGA bitstream through the TPM functionality. Moreover [10] proposes an implementation of TPM on current FPGA that does not provide bitstream encryption. [10] assumes that bitstream reverse engineering is too difficult to achieve.

Saar Drimer brings on the issue of replay attacks in [2]. He suggests two different avenues of research to solve the problem. The first suggestion requires the SD to implement additional security features in the user logic to send authenticated message to a trusted authority who then attests to the running FPGA configuration version. The second one is the use of nonces in the authentication process to ensure the freshness of the bitstream.

In this paper we propose a solution that leverages existing hardware (the SCM) to ensure the integrity and the confidentiality of the bitstream while communicated over a network with a particular focus on the secure management of FPGA configuration versions. Our goal is to provide a solution that can be implemented by FPGA vendors in static logic and that can be used for all applications. As touched on in [2], this solution enrolls a nonce in the authentication computations

IV. Secure Mechanism for Remote Update

A. Overview

Our objective is to provide bitstream confidentiality and integrity upon system updates performed remotely by the SD. To reach this objective, we propose that the SD encrypts his bitstreams and computes a MAC over them; however to ensure the freshness of those bitstreams the underlying MAC function must take as an extra input a tag called the Bitstream Version Tag (BVT). As we show in the following subsection, for the scheme to be secure this tag must be a NONCE (Number used ONCE). Our mechanism – called Secure Update Mechanism – located in the FPGA chip (Figure 2), securely stores a copy of this tag (BVT_{SUM}) – trusted since stored on the tamper-evident area, the FPGA chip – and handles the integrity verification and the decryption of the bitstream and implicitly authenticates of the SD upon system update. To do so, it computes a MAC over the received encrypted bitstream and over the trusted BVT_{SUM} and compares it with the MAC received from the SD.

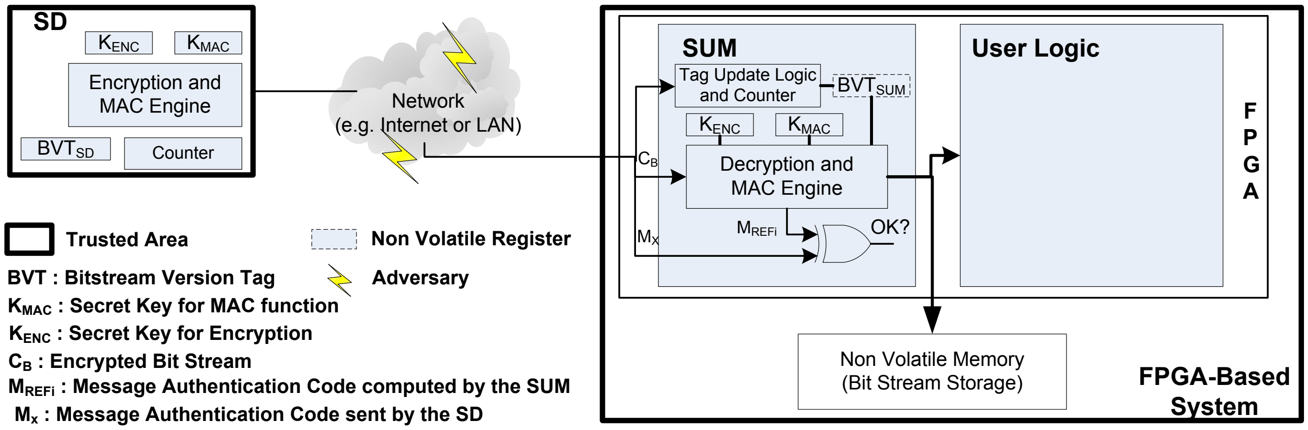

SD
K_{ENC}
K_{MAC}
Encryption and MAC Engine
BVT_{SD}
Counter
Network (e.g. Internet or LAN)
SUM
Tag Update Logic and Counter
BVT_{SUM}
K_{ENC}
K_{MAC}
C_B
Decryption and MAC Engine
M_X
M_{REFi}
OK?
User Logic
FPGA
Non Volatile Memory (Bit Stream Storage)
FPGA-Based System
Trusted Area
Non Volatile Register
Adversary
BVT : Bitstream Version Tag
K_{MAC} : Secret Key for MAC function
K_{ENC} : Secret Key for Encryption
C_B : Encrypted Bit Stream
M_{REFi} : Message Authentication Code computed by the SUM
M_x : Message Authentication Code sent by the SD

Figure 2. System overview – Secure Update Mechanism (SUM) architecture

B. *Secure Update Mechanism of FPGA Configuration*

Figure 2 depicts the architectural support, the Secure Update Mechanism (SUM) unit, required in an FPGA to implement our scheme. In the following E_{KENC}, D_{KENC}, and MAC_{KMAC} denote respectively the encryption and decryption functions enrolling the secret key K_{ENC} and the MAC function enrolling the secret key K_{MAC}, implemented in the SUM and by the SD. Moreover, in this paper we use || to refer to the concatenation operator.

Platform Initialisation. We assume that, before selling or deploying the FPGA platform on the field, the SD initializes it by storing both the secret key K_{ENC} used for bitstream encryption/decryption and the secret key K_{MAC} used for MAC computations in a Non-Volatile Memory (NVM) inside the FPGA device. The SD also initializes the non volatile register used to stored the Bitstream Version Tag in the SUM (BVT_{SUM}) at a given value that does not need to be secret (e.g. zero). The SD keeps a copy of K_{ENC}, K_{MAC}, and his copy (BVT_{SD}) of the initial value of the BVT in a trusted database.

As mentioned above, the BVT value must be a nonce shared by the two parties, the SD and the FPGA-SUM, in order to ensure the freshness of the bitstream. In the following we consider that it will be generated using a monotonic counter and we will show how the SD securely maintains the copy of BVT of the two parties (the FPGA-based system and himself) during the secure update process.

Secure FPGA Configuration Update. The protocol that the SD and the FPGA-SUM must follow during the update of a FPGA configuration is depicted in Figure 3. The SD is the only entity that should be able to update the FPGA configuration and the BVT_{SUM}. Therefore the SD must be authenticated by the SUM upon an update process. To do so a dedicated command (ComUpdate) is used by the SD that grants him access to the BVT_{SUM} register. Thus, to update a bitstream, the SD (Figure 3):

i) encrypts the new bitstream B ($C_B = E_{KENC}$ (B))

ii) increments the BVT_{SD} value with a monotonic counter, the new BVT uniquely identify the new bitstream, hence it is used in the subsequent MAC computations

iii) generates an update command (ComUpdate) and computes a MAC, M_{UPD}, over the concatenation of this command with the 1-bit value "0" and with the new BVT value (M_{UPD}= MAC_{KMAC}(ComUpdate || 0 || BVT_{SD})

iv) computes a MAC, M_{EB}, over C_B concatenated with the 1-bit value "1" and to the new BVT value (M_{EB}= MAC_{KMAC}(CB || 1 || BVT_{SD})

v) sends ComUpdate, M_{UPD} C_B and M_{EB} to the FPGA system.

Upon reception of those messages, the FPGA-SUM performs the following operations:

i) detects the update request by decoding ComUpdate

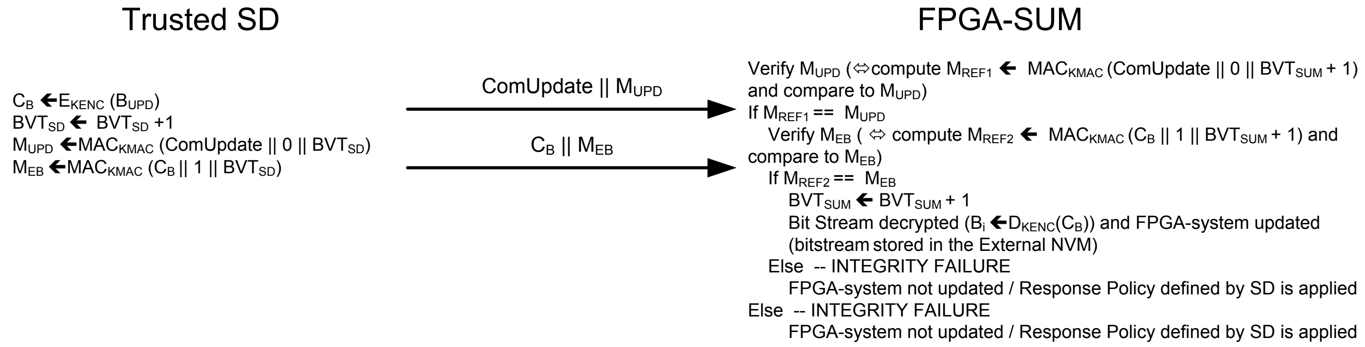
ii) checks that the request is valid i.e. it authenticated SD – by computing a reference MAC M_{REF1} over the concatenation of the ComUpdate, with the 1-bit value "0" and with BVT_{SUM} + 1 (M_{REF1}=MAC_{KMAC}(ComUpdate || 0 || BVT_{SUM} + 1)) and by comparing it to M_{UPD}.

iii) if the command is valid – M_{REF1} and M_{UPD} matched – it checks that the encrypted bitstream is genuine – by computing a reference MAC M_{REF2} over C_B || 1 || BVT_{SUM} + 1 (M_{REF2}=MAC_{KMAC}(C_B || 1 || BVT_{SUM} + 1)) and by comparing it to M_{EB}

iv) if the bitstream is genuine – M_{REF2} and M_{EB} matched – it updates BVT_{SUM} ($BVT_{SUM} \leftarrow$ BVTSUM + 1) to lock the platform with the new FPGA configuration bitstream and to synchronize the BVT_{SUM} value with the one kept by SD (BVT_{SD}).

The 1-bit value used as input in the MAC computations allows for the use of a single BVT value for a given bitstream update (i.e. 2 nonce values are generated from the same BVT). Note that if one of the MAC comparison processes fails, an integrity checking flag is raised and a response policy designed by the SD is applied.

BVT size. The tag size is a crucial parameter because it determines the number of FPGA configuration update the SD can perform using a given K_{MAC}. Indeed, once the counter reaches its limit a new K_{MAC} must be generated for a BVT value not to be used twice in a MAC computation enrolling this same secret key. This situation is not convenient because it might require the physical presence of the SD to change K_{MAC}. An easy way to avoid this situation is to choose a large tag, for instance 64-bit. With a 64-bit tag, the counter that generates it will never have the time to roll over during the FPGA lifetime.



```
$C_B \leftarrow E_{KENC}(B_{UPD})$
$BVT_{SD} \leftarrow BVT_{SD} + 1$
$M_{UPD} \leftarrow MAC_{KMAC}(ComUpdate || 0 || BVT_{SD})$
$M_{EB} \leftarrow MAC_{KMAC}(C_B || 1 || BVT_{SD})$
```

```
Verify $M_{UPD}$ ($\Leftrightarrow$ compute $M_{REF1} \leftarrow MAC_{KMAC}(ComUpdate || 0 || BVT_{SUM} + 1)$ and compare to $M_{UPD}$)
If $M_{REF1} == M_{UPD}$
   Verify $M_{EB}$ ($\Leftrightarrow$ compute $M_{REF2} \leftarrow MAC_{KMAC}(C_B || 1 || BVT_{SUM} + 1)$ and compare to $M_{EB}$)
   If $M_{REF2} == M_{EB}$
      $BVT_{SUM} \leftarrow BVT_{SUM} + 1$
      Bit Stream decrypted ($B_i \leftarrow D_{KENC}(C_B)$) and FPGA-system updated
      (bitstream stored in the External NVM)
   Else -- INTEGRITY FAILURE
      FPGA-system not updated / Response Policy defined by SD is applied
Else -- INTEGRITY FAILURE
      FPGA-system not updated / Response Policy defined by SD is applied
```

Figure 3. Secure Communication Protocol between the SD and the FPGA-SUM upon the Secure Update of the FPGA Configuration, and Pseudo Code of the cryptographic operations to be performed by the two parties.

C. Security Analysis

Our scheme assumes that the underlying block cipher encryption mode and MAC function are secure. Our analysis focuses on the security of the proposed communication protocol between the two parties, the SD and the FPGA-SUM, and on the prevention of the replay of bitstream configurations.

The integrity and the freshness of each transaction between the SD and the FPGA-SUM are ensured by the computation of MACs taking a nonce as input. The uniqueness property of nonces used in MAC computations for different bitstreams is ensured by BVT that is produced by a monotonic counter. For a given bitstream, the uniqueness of nonce is ensured for each MAC computation by a dedicated 1-bit value concatenated to the BVT. Two MAC computations are required for the management of a given bitstream: M_{UPD} and M_{EB}. Therefore, as mentioned above an extra 1-bit input (respectively "0" and "1") to each of these MAC computations allows for the use of nonces generated from the same BVT value. The BVT as well as this 1-bit value does not need to be secret but tamper-evident. This is enforced by design since the BVT is stored in trusted areas (the SD database and the FPGA-SUM) and therefore is tamper-proof. One may argue that the incrementation of the BVT in the FPGA-SUM may be triggered from outside. However, the tag update logic in the SUM is controlled by the ComUpdate which is authenticated by MAC enrolling a nonce and a secret key. Thus this command can neither be replayed nor spoofed.

V. Implementation Aspects

The objective of this paper is to propose a new security scheme to ensure the confidentiality and the integrity of the bitstream with a mechanism for secure management of FPGA configuration versions. Hence, ideally the hardware support required by our solution would be implemented in the static logic of the FPGA by FPGA vendors to allow SDs to implement it at lower cost. In this section we evaluate the cost of the proposed solution in this latter case.

FPGA vendors tools. FPGA vendors must slightly modify their bitstream generation tools to add MAC computation as well as the necessary support to generate the update command.

FPGA device impact. Current FPGAs already include an AES engine for decryption of the bitstream. This engine can be reused to implement the MAC function as suggested in [3,4]. Moreover, existing FPGA device already contains a non volatile register to store the encryption key (K_{ENC}), thus our solution would only require two extra non-volatile registers for K_{MAC} and BVT_{SUM}, typically of 128-bit and 64-bit. The logic that controls the configuration of the FPGA must also be modified to manage the tag update command. Therefore the hardware overhead implied by our solution is quite negligible since the main component (e.g. AES) are already implemented.

VI. Conclusion

Remote update of FPGA-based systems is a challenging issue from a security point of view. We showed that existing mechanisms to ensure bitstream confidentiality and integrity via encryption and authentication fail to prevent bitstream replay and thus system downgrade.

In this paper we proposed a new communication protocol between the System Designer (SD) and an FPGA platform to update the FPGA configuration while preserving its confidentiality and integrity. Moreover, we described the hardware support the FPGA vendors need to provide for the SD to implement the solution and we showed that the corresponding overhead is negligible when considering current FPGA technology.

On-going work considers the extension of the threat model to attack on buses and memory with as objective to only trust the FPGA device – indeed replay attacks may be conducted on the bus or in memory at power up of the system when the FPGA is configured from flash.. In addition, a convenient feature to add to the system is to provide the SD with an alert system to inform him if the device has been correctly updated or if it is under attack.

References

[1] Surratt, M.; Loomis, H.H.; Ross, A.A.; Duren, R. "Challenges of Remote FPGA Configuration for Space Applications" Aerospace Conference, 2005 IEEE

[2] Saar Drimer, 'Volatile FPGA design security – a survey, Computer Laboratory, University of Cambridge, available at: www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf

[3] M.Parelkar, K.Gaj (2005), Implementation of EAX mode of operation for FPGA bitstream encryption and authentication, Field-Programmable Technology, 2005. Proceedings. 2005 IEEE Intl Conference 11-14 Dec. 2005 Page(s): 335 – 336

[4] Saar Drimer Computer Laboratory, University of Cambridge, Cambridge, "Authentication of FPGA Bitstreams: Why and How", available at www.cl.cam.ac.uk/~sd410/papers/bsauth.pdf

[5] Xilinx datasheet, Virtex-4 configuration guide available at: , www.xilinx.com/support/documentation/user_guides/ug071.pdf

[6] Altera whitepaper, Design Security in Stratix III Devices, available at: www.altera.com/literature/wp/wp-01010.pdf

[7] LatticeXP2 Family Handbook available at: www.latticesemi.com/dynamic/view_document.cfm?document_id=24315

[8] Actel handbook, Actel ProASIC®3 Handbook, available at: http://www.actel.com/documents/PA3_HB.pdf

[9] Eisenbarth T., G neysu, T. Paar, C. Sadeghi, A.R. Schellekens, D. Wolf, M.: Reconfigurable Trusted Computing in Hardware. In: STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing, New York, NY, USA, ACM (2007) 15–20

[10] D. Schellekens, P. Tuyls, and B. Preneel, "Embedded Trusted Computing with Authenticated Non-Volatile Memory," In TRUST 2008, Lecture Notes in Computer Science, Springer-Verlag, 12 pages, 2008.

[11] L. Bossuet, G. Gogniat, W. Burleson, Dynamically configurable security for SRAM FPGA bitstreams, in: Proceedings of 11th IEEE Reconfigurable Architectures Workshop, RAW, Santa Fé, USA, 2004