International Journal of Computing and Digital Systems

Protecting FPGA-based Partially Reconfigurable Embedded Systems and IP Cores from Remote Update

[1]Tran Thanh, [2]Tran Hoang Vu, [3]Nguyen Van Cuong, [4]Pham Ngoc Nam

[1,2,4]School of Electronics and Telecommunications, Ha Noi University of Science and Technology, Ha Noi, Viet Nam.
[3]Faculty of Electronics and Telecommunications, Da Nang University of Technology, Da Nang, Viet Nam.

E-mail address: thanh.tran@hust.edu.vn, vu.tranhoang@ hust.edu.vn, nvcuong2000@gmail.com, nam.phamngoc@ hust.edu.vn.

Received 10 Sep 2013, Revised 4 Oct 2013, Accepted 13 Nov 2013, Published 1 Jan 2014

Abstract: Intellectual Property (IP) core remote update via the Internet is an advantage of FPGA-based devices. Whereby the system designer or user can change or renew a part of his equipment. However, the problem of communication in an untrusted environment can lead to significant damage: The risks for the IP cores of designers or damaging devices or losing the properties of the user. There have been already many proposals to overcome these problems, for example, using symmetric encryption techniques for bitstream, bitstream version to prevent replay attacks. This paper presents a framework, which include a protocol and authentication and encryption algorithms for protecting IP cores and partially reconfigurable embedded systems based on FPGA. Experimental results and analysis show that the proposed technique reduces resource overhead, increase the flexibility of the system, and it is robust against attacks.

Keywords: Partial reconfigurable, Bitstream security, Secure update.

I. Introduction

Remote update of the FPGA-based hardware system has changed the way to design, manufacture and distribute electrical products. Thanks to reconfiguration ability and reuseable IP cores, functions of the devices can be continually improved or completely changed in their life-cycle. However, most of the reconfigurations by remote update via the Internet, which has the most risk of attacks, for example, IP core cloning and reverse engineering or system destroying. The question of how to secure such updates against malicious interference may easily be answered: much existing cryptographic authentication protocol to protect the confidentiality and integrity of the transmission, such as transmission of IP cores from a server to a computer in the field. In fact, the secure remote update for the FPGA can be done relatively easily if the FPGA is a peripheral device of system, and the update of its bitstream can be handled entirely by software running on the system's main processor. In contrast, as not easily the FPGA with running applications on it updates itself without support of the processor from outside.

Consider, for example, an application scenario where a user has an IPTV (Internet Protocol Television) set-top-box (STB) connected to the service provider over the Internet. The central processing unit of the STB is a system on a FPGA chip which consists of an embedded microprocessor and a runtime reconfigurable hardware accelerator. The microprocessor is in charge of handling application software and user interface while the hardware accelerator is in charge of decoding video and audio streams. If the user wants to upgrade the video decoder to the latest version he/she can request the new bitstream for the decoder from the service provider. The new bitstream then will be downloaded to the STB and reconfigure the video hardware decoder without affecting the microprocessor and the audio hardware decoder. The downloading and reconfiguration of the new bitstream is taken care of by an application software running on the microprocessor. The bitstream should be compressed at the service provider side in order to save transmission bandwidth and storage space at both service provider and user sides. In addition, it should be encrypted to avoid intellectual property stealing and other attacks like spoofing and replay [1]. The management and encryption of the bitstream will incur an overhead at the service provider side while the decompression and decryption of the bitstream will incur an overhead at the user side.

To protect updating of these systems, F. Devic, L. Torres, and B. Badrignans [2] proposed a secure remote update with a number only ever use once ("nonce") to prevent man-in-the-middle attackers, which had an old bitstream version to downgrade the system impossibly. In [3], the authors presented a protocol that the replay attacks have been prevented. Partially reconfigurable FPGAs are an advantage of a system with many modules, where the modules can be designed for remote installing or upgrading independently without affecting the other ones, which are operating [4]. This flexibility allows the modules to renew its features or to fix security holes. Kean [5] and Bossuet [6] highlighted vulnerability of volatile FPGAs to IP piracy and reverse engineering, and proposed bitstream encryption as a countermeasure. Drimer [1] more recently examined a wide range of attack mechanisms and countermeasures. A more general review of security challenges facing embedded systems can be found in [7]. Adi [8] proposed a system based on the use of public and secret-key cryptography. In [9], Yuan summarized current IP protection goals and proposed various solutions. So far, it has few studied on the secure remote update of a partially reconfigurable system. Work in the next sections, we present a scheme for safely and flexibly remote updating of this system. This solution does not require the FPGA vendor to add any hard-wired circuits, and therefore, it can be done with existing volatile FPGAs.

The rest of the paper is organized as follows. Section 2 describes our framework. Section 3 describes an algorithm of the protocol in the FPGA side. The security analysis is presented in Section 4. Experimental results are shown in Secsion 5, and conclusions are drawn in Section 6.

II. The Framework

The structure of the framework is given in Fig. 1. Some components are part of the service provider side; others belong to the end-user side. These components are explained in more detail in the next subsections.

A. Components at the service provider side

The Bitstream/Software Repository consists of FPGA bistreams which are used to reconfigure the runtime reconfigurable hardware and softwares which can be installed and run on the embedded microprocessor at the user system. High level information (meta-data) is added for each bitstream. The meta-data consists of information about the version, the size of the bitstream, the target FPGA device and the resource requirements (e.g. flash memory, Block RAM…).

The User Profile Database stores information about all registered users including user system ID, version and ID of bitstream and software, etc. Based on the user profile, the service provider can send notification to the user whenever new updates are available.

The Server Update Manager (SuM) is in charge of bitstream and software updating management. Its functions include managing the bitstream/software repository, notifying new updates to the user, initiating communication session with the user side, guaranteeing a secure transmission of bitstreams to the user. Depending on the security protocol supported by the end user system, the SuM can select an appropriate protocol to communicate with the user system via the Client Update Manager (CuM). The flexibility in protocol selection allows the service provider to support a wide range of clients with different capability. Our framework therefore can use a standard cryptographic protocol as in [10] or a special protocol as in [2]. The SuM supports two modes of remote bitstream reconfiguration: on-line mode and off-line mode. In on-line mode, the reconfiguration is done during bitstream transmission and hence requires less temporary storage space at the end user system. In off-line mode, the reconfiguration only starts when the bitstream has been completely downloaded and stored in the memory of the end user system. The off-line mode therefore requires extra memory space at the end system.

The Encryption/Authentication Engine is in charge of encrypting and authenticating the bitstream. Similar to the protocol selection, depending on the algorithm supported by the end user system, the engine can select an appropriate encryption/authentication algorithm.

B. Components at the user side

At the user side, components drawn in dotted line are optional components. The system may have or may not have an operating system (OS). It may have a built-in hardware decryptor in which case a software decryptor may not be needed. For example, Xilinx Virtex devices have a built-in bitstream decryptor. Virtex-II and Virtex-II Pro support Triple-DES with a 56-bit key while Virtex-4 to Virtex-7 support AES with a 256-bit key. However, this feature is not available when using partial reconfiguration and therefore user logic decryptor or software decryptor is needed. In our framework, the decryptor, authenticator and decompressor can be built-in either software to save hardware resources or hardware to accelerate configuration.

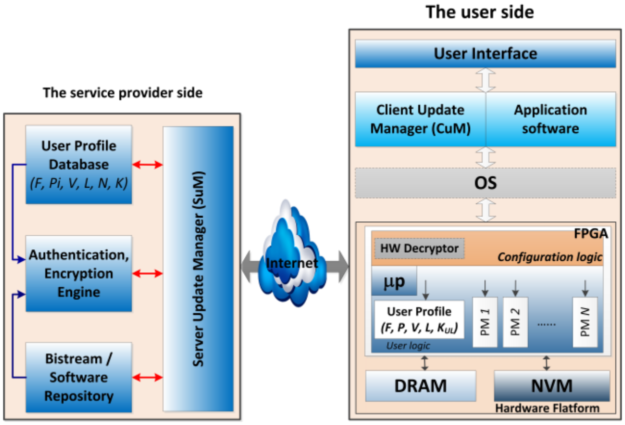
The service provider side
User Profile Database (F, Pi, V, L, N, K)
Authentication, Encryption Engine
Bistream / Software Repository
Server Update Manager (SuM)
Internet
The user side
User Interface
Client Update Manager (CuM)
Application software
OS
FPGA
HW Decryptor
Configuration logic
µp
User Profile (F, P, V, L, K_UL)
User logic
PM 1
PM 2
……
PM N
DRAM
NVM
Hardware Flatform

Figure 1. The framework structure

The CuM is responsible for managing the download and reconfiguration of the bitstream. It takes care of communicating with the service provider and maintains information about the profile of the end user system. The profile contains information about the system/FPGA ID, bitstream version, decryption algorithm (i.e. DES, AES…), authentication algorithm (i.e. SHA-1, SHA-2…), decompression algorithm and available resources. Table I shows an example of user system profile.

III. SECURITY REMOTE UPDATE PROTOCOL

A. Assumptions

Our protocol defines an interactive exchange between the SuM of service provider side, the entity in charge of distribu- ting new bitstreams to FPGA-based systems in the field, and the CuM of user side, the receiving end, implemented in theuser logic (UL) of each FPGA. This protocol is improved from Drimer's one [3] and applied within the framework had proposed in [11]. The framework for our proposed protocol is shown in the Fig. 1.

TABLE I. AN EXAMPLE OF USER SYSTEM PROFILE

System/FPGA ID	STB_123456A
Bitstream version	Ver 2.0
Decryption	AES-256
Authentication	SHA-512
Decompression	BitMask
H264 video bitstream size	0.6MB
Data flash	16 MB, used 10MB
DRAM	128 MB

We assume that the FPGA-based systems are exposed to malicious environments where physical but non-invasive attacks are feasible. FPGA is considered as a trusted zone. The side channel, invasive and power analysis attacks are not considered in this work.

We require a unique, non-secret, FPGA identifier F, which is used by the authentication process to ensure that messages can't be forwarded to other FPGAs. We also assume that, in the FPGA side, the message authentication function is implemented in the FPGA's user logic. The block decoding and decompression functions can be built in hardware or software.

B. The protocol

The secure remote update protocol consists of two algorithms for two sides of a system: Algorithm 1 for the FPGA side and Algorithm 2 for the service provider side. In this paper, we focus discussion on algorithm on the FPGA side, the Algorithm 1, that runs on the more constrained device.

Parameters used in the proposed protocol have been stored in the user profile database of Service provider and FPGA and are listed below:

Constants:

K_{PiUL}:	Keys shared with SuM
F:	FPGA (or System) ID

Variables:

P_i:	Reconfiguration Partitions ID
V_{Pi}:	Version ID of operating partial bitstreams
L_{max}:	Length of maximum partial bitstream of partition P_i
A_E:	Algorithmused for encryption
A_C:	Algorithm used for compression
V_{PiNVM}:	Version ID of NVM bitstream
N_{NVM}:	NVM counter value
L_{PiU}:	Length of uploaded partial bitstream
N_{PiLS}:	Partial bitstream license
V_{PiU}:	Version ID of uploaded bitstream
N_{US}:	Nonce generated by SuM
F_e, $P_{e,}$ V_e:	Expected value of F, P_i,V_{Pi}
N_{max}:	Upper bound for N_{NVM}
PB:	Partial bitstream file in compression
M_x:	MACs values

Algorithm 1: FPGA side

```
1:  $V_{PiNVM} := V_{Pi}$
2:  Receive($C$, $F_e$, $P_i$, $V_{Pie}$, $L_{PiU}$, $N_{max}$, $N_{US}$, $M_0$)
3:  **if** $L_{PiU} > L_{max}$ **then goto** 32
4:  **if** C ≠ "GetStatus" **thengoto** 2
5:  ReadNVM($N_{VNM}$)
6:  S := [$M_0$ = MAC($C$, $F_e$, $P_i$, $V_{Pie}$, $L_{PiU}$, $N_{max}$, $N_{US}$)] ∧
    ($V_{Pie} = V_{Pi}$) ∧ ($F_e = F$) ∧ ($N_{NVM} < N_{max}$)
7:  **if***S***then**
8:      $N_{NVM} := N_{NVM} + 1$
9:      WriteNVM($N_{VNM}$)
10: **end if**
11: $M_K$:= MAC($F$, $P_i$, $V_{Pi}$, $K_{PiUL}$)
12: $M_{LS}$:= MAC($F$, $P_i$, $V_{Pi}$,$N_{PiLS}$)
13: $M_1$ := MAC("RespondStatus", $F$, $P_i$, $V_{Pi}$, $L_{max}$, $N_{NVM}$,
    $A_E$, $A_C$, $M_K$, $M_{LS}$, $M_0$)
14: Send("RespondStatus", $F$, $P_i$, $V_{Pi}$, $L_{max}$, $N_{NVM}$, $A_E$,
    $A_C$, $M_K$, $M_{LS}$, $M_1$)
15: **If**¬*S***then goto** 2
16: Receive($C$, $M_2$)
17: **if**$M_2$ ≠ MAC($C$, $M_1$) **then goto** 2
18: **if** C = "Update" **then**
19:     Receive($PB$, $M_3$)
20: **end if**
21: **If** $M_3$ ≠ MAC($PB$, $M_2$) **then goto** 2
22: Receive("UpdateFinal", $V_{PiU}$, $M_4$)
23: **if** $M_4$ ≠ MAC("UpdateFinal", $V_{PiU}$, $M_3$) **then goto** 2
24: Decrypt()
25: Decompress()
26: ReconfCtr()
27: $V_{PiNVM} := V_{Pi}$
28: $M_5$ = MAC("UpdateConfirm", $V_{PiNVM}$, $M_4$)
29: Send("UpdateConfirm", $V_{PiNVM}$, $M_5$)
30: **if** C= "Abort" **then goto** 2
31: UpUserprofile($F$, $P_i$, $V_{Pi}$, $L_{max}$, $N_{NVM}$, $A_E$, $A_C$)
32: Warn("Resize your partial reconfigurable partition")
```

The Fig. 2 focuses on communications between the service provider side and the FPGA side. It explains how the process mechanism can verify the genuineness of pararameters to increment securely these non volatile values in view to a future update.

Each protocol session starts with an initial "GetStatus" message from the SuM and a "RespondStatus" response from the FPGA. In the "GetStatus" and "RespondStatus" messages, two sides exchanged necessary parameters for a secure update session,for example, numbers that are only ever use once (“nonces”), FPGA identifier *F*, bitstream version *V*, license number *N*, etc. After that, these parameters are indirectly transformed secretly into the message authentication code (MAC) during an update session. This ensured that each side verifies the freshness of any next data received in a simple way without having to repeat the old data of previous transaction.

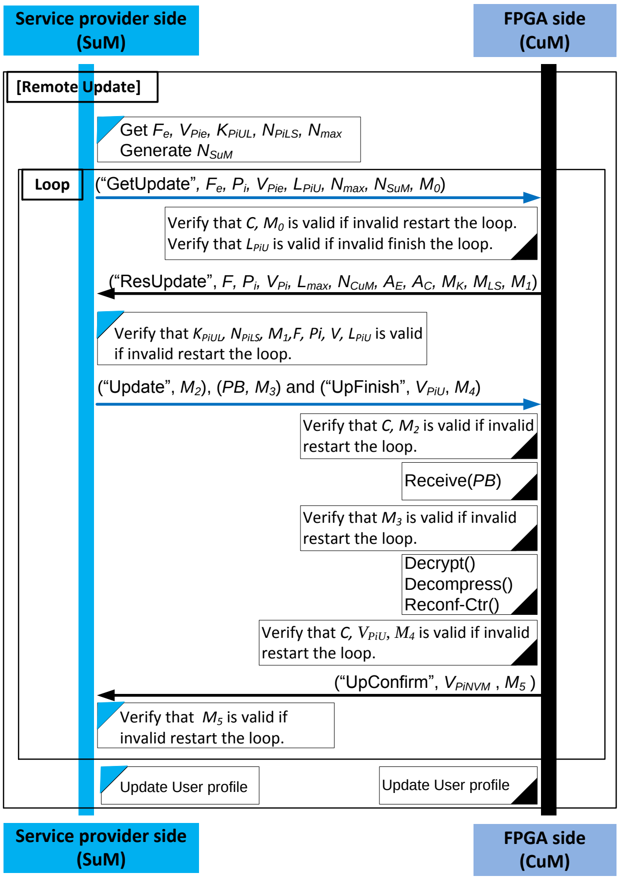
Service provider side (SuM)
FPGA side (CuM)
[Remote Update]
Get F_e, V_{Pie}, K_{PiUL}, N_{PiLS}, N_{max}
Generate N_{SuM}
Loop
(“GetUpdate”, F_e, P_i, V_{Pie}, L_{PiU}, N_{max}, N_{SuM}, M_0)
Verify that C, M_0 is valid if invalid restart the loop.
Verify that L_{PiU} is valid if invalid finish the loop.
(“ResUpdate”, F, P_i, V_{Pi}, L_{max}, N_{CuM}, A_E, A_C, M_K, M_{LS}, M_1)
Verify that K_{PiUL}, N_{PiLS}, M_1,F, Pi, V, L_{PiU} is valid if invalid restart the loop.
(“Update”, M_2), (PB, M_3) and (“UpFinish”, V_{PiU}, M_4)
Verify that C, M_2 is valid if invalid restart the loop.
Receive(PB)
Verify that M_3 is valid if invalid restart the loop.
Decrypt()
Decompress()
Reconf-Ctr()
Verify that C, V_{PiU}, M_4 is valid if invalid restart the loop.
(“UpConfirm”, V_{PiNVM} , M_5)
Verify that M_5 is valid if invalid restart the loop.
Update User profile
Update User profile
Service provider side (SuM)
FPGA side (CuM)

Figure 2. The secure protocol diagram for remote update

The nonce N_{US} generated by the SuM must be an unpredictable random number and also has not an opportunity to repeat. This prevents attackers from replaying the data of the previous session. The nonce N_{NVM} generated by the CuM is a monotonic counter maintained in the non-volatile memory (NVM). To protect this counter against attempts overflow itself, and beyond the capacity storage of the NVM, the CuM will only increase it when authorized to do so by the SuM. For this reason, the SuM includes "GetStatus" message an upper bound Nmax. In Algorithm 1, the protocol cannot proceed past the "RespondStatus" message unless the NVM counter is increased.

The initial exchange ensures that both sides agree the values of the parameters (*F, V, P, L, N*). CuM will not continue beyond the "RespondStatus" message unless these values are matched. This ensures that an attacker cannot reuse the "GetStatus" message for one particular FPGA (or a module) to update any other.

The parameters of the system, including the secret key are used as an input data to generate the message authentication code (MAC). The MACs values (M_0, M_2, M_3, M_4, M_5) ensure the freshness and continuity of the transaction between the two sides in one session. An attacker cannot generate any message that has not been issued by the SuM or the CuM because all transactions of the protocol are authenticated. After each successful session, the value of the parameters is updated and stored in the user profile database for new updates in future.

IV. Security Analysis

A. Confidentiality and authenticity

We assumed that at the FPGA side, a decryptor and an authenticator are implemented by hardware or embedded software. Thus, the partial bitstreams, which are transferred via the Internet, are always encrypted to resist attacks by using cloning and reverse engineering. The proposed protocol ensures the secure remote update for partially reconfigurable FPGA-based system. In the protocol, Algorithms 1 and 2 are used to protect the FPGA against replay attacks and maintain the freshness of the messages in a session. However, the protocol does not preventan attack of loading old bitstreams from NVM to FPGA. Research to protect NVM is beyond the goal of this article.

As analyzed above, the parameters of Algorithms 1 and 2 are used to calculate MAC known as M_0. Then M_0 is used again as a parameter to calculate MAC named as M_1. The process repeats several times until the end of the session. Thus, the parameters will influence values of calculating MACs during the session. The MACs are applied throughout the update process to prevent man-in-the-middle attackers to replay old bitstream or malicious code. We proposed a method in which uses a SHA hash function to calculate MAC with a secret key, K_{PiUL}, because of the above features. The key is included into SHA input data.

K_{PiUL}needs to be used in the proposed protocol to encrypt and decrypt the partial bitstreams, and also to authenticate the messages. Keys are the secret parameters, which are exchanged between the two sides only through a separate secure scheme such as a Trusted Authority. To verify the keys between the two sides, the hash function is used, which will convert the contents of the key and other parameters to MAC. The MAC, a confidential content of the key, is sent instead of transferring the key through the Internet. As shown in algorithm 2, when K_OK is true, the two sideshave the same keyK_{PiUL}.

In the protocol, the license of the partial bitstream, N_{PiLS}, is also considered as the second secret key and is included to calculate MAC, which makes the protocol more safety.

B. Parameter sizes

The nonce N_{US} should be large enough to make the creation of a dictionary of responses that can be replayed impractical. S. Drimer *et.al.*[1] referred that the use of an uniform distributed 64-bit word for N_{US} will ensure that an attacker who wants to perform 10^3 queries per second must spend a lot times upto many decades to findits matching value.

The proposed protocolusedK_{PiUL} of 256-bit length for AES encryption algorithm. The AES algorithm has been still considered safety, and in practice, it means that data encrypted with the AES algorithm has not been broken. The 256-bit key length of AES algorithm is enough to protect information classified as secret. Further details of AES can be found in FIPS-197 [2].

MAC values (M_0, M_1, M_2, M_3,M_4, M_5) are generated by a SHA-512 algorithm with 512-bit length. The MACs provide an equal generous safe margin to brute-force upload attempts. Further details of SHA can be found in FIPS-180 [3].

Other parameters' sizes (*F, P, V, etc.*) depend on the application.

V. Experimental Prototype System

A. System setup

To test the proposed framework, we have built a prototype system which consists of a reconfigurable embedded platform based on Xilinx Virtex-6 XC6VLX240T-1FFG1156 FPGA ML605 board and a laptop which plays the role of the service provider (Fig. 3). The ML605 board and the laptop are connected via a TCP/IP connection.

On the Virtex-6 FPGA chip, we embedded a MicroBlaze soft-core microprocessor using Xilinx Embedded Development Kit (EDK) ver. 14.1 software.EDK toolset allows designers to easily create platforms based on either MicroBlaze or PowerPC-405 processor. EDK offers a variety of peripherals (UARTs, counter, Ethernet, memory controller, general-purpose I/O and so on) and a one-connection solution based on the structure of the IBM CoreConnect bus [4]. The GNU compiler tool [5] of the MicroBlaze and PowerPC-405 has been used in the software flows. The source code for applications can be written in high level languages such as C and C + +.

For our current prototype system, we use a Xilkernel kernel. We implemented a TCP/IP stack on the MicroBlaze to provide an Ethernet connection between the FPGA platform and the laptop. For bitstream security, we implemented an AES-256 Decryptor and a SHA-512 authenticator in hardware. The detail implementation of the MicroBlaze system and applications will be explained in detail in the next subsections.

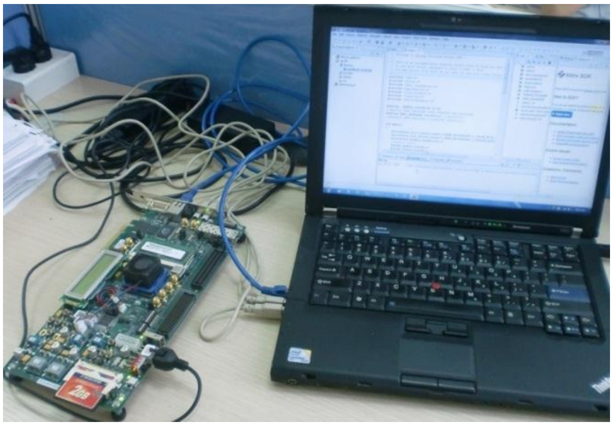

Figure 3. The prototype system

B. Porting MicroBlaze and TCP/IP stack

As mentioned above, we used EDK software to implement MicroBlaze processor. We selected different parameters and components for MicroBlaze with the Base System Builder tool [6]. For our prototype platform, the following parameters and components are shown in Table II. To implement the TCP/IP stack, we used Lightweight IP (lwIP), an open source TCP/IP networking stack for embedded systems. Xilinx EDK provides lwIP software customized to run on Xilinx embedded systems containing either a PowerPCor a MicroBlaze processor.

TABLE II. Parameters And Component For Microblaze

Parameter/Component	**Value**
Working frequency	100 MHz
I cache	32 KB
D cache	64 KB
Ethernet Controller	Tri-mode EMAC
Serial interface	RS232_UART lite
DRAM controller	MCB_DDR3

C. Implementation of AES-256 and SHA-512

An encryption system can prevent cloning and bitstream reverse engineering, but cannot prevent erroneous or malicious bitstream. In essence, encryption protects the contents of the bitstream independently with the device (from cloning, reverse engineering, etc.) while authentication ensures the confidentiality and integrity of the bitstream.

There are several authentication methods such as using bitstream encryption algorithms, message authentication code (MAC) or the hash functions. Among these, hash function SHA (Secure Hash Function) is the most widely used.

There have been a number of work that implement different bitstream encryption/authentication algorithms on embedded systems including MicroBlaze-based systems [7][8]. These algorithms include IDEA, DES, 3DES, Blowfish, AES-128, RC4, MD5, SHA-1 and SHA-256. To the best of our knowledge, no MicroBlaze implementation report has been found in literature for AES-256 and SHA-512, the most secure algorithms to date. Therefore, we decided to implement the two algorithms in both software running on MicroBlaze and hardware in order to compare and analyze the efficiency of the implementations.

The AES-256 algorithm consists of two parts: the AES controller and the AES decryptor. The SHA-512 algorithm consists of the SHA controller and the SHA-512 authenticator. Detail description of the algorithms can be found in [2] and in [3]. The AES controller is to receive the encrypted data from the Ethernet interface, group the data in 128 bit-packets and send the packets to the AES decryptor. The AES decryptor performs the decryption of the input data. Similarly, the SHA controller reads the data from the memory, groups data in 1024-bit packets and sends the packets to the SHA-512 authenticator. For software implementation, we used the open source code for AES-256 and SHA-512 from [19] and [20], respectively and ported them on MicroBlaze with some modification. However, we did not try to optimize the code, which will be the subject for future work. Table III summarizes the results of software implementation for AES-256 and SHA-512 algorithms in terms of size and speed.

For hardware implementation, we use AES-256 IP coresprovided by Xilinx and VHDL code for SHA-512 from [21]. The hardware implementation results in terms of hardware resource utilization and speed are shown in Table IV and Table V, respectively.

We can see that the speed of our software implementation is the same order of magnitude as that in [7] for AES-128 and SHA-256 running on MicroBlaze on Virtex-II Pro FPGA device. Although software implementation is two orders of magnitude slower than the hardware implementation, it does not consume any extra FPGA hardware resource while the hardware implementation takes up 23.36% of the Spartan-6 LX45 FPGA chip (Shown in Table VI). The overhead in terms of memory of software decryptor and authenticator is quite small compared to the size of flash memory and DRAM in most nowadays embedded systems. With the throughput as in Table III, it takes several minutes to update the bitstream for Xilinx Virtex-6 XC6VLX240T FPGA, which we believe is acceptable since an update only happens from time to time.

TABLE III. AES-256 AND SHA-512 SOFTWARE IMPLEMENTATION RESULTS ON VIRTEX-6 (XC6VLX240T)

Algorithm	LOC in C	Throughput (Kbps)	Size in memory (KB)
AES-256	488	105.5	128
SHA-512	500	185.4	138

TABLE IV. HARDWARE UTILIZATION OF AES-256 AND SHA-512 ON VIRTEX-6 (XC6VLX240T)

Module	Registers	(%)	LUTs	(%)	Slice	(%)
AES-256	3,096	1	3,751	2.48	1,293	3.43
SHA-512	2,246	0.75	2,299	1.52	843	2.24
Overall	5,342	1.75	6,040	4.00	2,136	5.67

TABLE V. PERFORMANCE OF HARDWARE AES-256 AND SHA-512 ON VIRTEX-6 (XC6VLX240T)

Algorithm	Throughput (Mbps)
AES-256	21.3
SHA-512	43.7

TABLE VI. HARDWARE UTILIZATION OF AES-256 AND SHA-512 ON SPARTAN-6 (XC6SLX45)

Module	Registers	(%)	LUTs	(%)	Slice	(%)
AES-256	757	1.37	1,516	5.52	724	10.61
SHA-512	1,177	2.13	2,052	7.47	870	12.75
Overall	1,934	3.50	3,568	13.19	1,594	23.36

VI. CONCLUSIONS

This paper presentsascheme for protecting the FPGA-based partially reconfigurable embedded systems of end users and IP cores of FPGA vendors. It maintains the confidentiality, integrity and freshness of the IP cores which are transferred over the Internet. TheAES-256 algorithmensures the prevention ofbitstream cloning and reverse engineering attacks, and the SHA-512 ensures the prevention replay attacks. In addition, the proposed protocol recommends using a compression algorithm to solve the problem of lack memory on FPGAs and bandwidth in the Internet. Our solution does not require additional hard-wire on the FPGA, so it can be applied to all current SRAM-based FPGA.

REFERENCES

[1] S. Drimer, “Volatile FPGA design security – a survey,” Journal of Engineering, Computer Laboratory, University of Cambridge, Version 0.96, pp. 1–51, 2008.

[2] F. Devic, L. Torres, and B. Badrignans, “Secure Protocol Implementation for Remote Bitstream Update Preventing Replay Attacks on FPGA,” 2010 International Conference on Field Programmable Logic and Applications, pp. 179–182, Aug. 2010.

[3] S. Drimer, “A protocol for secure remote updates of FPGA configurations,” Lecture Notes in Computer Science, vol. 5453, pp. 50–61, 2009.

[4] D. Koch and C. Beckhoff, “Advanced partial run-time reconfiguration on Spartan-6 FPGAs,” Field-Programmable Technology (FPT), 2010 International Conference on, pp. 361 – 364, 2010.

[5] T. Kean, “Secure Configuration of Field Programmable Gate Arrays,” Proceedings of the 11th International Conference on Field-Programmable Logic and Applications, pp. 142–151, 2001.

[6] L. Bossuet, G. Gogniat, and W. Burleson, “Dynamically configurable security for SRAM FPGA bitstreams,” Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004.

[7] S. Ravi, A. Raghunathan, and P. Kocher, “Security in Embedded Systems : Design Challenges,” vol. 3, no. 3, pp. 461–491, 2004.

[8] W. Adi and R. Ernst, “VLSI design exchange with intellectual property protection in FPGA environment using both secret and public-key cryptography,” Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on, vol. 00, pp. 24–32, 2006.

[9] L. Yuan, G. Qu, L. Ghout, and A. Bouridane, “VLSI Design IP Protection: Solutions, New Challenges, and Opportunities,” First NASA/ESA Conference on Adaptive Hardware and Systems (AHS’06), pp. 469–476, 2006.

[10] A. Braeken, J. Genoe, S. Kubera, N. Mentens, A. Touhafi, I. Verbauwhede, Y. Verbelen, J. Vliegen, and K. Wouters, “Secure remote reconfiguration of an FPGA-based embedded system,” 6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1–6, Jun. 2011.

[11] T. Thanh, P. N. Nam, T. H. Vu and N. V. Cuong, “A framework for secure remote updating of bitstream on runtime reconfigurable embedded platforms,” 2012 Fourth International Conference on Communications and Electronics (ICCE), pp. 471–476, Aug. 2012.

[12] NIST., “Announcing the Advanced Encryption Standard (AES),” National Institute of Standards and Technology (NIST). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.2001 ,

[13] NIST., “Secure Hash Standard (SHS),” National Institute of Standards and Technology (NIST). http:// csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, March, 2012.

[14] Xilinx_Inc., “Microblaze processor reference guide,” Reference manual, vol. 081, http://www.xilinx.com/support/documentation/sw_man, 2006.

[15] Xilinx_Inc., “Embedded System Tools Reference Manual,” EDK, vol. 111, http://www.xilinx.com/support/documentation/sw_man, 2012.

[16] Xilinx_Inc., “EDK Concepts, Tools and Techniques,” A Hands-On Guide to Effective Embedded System Design, vol. 683, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/edk_ctt.pdf, 2012.

[17] Y. Hori, A. Satoh, H. Sakane, and K. Toda, “Bitstream Encryption and Authentication with AES-GCM in Dynamically Reconfigurable Systems,” Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on, pp. 23–28, 2008.

[18] I. Gonzalez, F.J. Gomez-Arribas, “Ciphering algorithms in MicroBlaze-based embedded systems,” Computers and Digital Techniques, IEEE Proceedings -, vol. 153, no. 2, pp. 87–92, 2006.

Tran Thanh is a Ph.D. student in Electrical Engineering in ESRC laboratory of Hanoi University of Science and Technology (Vietnam), where he has been since 2010. He has a B. Eng. degree in Electronics and Telecommunications from Da Nang University of Technology and a M.Sc. degree from the University of Danang (Vietnam) in 2008, respectively. He works at The Vietnam Research Institute of Electronics, Informatics and Automation. His research is in reconfigurable computing, reconfigurable embedded systems and FPGA security.

Tran Hoang Vu is a Ph.D. student in Electrical Engineering of Hanoi University of Science and Technology (Vietnam), where he has been since 2010. He received B. Eng. degree in Electronics and Telecommunications from Da Nang University of Technology and M.Sc. degree from the University of Danang (Vietnam) in 2004 and 2008, respectively. From 2004 until now he has been working atDanang College of Technology-The University of Danang, Vietnam. His research interests include Reducing power consumption of Data Center Networks, reconfigurable embedded systems and low-power embedded system design.

Nguyen Van Cuong received B. Sc. degree In Solid State Electronics from Hue University of Science (Vietnam) in 1987. He was awarded a Ph.D. degree in Electrical Engineering from Uni. BW Munich, Germany in 2000. From 2000 until now he has been working at Danang University of Science and Technology, Vietnam. His main disciplinary focus is on embedded systems and energy–aware VLSI system design.

Pham Ngoc Nam received B. Eng. degree In electronics and Telecommunications from Hanoi University of Science and Technology (Vietnam) and M.Sc. degree in Artificial Intelligence from K.U. Leuven (Belgium) in 1997 and 1999, respectively. He was awarded a Ph.D. degree in Electrical Engineering from K.U.Leuven in 2004. From 2004 until now he has been working at Hanoi University of Science and Technology, Vietnam. His research interests include reconfigurable embedded systems and low-power embedded system design.