SECURE FPGA CONFIGURATION ARCHITECTURE PREVENTING SYSTEM DOWNGRADE

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

In the context of FPGAs, system downgrade consists in preventing the update of the hardware configuration or in replaying an old bitstream. The objective can be to preclude a system designer from fixing security vulnerabilities in a design. Such an attack can be performed over a network when the FPGA-based system is remotely updated or on the bus between the configuration memory and the FPGA chip at power-up. Several security schemes providing encryption and integrity checking of the bitstream have been proposed in the literature. However, as we show in this paper, they do not detect the replay of old FPGA configurations; hence they provide adversaries with the opportunity to downgrade the system. We thus propose a new architecture that, in addition to ensuring bitstream confidentiality and integrity, precludes replay of old bitstreams. We show that the hardware cost of this architecture is negligible.

1. INTRODUCTION

FPGA is nowadays commonly used in applications requiring regular upgrade of the hardware (e.g. Set-top Boxes, FPGA-based cryptographic co-processors, Space-based applications [1]). Remote update/upgrade is attractive in such systems to offer new multimedia features or to repair eventual security vulnerabilities. However, remote update requires transmitting the hardware Intellectual Property (IP) over insecure communication channels and thus introduces new security issues. First, an adversary spying on this communication channel can retrieve the hardware IP to sell illegal copies or leak it to the public domain. Then, an adversary can tamper with the bitstream on its way to the FPGA with a man-in-the-middle attack. Those attacks challenge respectively the confidentiality and, the integrity of the bitstream.

Our FPGA usage model is inspired from [2]. The **FPGA vendor** is the entity that designs and produces FPGA chips. **IP designers** provide hardware descriptions of components that a **System Designer** (SD) assembles to produce an FPGA-based system. Finally, the **system owner** is the end user who 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. FPGA vendors provide bitstream encryption for this need [3,6,7.8]. For the SD, the main security objective is to have his design behave as he specified it. Therefore, the SD wants to ensure that the running system configuration is genuine. Research efforts [4,5] have proposed solutions to provide bitstream authentication at power-up or upon update of the FPGA configuration. However, these schemes do not detect nor prevent the replay of an old version of the FPGA configuration. The opportunity an adversary has to replay a bitstream is a considerable security issue since this way, he can easily preclude a system update intended, for instance, to fix security vulnerabilities. When such a replay attack is performed, we say that the adversary has *downgraded* the system. This attack highlights the need for a secure management of the FPGA configuration version.

In this paper, we propose a new architecture to encrypt and authenticate a bitstream at power-up, when the configuration is loaded from an external memory, and upon a remote system update performed by the SD, in order to prevent any tampering with the FPGA configuration (including a replay). In the latter case, we provide a communication protocol that allows the SD to remotely manage and check the bitstream configuration running in the FPGA system. We also describe the required hardware support, the Secure Update Mechanism (SUM), which FPGA vendors need to provide for SDs to implement our solution.

The outline of this paper is as follows. Section 2 presents our trust and threat model. Section 3 presents existing schemes to encrypt and authenticate a bitstream and highlights their security flaws. Section 4 presents our solution to ensure bitstream confidentiality and integrity and describes the communication protocol the SD must follow to remotely update the FPGA bitstream. This section also provides a security analysis of our scheme. Finally section 5 evaluates the cost of our solution and section 6 concludes the paper.

2. SECURITY MODEL

We assume that the FPGA system is exposed to hostile environment where physical but non-invasive attacks are feasible. The FPGA chip is supposed resistant to physical attacks and is trusted. All other components like memories are untrusted. Side channel attacks on the FPGA device are not considered in this paper.

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

In this work, we focus on man-in-the-middle attacks where the adversary retrieves the bitstream before the FPGA is configured and possibly modifies it. This attack can be performed at the board level, by tampering with the memory or buses if an external memory stores the bitstream, or remotely over the network when the SD sends the bitstream to the FPGA system. This 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 prevent hardware configuration update.

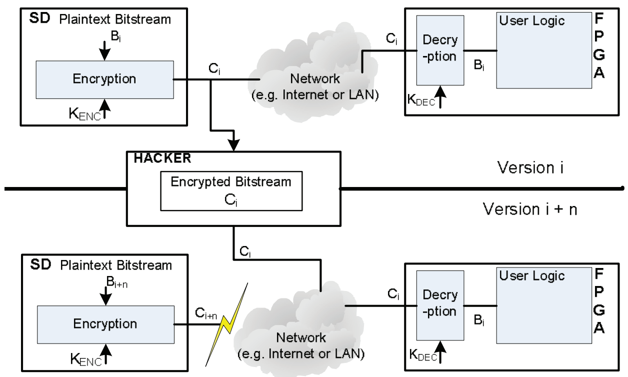


Fig.1. Replay Attack

In case of active attacks, we distinguish two kinds of attacks: *spoofing*, the adversary replaces the genuine bitstream by one of his choice, and *replay* (Figure 1) where the adversary replaces the genuine bitstream by a FPGA configuration previously recorded on a bus or over the network. A replay is particularly dangerous for system security because, even if bitstream encryption is enabled by the FPGA's static logic, it allows for system downgrading. Indeed, the objective of an update triggered by the SD can be to remove system vulnerabilities and with a replay, an attacker can prevent such a repair by configuring the FPGA with an old bitstream.

In this work, we assume that the SD is trusted and that the FPGA platform is initialized in a trusted area.

3. EXISTING MECHANISMS FOR BITSTREAM CONFIDENTIALITY AND INTEGRITY

Bitstream Confidentiality. Nowadays, high-performance [3,6,13,14] and some low-cost [7,8] FPGAs include hardwired mechanisms that ensure bitstream confidentiality (called here the Secure Configuration Module : SCM). The configuration stream is encrypted with a symmetric key (K_{ENC}) shared between the FPGA circuit and the system designer. Key setup is performed in a secure area by the system designer before the system is shipped. This mechanism allows for protection of the system designer's IP against cloning. It also precludes reverse engineering of the configuration stream that might lead to IP disclosure [9].

Bitstream Integrity. Some FPGA vendors implements Cyclic Redundancy Checks (CRC) [3,6,7]. However, the purpose of CRC is to detect transmission errors, not to check the integrity of data in the cryptographic sense. CRC codes are not collision-resistant; therefore, even with encryption, the probability of having a collision for two different bitstreams is non-negligible. This is why [4] and [5] 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 design activation.

Replay Attack and Bitstream Version Management. Existing bitstream integrity solutions prevent spoofing of the bitstream but are powerless against a replay attack and thus system downgrade threats. In [4,5,8], the keyed hash is computed only over the received bitstream. Therefore, the FPGA configuration logic is not able to distinguish 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 attack.

Recent work on reconfigurable trusted computing proposes FPGA-based implementations of the Trusted Platform Module (TPM) [10,11]. They address the issue of secure FPGA bitstream update through TPM functionalities. Moreover, [11] proposes an implementation of TPM on current FPGAs that do not provide bitstream encryption. [11] assumes that bitstream reverse engineering is too difficult to achieve and relies on a trusted external non-volatile memory.

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 to ensure the integrity and confidentiality of the bitstream with a particular focus on the secure management of FPGA configuration versions. To do so, we do not rely on an external trusted memory technology or on a TPM implemented in the user logic. 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 but also provides the SD with an alert system for detecting attacks and to check the running FPGA configuration version.

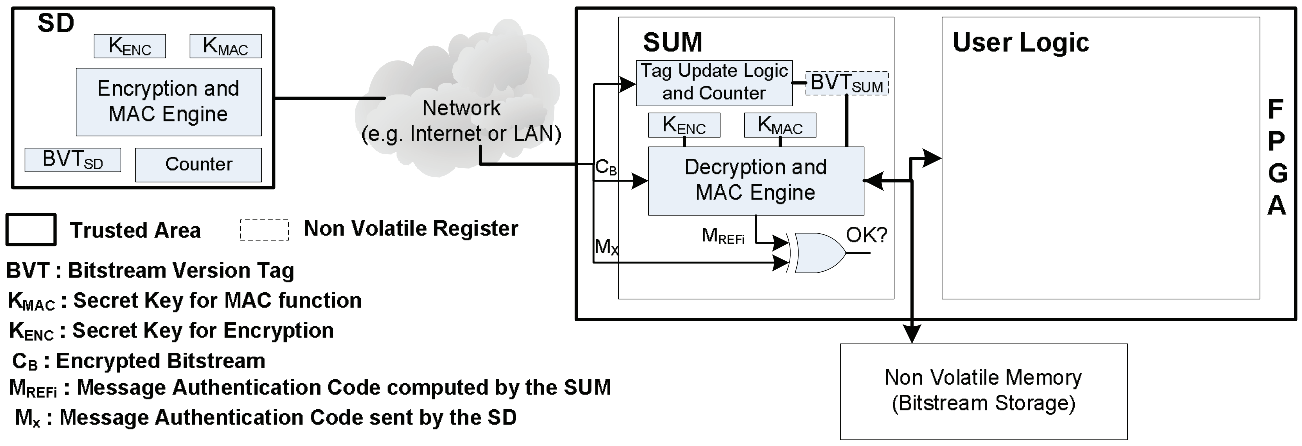

SD
K_{ENC}
K_{MAC}
Encryption and MAC Engine
BVT_{SD}
Counter
Network (e.g. Internet or LAN)
SUM
Tag Update Logic and Counter
BVT_{SUM}
K_{ENC}
K_{MAC}
C_B
Decryption and MAC Engine
M_X
M_{REFi}
OK?
User Logic
FPGA
Non Volatile Memory (Bitstream Storage)
Trusted Area
Non Volatile Register
BVT : Bitstream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

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

4. SECURE UPDATE MECHANISM

As described above, current secure FPGAs include a Secure Configuration Module (SCM) that allows encrypting the bitstream and in some cases (Actel [8]), computing MACs. However, we showed that existing mechanisms fail to ensure the freshness of those bitstreams. Therefore, we propose a new solution that leverages existing hardware to prevent replay attacks in addition to ensuring bitstream confidentiality and integrity.

4.1. Overview

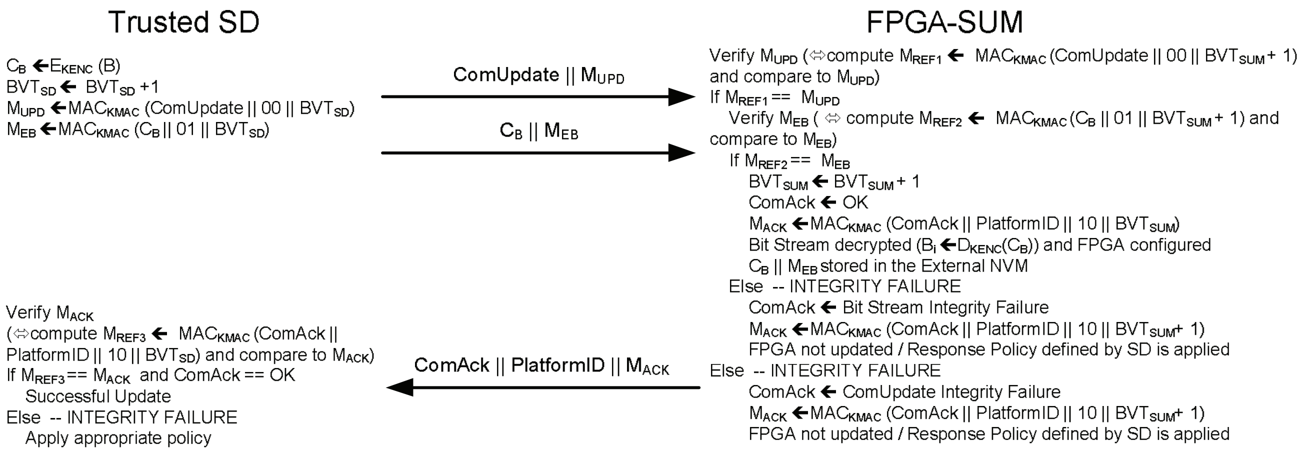
Our objective is to provide bitstream confidentiality and integrity in the two following situations: i) at system power-up when the FPGA configuration is transferred from external memory to the reconfigurable logic, ii) upon system updates performed remotely by the SD.
To reach this objective, we propose that the SD encrypt 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 SUM, located in the FPGA chip (Figure 2), securely stores a copy of this tag (BVT_{SUM}) – trusted since within the tamper-resistant area, the FPGA chip – and handles integrity verification of the bitstream and authentication of the SD at power-up and 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. Moreover, we provide the SD with an alert system to detect when the system is attacked and to check that the FPGA configuration is up-to-date; this feature is ensured by the SUM that sends to the SD a signature of its BVT copy (BVT_{SUM}) to attest of the bitstream version it has installed.

The rest of this section describes our proposition in details. We first detail the initialization process of the FPGA system. Then, we describe the communication protocol between the FPGA system and the SD that allows for the secure update of the FPGA configuration. We also explain how the SD securely updates the tag, called the Bitstream Version Tag (BVT), that locks a platform with a given version of the FPGA configuration. Finally, we provide a security analysis of our solution.

4.2. 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 || as the concatenation operator.
Platform Initialization. We assume that before selling or deploying the FPGA platform on the field, the SD initializes it by storing 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}) to a given value, BVT_0, that does not need to be secret (e.g. zero). The SD keeps a copy of $K_{ENC,}$ $K_{MAC,}$ the initial value of the BVT and the FPGA chip's unique identifier, the platformID in a trusted database. In addition, the SD encrypts the initial bitstream B_0 under K_{ENC} and stores the resulting ciphertext C_0 ($C_0=E_{KENC}(B_0)$) in the configuration memory of the FPGA system. He also has to compute a MAC M_0 using K_{MAC} over the concatenation of the ciphered[1] bitstream

1 Note that we are using the Encrypt-Then-MAC scheme since it has been proved by Bellare and al. [12] as the most secure way of combining encryption and authentication mechanisms.



```
$C_B \leftarrow E_{KENC}$ (B)
$BVT_{SD} \leftarrow BVT_{SD}$ +1
$M_{UPD} \leftarrow MAC_{KMAC}$ (ComUpdate || 00 || $BVT_{SD}$)
$M_{EB} \leftarrow MAC_{KMAC}$ ($C_B$ || 01 || $BVT_{SD}$)
```

```
Verify $M_{UPD}$ ($\Leftrightarrow$compute $M_{REF1} \leftarrow MAC_{KMAC}$ (ComUpdate || 00 || $BVT_{SUM}$ + 1)
and compare to $M_{UPD}$)
If $M_{REF1}$ == $M_{UPD}$
   Verify $M_{EB}$ ( $\Leftrightarrow$ compute $M_{REF2} \leftarrow MAC_{KMAC}$ ($C_B$ || 01 || $BVT_{SUM}$ + 1) and
compare to $M_{EB}$)
   If $M_{REF2}$ == $M_{EB}$
      $BVT_{SUM} \leftarrow BVT_{SUM}$ + 1
      ComAck $\leftarrow$ OK
      $M_{ACK} \leftarrow MAC_{KMAC}$ (ComAck || PlatformID || 10 || $BVT_{SUM}$)
      Bit Stream decrypted ($B_i \leftarrow D_{KENC}(C_B)$) and FPGA configured
      $C_B$ || $M_{EB}$ stored in the External NVM
   Else  -- INTEGRITY FAILURE
      ComAck $\leftarrow$ Bit Stream Integrity Failure
      $M_{ACK} \leftarrow MAC_{KMAC}$ (ComAck || PlatformID || 10 || $BVT_{SUM}$+ 1)
      FPGA not updated / Response Policy defined by SD is applied
Else  -- INTEGRITY FAILURE
      ComAck $\leftarrow$ ComUpdate Integrity Failure
      $M_{ACK} \leftarrow MAC_{KMAC}$ (ComAck || PlatformID || 10 || $BVT_{SUM}$+ 1)
      FPGA not updated / Response Policy defined by SD is applied
```

```
Verify $M_{ACK}$
($\Leftrightarrow$compute $M_{REF3} \leftarrow MAC_{KMAC}$ (ComAck ||
PlatformID || 10 || $BVT_{SD}$) and compare to $M_{ACK}$)
If $M_{REF3}$ == $M_{ACK}$ and ComAck == OK
   Successful Update
Else  -- INTEGRITY FAILURE
   Apply appropriate policy
```

Fig.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.

with a 2-bit value "01" and with the initial value of BVT (M_0=MAC_{KMAC}(C_0||01||BVT_0)). M_0 is appended to C_0 in the external non-volatile memory. The purpose of the 2-bit value is explained next. Note that the SD and the SUM both store the same value of the BVT in dedicated registers, respectively BVT_{SD} and BVT_{SUM}.

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. Therefore, it will be generated using a monotonic counter and we will show how the two parties securely maintain their copy of the BVT during the secure update process. However, as we will describe later, several MAC computations are performed during this process, each requiring a nonce; we thus use a 2-bit value in MAC computations related to a given bitstream update process to generate nonces from the same counter value, the BVT.

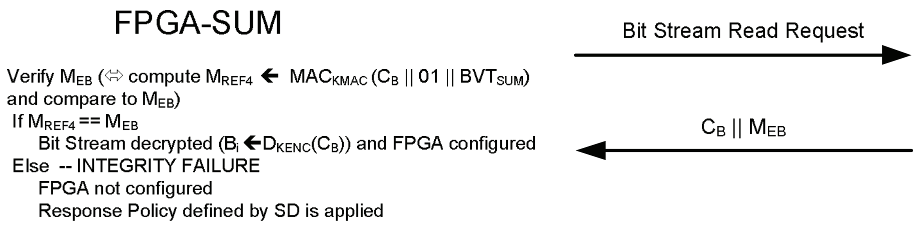
Secure FPGA Configuration Update. The protocol that the SD and the FPGA-SUM must follow during the update of an FPGA configuration is depicted in Figure 3. The SD is the only entity that should be able to update the FPGA configuration and the BVT_{SUM}. Therefore, the SD must be authenticated by the SUM upon an update process. This way the SUM is able to distinguish a bitstream update from a configuration from the external NVM and grant access to the BVT_{SUM} register to the SD. 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 identifies 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 2-bit value "00" and with the new BVT value (M_{UPD}= MAC_{KMAC}(ComUpdate || 00 || BVT_{SD})
iv) computes a MAC, M_{EB}, over C_B concatenated with the 2-bit value "01" and to the new BVT value (M_{EB}= MAC_{KMAC}(C_B || 01 || BVT_{SD}))
v) sends ComUpdate, M_{UPD} and C_B, 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 – by computing a reference MAC M_{REF1} over the concatenation of the ComUpdate, with the 2-bit value "00" and with BVT_{SUM} + 1 (M_{REF1}=MAC_{KMAC}(ComUpdate || 00 || 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 || 01 || BVT_{SUM} + 1 (M_{REF2}=MAC_{KMAC}(C_B || 01 || BVT_{SUM} + 1)) and by comparing it to MAC_{EB}
iv) if the bitstream is genuine – M_{REF2} and MAC_{EB} matched –, it updates BVT_{SUM} ($BVT_{SUM} \leftarrow BVT_{SUM}$ + 1) to lock the platform with the new FPGA configuration bitstream and to synchronize the BVT_{SUM} value with the one kept by SD. The external memory is updated with the encrypted bitstream C_B and its MAC, M_{EB}.
v) it prepares an acknowledgment message for the SD to attest to the completion of the platform update with a genuine bitstream (see Alert System)

Note that if one of the MAC comparisons fails, an integrity checking flag is raised and a response policy designed by the SD is applied.



```
Verify $M_{EB}$ ($\Leftrightarrow$ compute $M_{REF4} \leftarrow MAC_{KMAC}(C_B || 01 || BVT_{SUM})$
and compare to $M_{EB}$)
 If $M_{REF4} == M_{EB}$
    Bit Stream decrypted ($B_i \leftarrow D_{KENC}(C_B)$) and FPGA configured
 Else -- INTEGRITY FAILURE
    FPGA not configured
    Response Policy defined by SD is applied
```

Fig.4 Pseudo Code of the Decryption and Integrity checking of the Bitstream when loaded from the External NVM.

Alert System. After sending a new FPGA configuration, the SD wants to know if the update has been carried out successfully or if an adversary precluded the update – e.g. by tampering with the update command message or the new bitstream. Therefore the protocol for secure update of an FPGA system depicted in Figure 3 terminates by the alert system process. To perform such a process, the SUM must send an acknowledgment message to the SD including i) a header ComAck identifying the message as an update acknowledgment message and stating the result of the integrity checking processes performed by the SUM, ii) the platform ID and iii) a MAC M_{ACK} computed using K_{MAC} over this message concatenated with the 2-bit value "10" and BVT_{SUM} (M_{ACK}=MAC_{KMAC}(ComAck|| PlatformID||10||BVT_{SUM}). ComAck can take three different values: "ComUpdate Integrity Failure", "Bitstream Integrity Failure" or "OK". Those values respectively denote a corruption of the update command, a corruption of the bitstream or a successful termination of the update process. Upon reception of the acknowledgment message, the SD retrieves K_{MAC} and the current BVT value of the corresponding platform in his trusted database by using the platform ID. Then he computes a MAC MAC_{REF3} over the concatenation of ComAck with the platform ID, the 2-bit value "10" and his version of BVT (M_{REF3}=MAC_{KMAC}(ComAck||PlatformID||10||BVT_{SD})). If MAC_{REF3} matches M_{ACK}, the acknowledgment message is genuine; hence the SD deduces from ComAck whether his update has been correctly installed or not. If the SD does not receive any acknowledgement message, he can deduce that his update has been intercepted. Note that at the end of the bitstream update process, both the SD and the FPGA-SUM have the same value of BVT that locks the platform to the current FPGA configuration version.

4.3. Secure FPGA Configuration at power-up.

Once deployed, the FPGA device loads at power-up its configuration from the external NVM. At that time, the bitstream is decrypted and authenticated as described in Figure 4, the SUM:

i) reads the encrypted bitstream C_B and the corresponding MAC, M_{EB}, from the external NVM
ii) verifies M_{EB} by computing M_{REF4} over the concatenation of the loaded C_B, of the 2-bit value "01" and of the current value of BVT_{SUM} (M_{REF4}=MAC_{KMAC}(C_B||01||BVT_{SUM}) and by comparing it to M_{EB}
iii) if M_{REF4} matches M_{EB}, C_B is decrypted and the resulting bitstream B (B=D_{KENC}(C_B)) is used to configure the FPGA; if M_{REF4} does not match M_{EB}, it means that C_B is not genuine (i.e. it has been spoofed or replayed) an integrity checking flag is hence raised and a response policy designed by the SD is applied.

5. 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 the BVT, generated with a monotonic counter. For a given bitstream, the uniqueness of the nonce is ensured for each MAC computation by a dedicated 2-bit value concatenated to the BVT. Three MAC computations are required for the management of a given bitstream: M_{UPD}, M_{EB}, and M_{ACK}. Therefore, as mentioned above, we use three different 2-bit values (respectively "00", "01" and "10") to derive from the BVT a nonce for each of those computations. The BVT as well as those 2-bit values do not need to be secret, but only tamper-evident. This is enforced by design since the BVT is stored in trusted areas (the SD database and the FPGA-SUM). 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.

BVT size. The tag size is a crucial parameter because it determines the number of FPGA configuration updates the SD can perform using a given K_{MAC}. Indeed, once the counter reaches its limit, a new K_{MAC} must be generated to ensure a given BVT value is not used twice in two MAC computations enrolling the same secret key. An easy way to avoid this situation is to choose a large tag, for instance

64-bit. With such a tag, the counter that generates the BVT will never have the time to roll over during the FPGA lifetime.

6. IMPLEMENTATION ASPECTS

The hardware support required by our solution should be implemented in the static logic of the FPGA by FPGA vendors to allow SDs to implement it. In this section, we evaluate the cost of the proposed solution assuming a SUM in the static logic.

FPGA Vendor 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 [4,5]. Moreover, existing FPGA devices already contain a non-volatile register to store the encryption key (K_{ENC}), hence our solution would only require two extra non-volatile registers for K_{MAC} and BVT_{SUM}, typically of 128-bit each. Finally, a unique platformID must be set in the FPGA; this feature is for instance available on Spartan3-AN FPGA from Xilinx. The logic that controls the configuration of the FPGA must also be modified to manage the tag update command and the acknowledgement command generation. Therefore, the hardware overhead implied by our solution is quite negligible since the main component (e.g. AES) are already implemented.

As mentioned in [5], the configuration engine of an FPGA chip does not have enough temporary storage to hold the bitstream until the end of the authentication process. As a result, configuration bits must be loaded into the reconfigurable logic as they are decrypted and detection of corrupted bitstreams only occurs at the end of this process. Therefore a spoofed or replayed bitstream will overwrite, in the reconfigurable logic, the bits of the previous configuration. Even though the SUM immediately disables the FPGA when an attack is detected, it can only roll back to a known good state if a copy of the previous good configuration is stored in external memory.

7. 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. This protocol also provides the SD with an alert system that informs him when the FPGA system is under attack. Moreover, we described the hardware support the FPGA vendors need to provide for the SD to implement the solution and we show that the corresponding overhead is negligible when considering current FPGA technology.

ACKNOWLEDGMENTS. The authors wish to thank David Champagne for his valuable technical comments and his help in reviewing this paper.

REFERENCES

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

[2] S. 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] Xilinx commercial brochure, Lock Your Designs with the Virtex-4 Security Solution, available at:', www.xilinx.com/publications/xcellonline/xcell_52/xc_pdf/xc_v4security52.pdf.

[4] M.Parelkar, K.Gaj, “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.

[5] S. Drimer, “Authentication of FPGA bitstreams: Why and how”, In Proc. Of the International Workshop on Applied Reconfigurable Computing (ARC07), March 2007

[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: http://www.latticesemi.com/documents/HB1004.pdf.

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

[9] J. B. Note, E. Rannaud, ENS, “From the bitstream to the netlist”, International Symposium on Field Programmable Gate Arrays, FPGA’08, 2008.

[10] Eisenbarth T., Guneysu, 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.

[11] 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.

[12] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic construction paradigm. In T. Okamoto, editor, Asiacrypt 2000, volume 1976 of LNCS, p. 531-545. Springer-Verlag, Berlin Germany, Dec. 2000.

[13] 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.

[14] A. Lesea, IP security in FPGA, white paper Virtex-4 and Virtex-5 Devices, February 2007 available at: http://www.xilinx.com/support/documentation/white_papers/wp261.pdf.