Secure extension of FPGA soft core processors for symmetric key cryptography

Lubos Gaspar, Viktor Fischer, Lilian Bossuet, Robert Fouquet

Secure extensions of FPGA soft core processors for symmetric key cryptography

Lubos GASPAR, Viktor FISCHER, Lilian BOSSUET, Robert FOUQUET
Université de Lyon
Laboratoire Hubert Curien, UMR 5516, CNRS
42000, Saint-Etienne, France
{*lubos.gaspar, fischer, lilian.bossuet, fouquet*}*@univ-st-etienne.fr*

***Abstract*—When used in cryptographic applications, general-purpose processors are often completed by a cryptographic accelerator – crypto-coprocessor. Secret keys are usually stored in the internal registers of the processor, and are vulnerable to attacks on protocols, software/firmware or cache memory. The paper presents three ways of extending soft general purpose processors for cryptographic applications. The proposed extension is aimed at symmetric key cryptography and it guarantees secure key management. Three security zones are created and physically separated in each of three configurations: processor, cipher and key storage zones. In the three zones, the secret keys are manipulated in a different manner – in clear or enciphered, as common data or keys. The security zones are separated on the protocol, system, architectural and physical levels. The proposed principle is validated on Altera NIOS II, Xilinx MicroBlaze and Actel Cortex M1 soft core processor extensions. The NIOS II processor needs fewer clock cycles per data block encryption, because the security module is included in the processor's data path. The data path of the MicroBlaze is unchanged and thus shorter, but additional clock cycles are necessary for data transfers between the processor and the security module. The Cortex M1 processor is connected via AHB bus and the cryptographic extension is accessed as an ordinary peripheral – a coprocessor. Although the interfacing is different, the three processors with their extensions attain the required high security level.**

***Keywords*-Hardware security; Crypto-processor; FPGA soft-core; NIOS II; MicroBlaze; Cortex M1**

I. INTRODUCTION

Data security systems implement very often both computationally extensive parallel cryptographic functions (such as multiplication in Galois fields, modular multiplication, matrix multiplication, etc.) and complex sequential algorithms (such as cipher modes, key management and cryptographic protocols). In order to fulfill contradictory speed/complexity requirements, sequential algorithms are mostly implemented by the use of general-purpose processors, while parallel functions are implemented in the coprocessor placed inside the same cryptographic module or logic device (when a monochip solution is required). This approach is very frequent in asymmetric key cryptography [1], [2], and also in block ciphers (such as AES) hardware implementation [3], [4]. Some embedded systems using processor/coprocessor approach implement both symmetric and asymmetric key cryptography algorithms in the same device [5].

The use of processors weakens security in all the above mentioned solutions. In order to face side-channel attacks [6], [7], the keys must be changed/manipulated regularly using a key management protocol. When a general-purpose processor manipulates confidential keys, the keys are saved in clear in processor registers or cache memory and are exposed to potential software attacks. One such attack has been recently demonstrated by Bangerter et al. in [8]. A small malicious software monitored the processor's cache memory during encryption and the confidential key was recovered remotely from the captured temporary data within a few minutes.

In order to counter software attacks, the authors in [9] used two processors executing different tasks on different security levels. They created two virtual zones inside the physical memory: the protected memory zone was dedicated to private key storage and the unprotected one to public data memorization. The security processor had access to both zones and the general-purpose processor was allowed to access only the public virtual zone. Since both zones were located in the same physical memory, certain types of attacks, such as protocol attacks [10] or timing violation attacks, were still possible.

It is clear that software attacks targeting confidential keys can be countered only if enciphering is realized independently from the general-purpose processor (GPP), e.g. in a hardware cipher and if the keys are stored in a dedicated memory. However, storing the keys in an external memory is not sufficient when facing software attacks. If confidential keys pass to the cipher via the processor in clear, they are vulnerable to attacks. In the proposed novel solution, the processor can manipulate the keys only indirectly: the keys are read/written from/to the key memory via a cipher and the processor can never read them in clear.

The paper is organized as follows: Section II proposes and discusses creation of hardware security zones that enable secure key management in conjunction with GPPs. Section III compares three basic types of interfacing GPP with

external modules. Section IV describes a novel principle of the security extension for soft GPPs. In Section V, the proposed security extension is evaluated and validated on three implementations based on NIOS II, MicroBlaze and Cortex M1 processors. Results are presented in Section VI and they are discussed in Section VII. Section VIII concludes the paper.

II. Principle of separation of protected and unprotected zones

We have explained in the previous section that for facing software attacks on embedded systems using GPPs, processor shouldn't have access to confidential keys in clear. It is thus necessary to isolate it from the key memory. In order to fulfill the highest security requirements, the separation should be realized on four levels:

- the protocol level,
- the system level,
- the architectural level,
- the physical level.

Next, we will describe these levels and their principles in more detail.

A. Separation on the protocol level

Integrity and confidentiality of exchanged keys is one of the most discussed security issues. In order to achieve a high security level, a robust communication protocol is needed. During key exchange, encrypted keys are usually transfered to the processor in packets. When a packet arrives to the processor, the keys have to be decrypted before being used for data encryption or decryption. However, if the decryption of the keys is carried out by the processor, these keys are exposed to software attacks. Therefore, keys have to be decrypted outside the processor in a dedicated unit so that unencrypted key or part of it can never leave the unit. Moreover, keys have to be authenticated before being used for data enciphering/deciphering. Once keys are decrypted and authenticated, they can be used for data encryption/decryption and authentication, but still outside the processor so that it will not have access to them. However, encrypted/decrypted data blocks can be processed directly by the processor e.g. when performing cipher mode operations.

It is the protocol that has to clearly separate key management and data processing tasks and define how and by which system units these tasks are performed. In addition to the separation of tasks, the protocol defines multiple-level key hierarchy. Higher-level keys (i. e. master keys) are used to encipher lower-level keys (i.e. session keys). The highest-level key has to be introduced to the system in unencrypted form through a separate entry by the trusted entity. All low-level keys have to be either generated inside the security module using a True Random Number Generator (TRNG) or received in a packet and decrypted. The low-level keys are used for data encryption, decryption and authentication.

B. Separation on the system level

The principle of the separation on the system level is illustrated in Fig. 1. This principle is based on the creation of three zones: a processor zone, a cipher zone and a key zone. The processor exchanges data with a cipher through the data bus (in black in Fig. 1). Encrypted session keys are also transported through this data bus when being exchanged with other communication counterparts. No other way for accessing the key memory from the GPP must exist.

Unencrypted keys are stored in a dedicated memory situated in a key storage zone. The key memory is separated from the GPP by the cipher. The keys are transfered between the cipher block and the key memory via the key bus (in grey in Fig. 1). This bus must be completely separated from the data bus interconnecting the GPP with the cipher. It is essential that paths allowing unencrypted secret keys to pass from the key bus to the data bus must not exist. This precondition is one of the most important from the security point of view because it enables separation of the key storage and the processor zones.

Before enciphering/deciphering data blocks, the cipher is initialized with the selected key via cipher key bus (in dashed grey in Fig. 1). Key selection is controlled by the processor through a control bus. The control path must be organized so that in case of an attack on the control bus, the selection of an incorrect key (e.g. by addressing an all-zeros key out of the key address range) must be prevented.

The principle of creation of security zones is independent from the type of encryption algorithm – any symmetric key block cipher with or without side channel attack countermeasures can be used. Furthermore, the two separation walls delimiting the cipher can be used during the dynamic cipher reconfiguration when changing cryptographic algorithm, while maintaining the key memory contents unchanged.

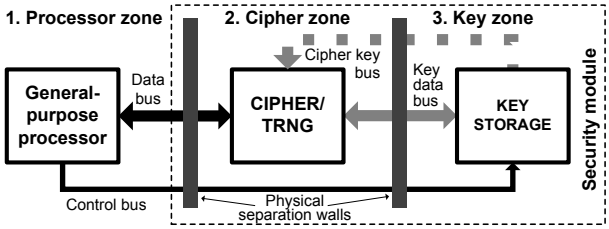

Figure 1: Separation on the architectural level

C. Separation on the architectural level

To achieve a high quality physical separation, the architecture of buses has to be organized in such a way that even after a physical attack, keys from the key zone cannot pass unprotected into the data zone. Bus multiplexers directing the flow of data to/from the cipher have to be placed so that

even if their control is violated, no physical path for keys to escape from the key zone can be created.

D. Separation on the physical level

In order to achieve a higher level of physical separation, an isolated area of the chip has to be dedicated to each zone. On the border of the two neighboring zones, an empty area has to be used. This area represents an insulation wall between two neighboring zones (see Fig. 1). Only selected signals are allowed to cross this wall. This countermeasure minimizes the possibility of the loss or corruption of secret keys by residual electromagnetic radiation from the protected zone (i.e. key zone) to the unprotected zone (i.e. processor zone). Using this approach, a physical attack on one zone has a minimal impact on the other zones. This physical insulation principle is recommended by NSTISS in its Red/Black installation guidance [11].

III. Communication interface

The data interface between the security module and the processor plays very important role in the system design. When considering the architecture, a trade-off has to be found between the performance, area and security criteria. Unfortunately, this can lead often to contradictory requirements.

A. Interface properties

Overall system performance depends on the interface type and parameters, such as bus width and latency. When managing keys, small data blocks are exchanged between the processor and the security module and pipelining is not efficient. In order to achieve higher performance, it is the best to suit the bus width to the cipher width. Unfortunately, this is not always possible.

From the security point of view, point-to-point communication is in general more secure than point-to-multipoint commnunication, because data are exchanged only between two units, and other peripherals are not physically connected. When using a point-to-multipoint interface, the bus is shared among all communication counterparts, and data exchanged between the security module and the processor can be potentially eavesdropped by other peripherals. In this case, some techniques, such as small firewalls protecting each peripheral on the bus [12], can be used.

B. Interface types

The security module can be interfaced with a GPP using the following types of connections:

1) The internal processor bus: The security module is included in the processor data path. Data pass to the module directly from registers as operands. Results are returned to the registers. The security module is controlled directly by the processor control unit through a dedicated control bus. The advantage of this solution is its high performance and minimal latency. Unfortunately, the module is a part of the processor's critical path and therefore it can slow the whole system down if not properly designed. A high security level is naturally achieved by the point-to-point connection.

2) The coprocessor-like bus: The security module is not included in the processor data path, but it is connected through a fast internal bus (often a coprocessor bus), running mostly at the same clock frequency as the processor core. This bus enables direct access to the processor registers thus minimizing communication latency, although the latency is higher when compared to the previous alternative. Considering the connection between a single security module and a processor, the connection has a point-to-point nature therefore high security level is maintained.

3) The peripheral bus: The security module is connected to the processor through a bus using a point-to-multipoint connection. In the best case, the module is connected directly to a high-performance system bus, otherwise data are transferred across one or several bus bridges increasing the latency. Furthermore, because bus is shared between all system units, performance is significantly decreased. From the security point of view, point-to-multipoint bus is less suitable for security applications because data transfers between security unit and processor can be potentionally eavesdropped by other units connected to the same bus.

IV. Implementation of the security module

The implementation of the security module is illustrated in Fig. 2. Three zones (the processor interface, the cipher and the key storage zone) can be clearly distinguished. Three type of buses are used: data bus (in black), key transfer bus (in grey) and cipher key bus (in dashed grey). It is important that key buses never pass through the processor interface zone and data buses never pass through the key storage zone. The module is organized so that secret keys can never leave the key zone without passing through the cipher. This strict separation on the architectural and physical level guarantees a very high security level.

As presented in Sec. II, any encryption algorithm can be used in the security module. In order to validate the principle, we use a 128-bit Advanced Encryption Standard (AES). Input data, keys and output data are registered in cipher input/output registers. The cipher has a 128-bit folded data path and encryption is completed in 11 clock cycles.

Keys are organized in two hierarchical levels. High-level master keys for encryption and authentication are stored in the master key register. These keys are initialized via dedicated key input during system initialization by the trusted entity. Master keys are used solely for encryption, decryption and authentication of low-level session keys. Session keys are generated inside the module by a TRNG (any principle can be used) or received from the processor and decrypted and authenticated using master keys. When generated inside the module, keys are post-processed in the decipher core.

Session keys are used only for encryption/decryption (using cipher modes) and authentication of data (e. g. using CBC-MAC mode).

The authentication of keys is supported directly inside the module by including a comparator (CMP) that is responsible for comparison of digital fingerprints. The security module is controlled by a local control unit (CTRL) that interprets the fetched instructions. Comparison results or cipher status flags are saved in the status register, which can be read by the processor. This principle of demand-response dialog permits a secure and high-performance operation of the module.

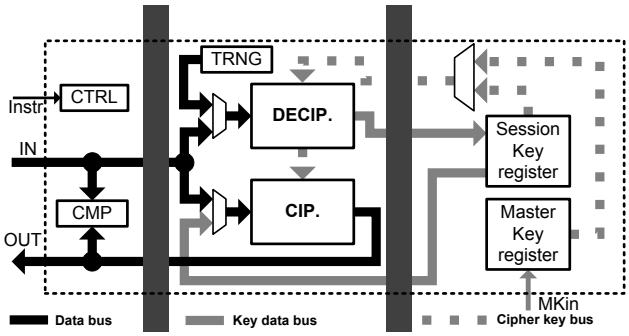

Figure 2: Security module implementation

A. Communication algorithm

The communication protocol between two devices denoted as side A and B is illustrated in Fig. 3 where tasks 1, 2, 3, 5, 9 and 10 are performed solely by the security module. On the other hand, tasks 6, 7, 8 are executed only by the GPP. Tasks 4 and 11 are performed partly by the GPP (xor, cmp) and partly by the security module (AES ciphering).

Before establishing the communication channel between sides A and B, master key registers of the communicating security modules have to be initialized with the same encryption (MK) and authentication keys (AMK) by a trusted entity TE.

Subsequently, the device that starts the communication (A) has to generate new session key SK. The session key CSK is generated by the TRNG inside the security module on the communication side A. The key CSK is then post-processed in the decipher, using a master key MK, and stored in the session key register. Afterwards, a digital fingerprint FP_A is generated by enciphering the generated key using authentication master key AMK.

When both session key SK and its fingerprint FP_A are generated, the data blocks ($DATA$) can be encrypted using the session key SK and sent to the GPP. The GPP implements cipher modes by processing subsequent data blocks according to the selected algorithm. Finally, the processor creates a packet P. This packet contains generated session key CSK encrypted with the master key MK, its fingerprint FP_A, and encrypted data $CDATA$. At the end of the transaction, the packet P is sent to communication partner B.

Processor B receives the packet P, recognizes the packet header and extracts the encrypted session key CSK with its digital fingerprint FP_A. The key is sent to the security module, where it is decrypted using master key MK and stored in the session key register. The fingerprint FP_B of the session key SK is computed by enciphering it with the master authentication key AMK. The processor sends the received fingerprint FP_A to the security module, where it is compared with the computed fingerprint FP_B. If the fingerprints match, the receiving processor can decrypt data using the acquired session key SK by executing selected cipher mode operations.

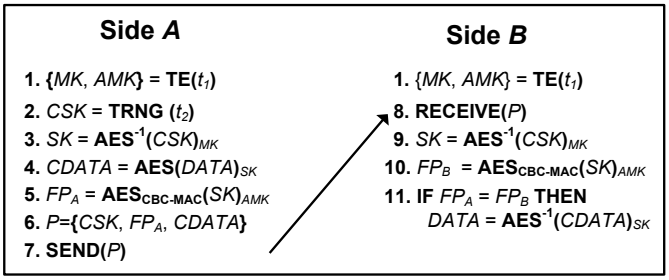

Figure 3: Communication protocol between two devices

V. Extension of selected soft-core processors

Proposed security extensions were implemented in three FPGA families using Altera NIOS II, Xilinx MicroBlaze and Actel version of Cortex M1 soft processors. Each processor system included the same security module that was connected to the processor using a wrapper, which was compatible with the processor interface. The wrapper is responsible for the translation of the control commands and the bus width conversion (128-bit security module data bus is transformed into 32-bit processor bus). Both communication interfaces use point-to-point communication increasing device security.

A. NIOS II with the security module extension

Implementation of the NIOS II processor with its security extension is illustrated in Fig. 4. All security module operations are implemented as custom instructions. Data are transfered from the processor register in 32-bit words via the wrapper. When the instruction execution is finished, data from the security module are sent back to the processor, again in 32-bit blocks.

Because of the use of custom instructions, the NIOS II control unit drives signals that control directly the operation of the security module. This direct connection of the control interface eliminates unwanted latency increase, thus accelerating the execution of custom instructions.

Since the security module is included in the NIOS II data path, the critical path of the processor is extended by the data path of the security module. This directly affects the processor maximal clock frequency.

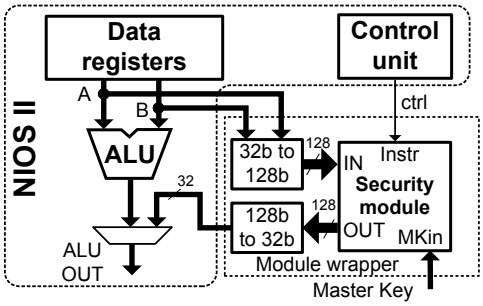

Figure 4: Interfacing NIOS II with the security module

B. MicroBlaze with the security module extension

Implementation of the MicroBlaze processor with its security extension is illustrated Fig. 5. In contrast to the NIOS II processor, custom instruction set implementation is not possible. However, MicroBlaze architecture supports a high-performance coprocessor-like Fast Simplex Link (FSL) bus, which allows interfacing external modules with processor registers. The FSL bus is 32-bit wide. Therefore 128-bit data blocks have to be divided into four 32-bit blocks before being transfered to the security module.

Unfortunately, FSL standard doesn't define any control interface, so that control signals cannot pass directly from the MicroBlaze control unit to the security module. Control instructions are transfered with data to the security module via the FSL data bus and FIFOs. Before each operation, one 32-bit instruction word has to be sent. However, this operation imposes an additional instruction on the program code, what slows down the code execution and data exchange between the processor and the security module. On the other hand, the security module is not a part of the processor's critical path, thus the clock frequency of the processor is not affected. Despite the fact that FIFOs insert additional latency, they enable separation of processor and security module clock domains, thus security module can run on a higher clock frequency than the complex processor.

C. Cortex M1 with the security module extension

Implementation of the Cortex M1 processor with its security module is illustrated Fig. 6. Unlike the previous two processors, Cortex M1 doesn't allow implementation of any point-to-point interface interface suitable for security module implementation. The only way to add custom blocks to the Cortex M1 system is to interface them through an AMBA bus. The AMBA specification describes two types of buses – AHB and APB bus. The APB bus is not considered in this work because of its lower performance. The AHB

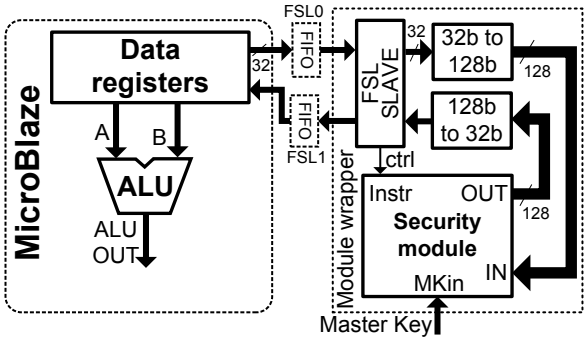

Figure 5: Interfacing MicroBlaze with the security module

bus is a high-performance point-to-multipoint system bus used by the Cortex M1 for instruction fetching as well as I/O data transfers, RAM access (internal/external) and communication with all high-performance peripherals.We use 32-bit AHB bus, therefore each 128-bit data block must be divided into four 32-bit blocks before being transferred to the security module.

Although AMBA specification doesn't describe control interface for transfer of instructions, instructions generated by the Cortex M1 control unit can pass to the security module using AHB control signals. This advantage permits transferring of control instructions and data in parallel. On the other hand, the AHB bus is shared among several bus slaves (program flash memory, RAM memory, etc.), therefore data exchange rate with the security module is significantly decreased.

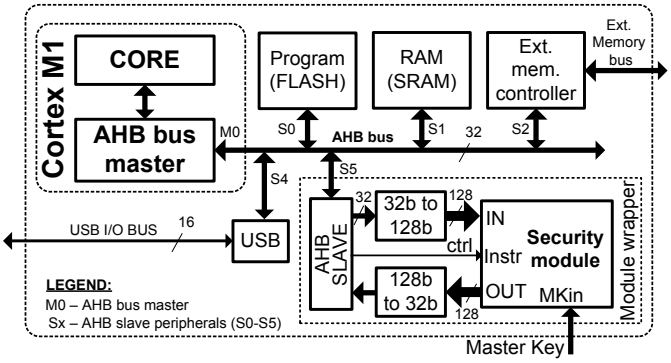

Figure 6: Interfacing Cortex M1 with the security module

VI. Results

The three processors with their security extension modules presented in the previous section were described in VHDL and mapped to three FPGA families. The NIOS II system was implemented in Altera NIOS II evaluation board featuring Stratix II device EP2S60F672C5ES. The project was compiled and mapped to the selected device using Quartus

II version 9.2. The MicroBlaze system and its extension were implemented in Xilinx ML605 evaluation kit featuring Virtex 6 device XC6VLX240TFF1156. For synthesis and mapping, ISE version 12.4 was used. The Cortex M1 system and its extension were implemented in Actel Fusion embedded development kit board featuring Actel Fusion device M1AFS1500-FGG484. The project was compiled and mapped to the selected device using Libero version 8.5 SP2. A small hardware module including the Cypress EZ USB interface device CY7C68013A was connected to all three evaluation boards for data transfers from/to the PC.

The implementation results concerning the logic area and memory requirements are presented in Tab. I and Tab. II. The area is expressed in number of occupied Adaptive Logic Modules (ALM) for Altera family, Slices for Xilinx family and Tiles for Actel family. For comparison, we recall that one ALM in Altera Stratix II family contains two 4-input Look-up tables (LUTs) and two flip-flops (FFs). One Slice in Xilinx Virtex 6 family contains four 6-input LUTs and eight FFs. One Tile in Actel Fusion family contains either one 3-input combinatorial function or one FF. Therefore, the results cannot be directly compared. The memory requirements are given in kbits for all technologies. Besides the processor and its security extension, a small block containing 16-bit data interface to the external Cypress USB device was embedded in all systems. It was used only for testing purposes and it does not constitute an inherent part of the system. It is therefore not included in Tab. I and Tab. II. For clarity, we present the results for the processor and for its extension separately. The results are discussed in the next section.

In order to compare the achieved throughput fairly, the clock frequency of all three systems was set to 50 MHz. The throughput was evaluated by transferring packets from the PC to the FPGA (and vice versa) via USB interface. Each packet contained an encrypted session key, its digital fingerprint and five 128-bit payload blocks. Packets were analyzed in the processor, which then sent the session key and its fingerprint to the security module. Once the key was decrypted and authenticated, the processor sent data blocks to be decrypted. Subsequently, the processor recreated new packets containing received decrypted data and sent them back to the PC. When implementing this complete protocol, the NIOS II-based system achieved the overall throughput of 25,1 Mb/s, the MicroBlaze-based system achieved 18,4 Mb/s and the Cortex M1 system achieved 12,2 Mb/s.

The security of all three solutions depends on the type of interface between security module and processor. Since separation of the key and processor zones was achieved on the protocol and architectural level, all three implementations should be robust against software and timing violation attacks. In order to validate this assumption, software attacks were implemented in tests by reordering and/or replacing instructions. However, these tests have shown that during CBC decryption mode, replacement of GPP instructions

Table I: FPGA fine-grain resource utilization

	Extended NIOS II (ALMs)	**Extended MicoBlaze (Slices)**	**Extended Cortex M1 (Tiles)**
System total	2531	1954	15053
→ Processor	1204	1350	9433
→ Sec. module	1327	604	5620
Ext. overhead	110.2%	44.7%	59.6%

Table II: FPGA embedded RAM utilization

	Extended NIOS II (RAM kb)	**Extended MicoBlaze (RAM kb)**	**Extended Cortex M1 (RAM kb)**
System total	243.9	1206.0	216
→ Processor	187.9	774.0	104
→ Sec. module	56.0	432.0	112
Ext. overhead	29.8%	55.8%	107.7%

led to a substitution of data by encrypted session keys previously stored in the GPP registers and their subsequent decryption inside the security module. It was clear that CBC mode can not be fully supported, otherwise security backdoor is created. Implementation of security module enabling the CBC decryption mode backdoor is illustrated in the Fig. 7. For the sake of security, CBC decryption mode is no longer supported in the implementation presented in this paper. However, CBC mode can be still used for authentication because CBC-MAC standard does not use the Decipher (encryption is safe). Furthermore, the timing violation attack was carried out by increasing the clock frequency over its maximum allowed value. This technique allowed to introduce faults to the control logic of the security module, however no key fragments were acquired from the module. The tests confirmed the robustness of the proposed techniques of separation that were shown to be secure-by-design.

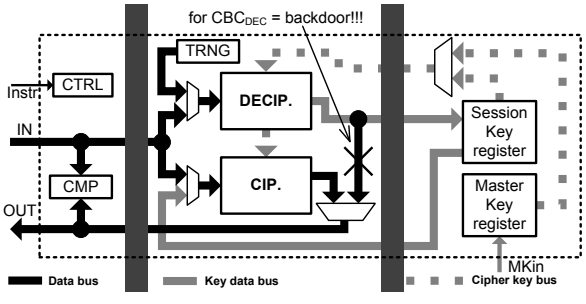

Figure 7: Security module with CBC decryption mode backdoor

VII. Discussion

Area requirements for NIOS II, MicroBlaze and Cortex M1 extensions from Tab. I and Tab. II seem to be different. This is due to differences between ALMs, slices and tiles in Altera, Xilinx and Actel families and also because of the size of the soft processors. In the Altera FPGA, the security module area is similar to that of the NIOS II processor (1327 versus 1204 ALMs giving 110%). However, since the security module included both AES cipher and decipher cores, we can conclude that the sole security extension cost due to zones separation is negligible. On the other hand, the MicroBlaze processor occupies bigger area and the security module overhead is only 45%. The separation cost remains negligible. In Cortex M1 system, the security module represents a 60% overhead therefore cost remains negligible.

As presented in Section VI, the MicroBlaze processor with its extension achieves approximately 73% of the throughput of the NIOS II implementation. This is caused by more complicated communication protocol (data and control words) across the FSL bus in MicroBlaze, compared to the straightforward custom instruction implementation in NIOS. This difference could be reduced if data were transferred to the security module using DMA transfers. This way, the FSL bus would serve only for transporting instructions to the security module. Moreover, the Cortex M1 processor with its security module extension achieves only 49% of the throughput of the NIOS II implementation. This is because of the nature of the AHB bus that is shared among all communicating units. Therefore access of the security module to the AHB bus is multiplexed with flash memory, RAM and I/O (USB). This fact limits significantly overall system performance.

Another useful aspect of comparing the three processor extensions is the system design complexity. When interfacing the NIOS II processor with the security module, several actions are needed: 1) the processor data path and control signals are output from the processor; 2) a wrapper containing simple control interface and data bus translation interface has to be implemented; 3) the wrapper with the security unit has to be interfaced with the NIOS II processor using SOPC builder software; 4) custom instructions have to be generated and described in the software implementation. We can conclude that the design is relatively complex, but as a result, the security module is closely tied to the processor and thus maintaining high performance. Connecting the security module to the MicroBlaze processor is simpler on hardware level, but more complex on protocol level: more complex state machine has to be implemented in the wrapper to pick up control instruction from the data stream transported via the Fast Simplex Link. Because of this fact, the overall speed of the system is significantly lower. Connecting security module via a peripheral bus is the most general solution. Once the data interface is designed, the module can be reused with any processor featuring a common peripheral bus. However, in this case, the speed of the system is the lowest.

Attacks that were carried out revealed a vulnerability in the CBC decryption mode and this backdoor was eliminated by disconnecting Decipher output from the output of the security module. Nevertheless, CBC decryption mode is often used nowadays and its secure implementation remains an open problem. Timing violation attacks that were carried out against the processor implementations were not successful. The implemented protocol and architectural separations of the key and processor zones were therefore shown to be efficient. Furthermore, higher security level could be achieved if the security zones were physically isolated on the chip. Although this physical isolation by insulation walls was not realized, it can be easily done in the future because of existing strict separation of modules.

The principle presented in this paper concerns three soft core GPPs. We believe that it can be extended to other GPPs. However, faster and perhaps more secure implementations can be obtained by creating a specific-purpose processor such as the one published in [13], which takes advantage of reduced instruction set oriented in cryptographic operations. Nevertheless, this custom processor has to comply with all the above mentioned separation techniques.

Since encryption standards are always evolving, it is important that the cipher implementation inside the security module could be easy to update. One of possible solutions could be to use a dynamic reconfiguration technique. This technique allows for creation of a special dynamically reconfigurable zone reserved for the complete Cipher zone from Fig.1. Afterwards, the enciphering algorithm can be updated on the fly. It is very useful that the used physical separation techniques are similar to those used in dynamic reconfiguration. Thanks to this, only a small effort is necessary when implementing reconfigurability features in the device. Moreover, the zone that is dynamically reconfigurable can also be easily isolated from surrounding modules by creating an empty space that will serve as an insulation wall around the secured reconfigurable zone.

VIII. Conclusion

We have proposed a novel principle allowing general-purpose processors to operate with secret keys in a highly secure way. The principle is based on the creation of separated processor, cipher and key zones. Separation is implemented on the protocol, system, architectural and physical levels, and it guarantees that unencrypted keys can never be transfered from protected key zone to unprotected processor zone. The only way to transfer the keys to the processor zone is to pass across the cipher zone: the keys are encrypted before entering the processor zone and must be decrypted when going in opposite direction (when entering the memory

zone). The proposed solution enhances security substantially when compared with existing soft-core cryptographic extensions.

The separation principle was implemented in FPGAs and tested using NIOS II, MicroBlaze and Cortex M1 processors. The obtained throughput including the processing of packets, key management and data encryption/decryption and authentication was about 25, 18 and 12 Mb/s, respectively. This speed was limited mainly by processors and their data interfaces. The area of the system increased by 110% when compared with the smaller NIOS II processor, by 44% when the MicroBlaze processor was taken as a basis and by 60% when Cortex M1 was extended using the security module.

ACKNOWLEDGMENT

The work presented in this paper was realized in the frame of the SecReSoC project number ANR-09-SEGI-013, supported by the French National Research Agency (ANR).

REFERENCES

[1] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF (2^n)," *IEEE Transactions on Computers*, pp. 1269–1282, 2007.

[2] M. Machhout, Z. Guitouni, K. Torki, L. Khriji, and R. Tourki, "Coupled FPGA/ASIC Implementation of Elliptic Curve Crypto-Processor," *International Journal*, vol. 2.

[3] F. Crowe, A. Daly, T. Kerins, and W. Marnane, "Single-chip FPGA implementation of a cryptographic co-processor," in *2004 IEEE International Conference on Field-Programmable Technology, 2004. Proceedings*, 2004, pp. 279–285.

[4] Y. Eslami, A. Sheikholeslami, P. Gulak, S. Masui, and K. Mukaida, "An area-efficient universal cryptography processor for smart cards."

[5] M. Hani, H. Wen, and A. Paniandi, "Design and implementation of a private and public key crypto processor for next-generation it security applications," *Malaysia Journal of Computer Science*, vol. 19, no. 1, pp. 29–45, 2006.

[6] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *CRYPTO99*, pp. 789–789, 1999.

[7] F. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J. Quisquater, "Power analysis of FPGAs: How practical is the attack?" *FPL'03*, pp. 701–711, 2003.

[8] E. Bangerter, D. Gullasch, and S. Krenn, "Cache games–Bringing access-based cache attacks on AES to practice," *Workshop COSADE*, pp. 215–221, 2011.

[9] A. Ashkenazi and D. Akselrod, "Platform independent overall security architecture in multi-processor system-on-chip integrated circuits for use in mobile phones and handheld devices," *Computers & Electrical Engineering*, vol. 33, no. 5-6, pp. 407–424, 2007.

[10] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic processors - a survey," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 357–369, 2006.

[11] J. M. McConnell, "TEMPEST/2-95," *NSTISSAM*, 1995.

[12] P. Cotret, J. Crenne, G. Goniat, J.-P. Diguet, L. Gaspar, and G. Duc, "Distributed security for communications and memories in a multiprocessor architecture," *Workshop RAW*, 2011.

[13] L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret, "HCrypt: A Novel Concept of Crypto-processor with Secured Key Management," *ReConFig'10*, pp. 280–285, 2010.