Crypto-processor – architecture, programming and evaluation of the security

Lubos Gaspar

Thèse

pour obtenir le grade de docteur
de Université Jean Monnet

Discipline : **Image, Vision, Signal**
Équipe : **Systèmes Embarques Sécurisés**

Crypto-processeur – architecture, programmation et évaluation de la sécurité

Lubos Gaspar

La thèse a été soutenue le 16 Novembre, 2012 avec le jury suivant

Ingrid Verbauwhede	**KUL, Belgium**	**Rapporteur**
Lionel Torres	**LIRMM, France**	**Rapporteur**
François-Xavier Standaert	**UCL, Belgium**	**Examinateur**
Tim Güneysu	**RUB, Germany**	**Examinateur**
Yannick Teglia	**STM, France**	**Examinateur**
Lilian Bossuet	**LaHC, France**	**Examinateur**
Viktor Fischer	**LaHC, France**	**Directeur de thèse**
Florent Bernard	**LaHC, France**	**Codirecteur de thèse**

PhD Thesis

presented to obtain the doctor degree
of Jean Monnet University

Speciality: **Image, Vision, Signal**
Group: **Secure Embedded Systems**

Crypto-processor – architecture, programming and evaluation of the security

---

Lubos Gaspar

The thesis was defended on November 16th, 2012 in front of the following committee

Ingrid Verbauwhede	**KUL, Belgium**	**Reviewer**
Lionel Torres	**LIRMM, France**	**Reviewer**
François-Xavier Standaert	**UCL, Belgium**	**Member**
Tim Güneysu	**RUB, Germany**	**Member**
Yannick Teglia	**STM, France**	**Member**
Lilian Bossuet	**LaHC, France**	**Member**
Viktor Fischer	**LaHC, France**	**Supervisor**
Florent Bernard	**LaHC, France**	**Joint Supervisor**

Résumé

Les architectures des processeurs et coprocesseurs cryptographiques se montrent fréquemment vulnérables aux différents types d'attaques ; en particulier, celles qui ciblent une révélation des clés chiffrées. Il est bien connu qu'une manipulation des clés confidentielles comme des données standards par un processeur peut être considérée comme une menace. Ceci a lieu par exemple lors d'un changement du code logiciel (malintentionné ou involontaire) qui peut provoquer que la clé confidentielle sorte en clair de la zone sécurisée. En conséquence, la sécurité de tout le système serait irréparablement ménacée. L'objectif que nous nous sommes fixé dans le travail présenté, etait la recherche d'architectures matérielles reconfigurables qui peuvent fournir une sécurité élevée des clés confidentielles pendant leur génération, leur enregistrement et leur échanges en implantant des modes cryptographiques de clés symétriques et des protocoles.

La première partie de ce travail est destinée à introduire les connaissances de base de la cryptographie appliquée ainsi que de l'élécronique pour assurer une bonne compréhenension des chapitres suivants.

Deuxièmement, nous présentons un état de l'art des menaces sur la confidentialité des clés secrètes dans le cas où ces dernières sont stockées et traitées dans un système embarqué. Pour lutter contre les menaces mentionnées, nous proposons alors de nouvelles règles au niveau du design de l'architecture qui peuvent augmenter la résistance des processeurs et coprocesseurs cryptographiques contre les attaques logicielles. Ces règles prévoient une séparation des registres dédiés à l'enregistrement de clés et ceux dédiés à l'enregistrement de données : nous proposons de diviser le système en zones : de données, du chiffreur et des clés et à isoler ces zones les unes des autres au niveau du protocole, du système, de l'architecture et au niveau physique.

Ensuite, nous presentons un nouveau crypto-processeur intitulé HCrypt, qui intègre ces règles de séparation et qui assure ainsi une gestion sécurisée des clés. Mises à part les instructions relatives à la gestion sécurisée de clés, quelques instructions supplémentaires sont dédiées à une réalisation simple des modes de chiffrement et des protocoles cryptographiques.

Dans les chapitres suivants, nous explicitons le fait que les règles de séparation suggérées, peuvent également être étendues à l'architecture d'un processeur généraliste et coprocesseur. Nous proposons ainsi un crypto-coprocesseur sécurisé qui est en mesure d'être utilisé en relation avec d'autres processeurs généralistes. Afin de démontrer sa flexibilité, le crypto-coprocesseur est interconnecté avec les processeurs soft-cores de NIOS II, de MicroBlaze et de Cortex M1.

Par la suite, la résistance du crypto-processeur par rapport aux attaques DPA est testée. Sur la base de ces analyses, l'architecture du processeur HCrypt est modifiée afin de simplifier sa protection contre les attaques par canaux cachés (SCA) et les attaques par injection de fautes (FIA). Nous expliquons aussi le fait qu'une réorganisation des blocs au niveau macroarchitecture du processeur HCrypt, augmente la résistance du nouveau processeur HCrypt2 par rapport aux attaques de type DPA et FIA.

Nous étudions ensuite les possibilités pour pouvoir reconfigurer dynamiquement les parties sélectionnées de l'architecture du processeur – crypto-coprocesseur. La reconfiguration dynamique peut être très utile lorsque l'algorithme de chiffrement ou ses implantations doivent être changés en raison de l'apparition d'une vulnérabilité.

Finalement, la dernière partie de ces travaux de thèse, est déstinée à l'exécution des tests de fonctionnalité et des optimisations stricts des deux versions du crypto-processeur HCrypt.

Abstract

Architectures of cryptographic processors and coprocessors are often vulnerable to different kinds of attacks, especially those targeting the disclosure of encryption keys. It is well known that manipulating confidential keys by the processor as ordinary data can represent a threat: a change in the program code (malicious or unintentional) can cause the unencrypted confidential key to leave the security area. This way, the security of the whole system would be irrecoverably compromised. The aim of our work was to search for flexible and reconfigurable hardware architectures, which can provide high security of confidential keys during their generation, storage and exchange while implementing common symmetric key cryptographic modes and protocols.

In the first part of the manuscript, we introduce the bases of applied cryptography and of reconfigurable computing that are necessary for better understanding of the work.

Second, we present threats to security of confidential keys when stored and processed within an embedded system. To counteract these threats, novel design rules increasing robustness of cryptographic processors and coprocessors against software attacks are presented. The rules suggest separating registers dedicated to key storage from those dedicated to data storage: we propose to partition the system into the data, cipher and key zone and to isolate the zones from each other at protocol, system, architectural and physical levels.

Next, we present a novel HCrypt crypto-processor complying with the separation rules and thus ensuring secure key management. Besides instructions dedicated to secure key management, some additional instructions are dedicated to easy realization of block cipher modes and cryptographic protocols in general.

In the next part of the manuscript, we show that the proposed separation principles can be extended also to a processor-coprocessor architecture. We propose a secure crypto-coprocessor, which can be used in conjunction with any general-purpose processor. To demonstrate its flexibility, the crypto-coprocessor is interconnected with the NIOS II, MicroBlaze and Cortex M1 soft-core processors.

In the following part of the work, we examine the resistance of the HCrypt cryptoprocessor to differential power analysis (DPA) attacks. Following this analysis, we modify the architecture of the HCrypt processor in order to simplify its protection against side channel attacks (SCA) and fault injection attacks (FIA). We show that by rearranging blocks of the HCrypt processor at macroarchitecture level, the new HCrypt2 processor becomes natively more robust to DPA and FIA.

Next, we study possibilities of dynamically reconfiguring selected parts of the processor - crypto-coprocessor architecture. The dynamic reconfiguration feature can be very useful when the cipher algorithm or its implementation must be changed in response to appearance of some vulnerability.

Finally, the last part of the manuscript is dedicated to thorough testing and optimizations of both versions of the HCrypt crypto-processor.

Architectures of crypto-processors and crypto-coprocessors are often vulnerable to software attacks targeting the disclosure of encryption keys.

The thesis introduces separation rules enabling crypto-processor/coprocessors to support secure key management. Separation rules are implemented on novel HCrypt crypto-processor resistant to software attacks targetting the disclosure of encryption keys.

Acknowledgements

First and foremost, I would like to thank Diana, the most beautiful discovery I have made, for her unconditional love, encouragement, patience, help and for reminding me that there is much more to life than study. I dedicate this dissertation also to my parents Vladimir, Emilia and my brother Vladimir who have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

I am very grateful to the dissertation reviewers Lionel Torres and Ingrid Verbauwhede for their time spent on reading the thesis and their valuable comments, and also to all dissertation committee members for their valuable effort and time: Florent Bernard, Lilian Bossuet, Viktor Fischer, Tim Güneysu, François-Xavier Standaert, Yannick Teglia, Lionel Torres and Ingrid Verbauwhede.

This thesis would not have been possible without help, guidance, support and patience of my principal supervisor, professor Viktor Fischer. He is the one who has given me many invaluable advices on both an academic and a personal level, dedicated a great portion of his time to reading and correcting all my works, and enabled me to meet many great people in cryptography research domain. For all this and much more, I am greatly indebted to him. Thank you sir, dakujem!

Many thanks go to my second supervisor, Florent Bernard, for his good advice, friendship and help on the administration battlefield. I am especially grateful to all other colleagues in the Secure Embedded Systems group for making my days much brighter, for creating pleasant working environment in the laboratory, and for helping me to improve my French: Alain Aubert, Pierre Bayon, Nathalie Bochard, Lilian Bossuet, Pierre-Louis Cayrel, Karim Cherkaoui, Robert Fouquet, Patrick Haddad, Abdourhamane Idrissa, Tania Richmond and Boyan Valtchanov. I also thank the research and staff members of the Laboratoire Hubert Curien in Saint-Etienne.

This work, in the frame of the project SecReSoC (ARPEGE 2009 program, ANR-09-SEGI-013), would not have been possible without the support of the ANR (French National Research Agency). I thank also to the SecReSoC team members: Benoît Badrignans, Lyonel Barthe, Pascal Benoit, Pascal Cotret, Jean-Luc Danger, Florian Devic, Viktor Fischer, Guy Gogniat, Houssem Maghrebi and Lionel Torres.

I thank Tim Güneysu for inviting me to Bochum, for fruitful discussions and many good advices. I would like to acknowledge all the members of the EMSEC and Hardware security groups.

I acknowledge Marek Repka from Slovak Technical University of Bratislava for excellent cooperation and many passionate discussions producing interesting scientific results.

I could not have obtained excellent practical knowledge in complex PCB designs without guidance and support of Micronic company in Trebejov, Slovakia.

This way I would also like to express my sincere gratitude to Milos Drutarovsky from Technical University of Kosice in Slovakia for introducing me into the great world of cryptography, microprocessors, and FPGAs during my master studies.

Last but by no means least, I would like to thank to all my friends in Saint-Etienne for all those great unforgettable moments we had during many different social events, sports activities and trips. Thank you Ali, Alius, Anton, Christina, Claire-Marie, Fanny, Hassan, Iseline, Johanns, Johari, Katka, Laura, Lina, Maggie, Mathias, Max, Merike, Monika, Montse, Noelia, Omar, Rosaura, Sandra, Sergey, Silvia, Sona, Tomek, Tristan, Zdenka and many others!

Contents

List of Figures

List of Tables

Glossary

ADD:	Addition
AES:	Advanced Encryption Standard
AES-NI:	Advanced Encryption Standard - New Instructions
AHB:	Advanced High-performance Bus
ALM:	Adaptive Logic Module (In Altera Stratix-II)
ALU:	Arithmetic Logic Unit
AMK:	Authentication Master Key
ANSI:	American National Standards Institute
APB:	Advanced Peripheral Bus
ASIC:	Application Specific Integrated Circuit
BB:	Bitstream Bus
BRAM:	Block RAM (in Xilinx Virtex)
CBC:	Cipher Block Chaining
CBC-MAC:	Cipher Block Chaining - Message Authentication Code
CC:	Common Criteria
CCM:	Counter mode with CBC-MAC
CDONE:	Configuration Done flag
CFB:	Cipher FeedBack mode
CIP:	Cipher
CLB:	Configurable Logic Block (in Xilinx Virtex)
CLK:	Clock signal
CMT:	Clock Management Tile (in Xilinx Virtex)
CLR:	Clear
CMAC:	Cipher Block Chaining Message Authentication Code
CMOS:	Complementary Metal Oxid Semiconductor
CMP:	Compare
CPA:	Correlation Power Analysis
CPU:	Central Processing Unit
CSEL:	Configuration Select
CSK:	Ciphered Session Key
CSP:	Critical Security Parameter
CSTART:	Configuration Start
CT:	Ciphertext
CTI:	Ciphertext Input
CTO:	Ciphertext Output
CTR:	Counter mode
CTRL:	Control
DCM:	Digital Clock Manager (in Xilinx Virtex)

DEC: Decrement
Decip: Decipher
DES: Data Encryption Standard
DFA: Differential Fault Analysis
DFF: D Flip-Flop
DH: Diffie-Hellmann
DI: Data Input
DLL: Delay-Locked Loop (in Xilinx FPGAs)
DO: Data Output
DPA: Differential Power Analysis
DPRAM: Dual Port RAM
DRM: Digital Rights Management
DRNG: Deterministic Random Number Generator
DRP: Dual-Rail Precharge logic
DSP: Digital Signal Processing
DVLD: Data Valid flag
EAL: Evaluation Assurance Level (in CC)
ECB: Electronic Code Book
ECC: Elliptic Curve Cryptography
EM: Encryption Modes
EMA: Electromagnetic Analysis
ENA: Enable signal
FF: Flip-Flop
FIA: Fault Injecition Attacks
FIFO: First In First Out
FIPS: Federal Information Processing Standard
FP: Fingerprint (in communication protocol)
FPGA: Field Programmable Gate Arrays
FSL: Fast Simplex Link
GCD: Greatest Common Divisor
GCM: Galois/Counter Mode and GMAC
GPP: General-Purpose Processor
HCrypt: Hubert Curien Crypto-processor
HCrypt-C: Hubert Curien Crypto-coprocessor
HD: Hamming Distance
HMAC: Keyed-Hashing MAC
HRNG: Hybrid Random Number Generator
HW: Hamming weight (in SCA)
IAP: In-Application Programmability (in Microsemi SmartFusion)
ICAP: Internal Configuration Access Port (in Xilinx Virtex)
ID: Identification Code
I/F: Interface
INC: Increment
I/O: Input/Output

IP:	Intellectual Property (in general)
IV:	Initialization Vector
IVT:	Isolation Verification Tool
JTAG:	Joint Test Action Group
KI:	Key Input
KO:	Key Output
LAB:	Logic Block Array (In Altera Stratix-II)
LFSR:	Linear Feedback Shift Register
LUT:	Look-Up Table
MAC:	Message Authentication Code
MC:	Mix Columns (in AES)
MCCP:	Multi-Core Crypto-Processor
MD5:	Message Digest algorithm 5
MIM:	Man In the Middle
MK:	Master Key
MMCM:	Mixed-Mode Clock Management
MPSoC:	Multi-Processor System on Chip
NIST:	U.S. National Institute of Standards and Technology
NSA:	U.S. National Security Agency
OFB:	Output FeedBack
PBB:	Partial Bitstream Bus
PC:	Personal Computer
PC1:	Permutation Choice 1 (in DES)
PC2:	Permutation Choice 2 (in DES)
PIP:	Programmable Interconnect Points
PK:	Public Key
PLL:	Phase Locked Loop
PM:	Personalization Module
PP:	Partition Pin
PR:	Partial Reconfiguration
PRNG:	Pseudo-Random Number Generator
PRP:	Partial Reconfiguration Port
PSK:	Protected Session Key
PT:	Plaintext
PTI:	Plaintext Input
PTO:	Plaintext Output
RAM:	Random Access Memory
RC:	Resistor-Capacitor oscillator
Rcon:	Remote Constant (in AES)
RCU:	Reconfiguration Control Unit
RD:	Read
RK:	Round Key
RM:	Reconfigurable Module
ROL:	Rotation to Left

ROM:	Read-Only Memory
RP:	Reconfiguration Port
RSA:	Rivest Shamir Adleman algorithm
SB:	SubBytes operation (in AES)
SCA:	Side-Channel Attack
SCC:	Single Chip Crypto
SHA:	Secure Hash Algorithm
SK:	Session Key (in general)
SKF:	Secure Key Flash
SL:	Shift to Left (in DES)
SoC:	System on Chip
SPA:	Simple Power Analysis
SPP:	Specific-Purpose Processor
SR:	Shift Rows (in AES)
SRAM:	Static RAM
STS:	Station-to-Station protocol
TBM:	Trusted Bus Macro
TDEA:	Triple Data Encryption Algorithm
TDPRAM:	True Dual Port RAM
TE:	Trusted Entity
TERO:	Transition Effect Ring Oscillator
TOE:	Target Of Evaluation (in CC)
TPM:	Trusted Platform Module
TRNG:	True Random Number Generator
USB:	Universal Serial Bus
VHDL:	VHSIC Hardware Description Language
VHSIC:	Very-High-Speed Integrated Circuit
WR:	Write
VLIW:	Very Long Instruction Word
XOR:	Exclusive OR

Chapter 1

Introduction

1.1 Flexible Security and Hardware Implementations

Nowadays, demands of data security are increasing, especially after introduction of wireless communications to the masses. Technologies like WIFI, Bluetooth, UMTS,... are all widely used. Moreover, usage of portable devices is skyrocketing. Particularly, in this lucrative market sector combination of reconfigurable, low power logic solution with embedded crypto-processor (hardware accelerator) and random number generator are extremely attractive. Furthermore, if being able to guarantee security on physical implementation level (concept of the black/red zone separation) as well as other countermeasure techniques, these solutions can surpass ordinary consumer electronics application to be used in avionics, automotive and military applications featuring higher security needs.

Hardware cryptographic systems must fulfill contradictory requirements: fast parallel structures implementing computationally extensive cryptographic functions must coexist with complex sequential structures used to implement cryptographic algorithms such as cipher modes, key management operations and cryptographic protocols. Implementation of cryptographic algorithms and protocols in hardware necessitates employing many complex state machines that make the logic vulnerable. Furthermore, upgrades of hardwired logic can become complicated, long and expensive. On the other hand, security of the system itself and protection of confidential data is often underestimated.

The most common solution consists in the use of a general-purpose processor employing one or more cryptographic coprocessors. This solution permits to implement sequential algorithms (that evolve very frequently as a consequence of attacks and/or evolution of standards) by the processor program, while the tasks that can be executed in parallel are implemented in the coprocessor placed inside the same logic device. However, this solution brings some difficulties concerning the system security: first, the general-purpose processor manipulates the keys as ordinary data and modification (intentional or unintentional) of the program memory contents can enable reading the keys in clear outside the system; second, the use of general-purpose processors does not permit to isolate efficiently the red (unprotected) and black (protected) communication zones inside the device.

One problem concerning the use of a general-purpose processor in cryptographic applications is related to the speed limitation caused by the bus width: data and bus size (usually 32 bits) limits considerably the performance of the system. Another problem related to the use of general-purpose processors in cryptographic systems

and their speed concerns the complexity of the instruction set: processors aimed at cryptographic applications (completed by cryptographic coprocessors) do not need a complex instruction set that is necessary in general-purpose processors and optimization (minimization) of the instruction set could increase the speed of the system. The design of dedicated processors with dedicated instruction set is especially interesting in the case of emerging application areas based on Multi-Processor System-on-Chip (MPSoC).

1.2 Objectives of the Thesis

In order to fulfill aforementioned, often contradictory, requirements on the cryptographic system, French National Research Agency (ANR) founded the Secured Reconfigurable System on Chip project (SecReSoC) proposed by the Secure Embedded Systems research team of the Hubert Curien Laboratory. The aim of SecReSoC was to examine security aspects of MPSoC, and to demonstrate the final MPSoC system implemented in a Field Programmable Gate Array (FPGA). The work presented in this thesis is done in the framework of the SecReSoC project, and its objectives were defined as follows:

1. Propose a new architecture of the crypto-processor that includes cipher block as an independent module and that allows secure separation of data and key registers.
2. Optimize the architecture of the Arithmetic Logic Unit (ALU) for operations used by cryptographic protocols and cipher modes.
3. Propose an instruction set providing separate data and key processing.
4. Propose at least one version of the cipher core including countermeasures against side-channel attacks.
5. Propose some efficient solution enabling easy adaptation of the assembler to the modification/evolution of the instruction set.
6. Integrate all elements necessary for secure programming and reconfiguration into the structure of the crypto-processor.
7. Evaluate the security level of the proposed crypto-processor.

1.3 Contribution

The thesis not only achieves all aforementioned objectives, but also proposes novel principles, design rules and architectures enabling secure key management on cryptographic processors or even general-purpose processors (extended with secure cryptographic coprocessors). The developed cryptographic processors (further just crypto-processors) and cryptographic coprocessors (further just crypto-coprocessors) can

be directly used in practice. A part of the research work is also dedicated to countermeasures against side-channel attacks and protection of a totally or partially reconfigurable system. This work does not propose partial solutions, but its outcome is a fully functional and tested HCrypt cryptographic processor and HCrypt-C cryptographic coprocessor in many variations focused on a specific objective.

Every specific problem in this research work is preceded by an extensive study of the state-of-the-art work which is summarized in every chapter of this thesis.

1.4 Thesis Preview

The manuscript describes the work progress during the thesis in chronological order. Chap. 3–6 constitute the core of the research work.

The objective of Chap. 2 is to introduce the theoretical background which is essential for further understanding of the thesis. First, basic cryptography objectives for secure communications are listed. Next, most important symmetric key cryptographic algorithms (i.e. DES, AES, block cipher modes, etc.) are described, followed by hash functions and random number generators. Subsequently, key management, assumed participating parties and all involved security threats are presented. Consequently, important key establishment schemes are briefly introduced and compared. The second part of Chap. 2 is dedicated to FPGAs and corresponding advanced design flows.

Chap. 3 starts by summarizing the state-of-the-art related to cryptographic processors. Security issues of cryptographic software implementations are discussed first, followed by overview on cryptographic hardware architectures. Further, novel separation rules for secure key management are presented. These rules are implemented on original HCrypt crypto-processor. Subsequently, its hardware and software design, and communication protocol is described, followed by simulations, and hardware tests. Finally, HCrypt properties and implementation results are discussed.

Chap. 4 is oriented on cryptographic coprocessors with secure key management. After introducing the work that has been conducted in the study of crypto-coprocessors, slight modifications of separation rules for crypto-coprocessors are proposed. These separation rules are applied on the HCrypt-C crypto-coprocessor. Subsequently, most common interface topologies for interconnection of a coprocessor to a general-purpose processor are described. Next, three different general-purpose processors are extended by HCrypt-C and their design and results are given.

Side-channel analysis attack (SCA) threat is considered in Chap. 5. After introducing state-of-the-art in hardware attacks, an extensive security analysis of the AES cipher and HCrypt is given. The HCrypt security analysis uncovers security vulnerabilities when considering SCA, and so the HCrypt2 crypto-processor is designed. HCrypt2 implements new countermeasures (zero-cost countermeasures). The HCrypt2 security analysis confirms increased robustness to SCA. HCrypt2 design and results are presented and evaluated.

Chap. 6 contemplates an FPGA-specific advanced technique – partial reconfiguration flow. First, reasons for using this technique are discussed. Next, the work that has been done in the field of partially reconfigurable system is presented. Subsequently, separation rules in conjunction with partial reconfiguration are given. A total and partial reconfiguration of an FPGA and corresponding security implications are discussed. Finally, a partially reconfigurable HCrypt-C crypto-coprocessor is described and tested using three different partial configurations.

Finally, Chap. 7 summarizes all important contributions, proposals and conclusions. It also presents new challenges and perspectives that can be a subject of future studies.

Chapter 2

Theoretical and Technological Background

Contents

This chapter introduces basic knowledge concerning cryptography for secure communications, FPGA technology and advanced design flows. It is necessary for better understanding of the following parts of the manuscript.

2.1 Secure Communications

In general, secure communication can be achieved by two methods: 1) steganography 2) cryptography. Steganography exploits different methods for hiding a secret message inside an insecure message. The secret message can be hidden in digital data like images, videos, music, etc. On the contrary, cryptography does not try to hide the secret message (plaintext) from the attacker, but rather to transform it to a meaningless message (ciphertext) using a key. This transformation is performed by a cipher. Only the owner of the key can transform the ciphertext message back to its plaintext version. The attacker, however, may have knowledge of the ciphering algorithm and try to recover the confidential key or the plaintext using cryptanalysis. For this reason, security should not be based on the confidentiality of the ciphering algorithm, but only on the confidentiality of the key. This Kerckhoffs's principle has a practical reason, because it is easier to change the confidential key than to redesign the ciphering algorithm in case of their disclosure.

Different objectives have been defined for cryptography [4], [5], [6]. The following are the four cryptography objectives defined by Menezes et al. [3]:

1. **Confidentiality** is a service used to keep the content of information from all but those authorized to have it. Secrecy is a term synonymous with confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.

2. **Data integrity** is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.

3. **Authentication** is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

4. **Non-repudiation** is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

This work is focused especially on secure key management in logic devices. The confidential keys are essential piece of information for many cryptographic primitives

and algorithms for fulfilling the described four objectives of cryptography. Next, we describe the cryptographic primitives, algorithms and protocols that are necessary for better understanding of this work.

2.2 Symmetric Key Cryptography

The symmetric key cryptography utilizes the same key for both encryption and decryption. Contrary to the symmetric key cryptography, the asymmetric key cryptography is based on two different keys. One key is confidential and is called private and the second is freely distributed and is called public. This work concerns only the symmetric key cryptography and so the asymmetric key cryptography will not be further considered.

The use of the symmetric key cryptography in military dates back to the Roman empire when the emperor Julius Caesar used a special cipher for his private correspondence. The Caesar cipher is very simple. Each letter in the message has to be substituted by another letter shifted in the alphabet by 3 letters.

Before the digital era, the cryptographic algorithms operated on letters. The digital era allowed to encode any piece of information by a set of bits, and so the ciphering started to be performed on bits. The ciphers that operate on an infinitely long bit sequence are called stream ciphers. The opposite are the block ciphers which operate on a constant number of bits (blocks).

2.2.1 Block Ciphers

Block ciphers can be divided into two classes: substitution ciphers and transposition ciphers.

The substitution ciphers replace the symbols by other symbols using some algorithm or substitution tables. Substitution tables may be defined by the ciphering algorithm or calculated from the key. The Caesar cipher is a typical substitution cipher. The transposition ciphers are rather based on changing the order of symbols in the block (symbol permutations) according to some key-dependent algorithm. Substitution or transpositions ciphers do not provide sufficient security. However, if the two are combined together they can provide a very high level of security.

Two important properties of the block ciphers are confusion and diffusion. Confusion makes the relationship between the plaintext and ciphertext as complex as possible. The substitution operation significantly increases confusion. To achieve high confusion, the substitution transformation should be non-linear. A unit performing substitution is called S-box. Diffusion, however, spreads or rearranges the plaintext bits over the ciphertext. This way any redundancy in the plaintext can be effectively spread out (diffused) over the ciphertext. Diffusion can be increased by transposition operation. Operations used in modern ciphers can be categorized as those increasing confusion or diffusion.

Next, some modern standardized ciphering algorithms will presented.

2.2.1.1 Data Encryption Standard (DES)

The Data Encryption Standard (DES) was developed in the 1970s by IBM in cooperation with the National Security Agency (NSA) of the USA and published as the Federal Information Processing Standard 46 (FIPS-46) [7]. DES is based on another cipher developed by Horst Feistel. For this reason, DES is based on the so called Feistel scheme.

DES is a block cipher which uses a 56-bit long symmetric key and operates on 64-bit data blocks. The transformation of a plaintext block to a ciphertext block is performed in 16 rounds. The structure of the DES cipher is depicted in Fig. 2.1. First, a 64-bit data block is transformed into two 32-bit state words by Initial Permutation (IP). The less significant 32-bit word (right word) passes directly to the next round to the left word position. The more significant 32-bit word (left word) is XOR-ed with the output of the F-function (see the part B in Fig. 2.1) and the result passes into the next round to the right word position. This algorithm is repeated for every round except the last round where the two words are not swapped. Finally, the last two 32-bit words in the last round are transformed into the ciphertext by the Final Permutation (FP).

The structure of the F-function is illustrated in the part B in Fig. 2.1. The F-function loads a 48-bit subkey (SK) and a 32-bit right state word (R). In order to match two bus widths, the 32-bit state word is expanded into 48-bits by the Expansion permutation (E) by duplicating half of the bits. The expanded value is added with the subkey and the resulting 48-bit word enters the substitution part. The substitution function is represented by eight different parallel substitution blocks (S-boxes). Each S-box substitutes six bits with four bits. The resulting substituted values form a 32-bit word which is subsequently transformed by the Permutation (P) to form an output of the F-function. S-boxes together with E and P permutations contribute to high confusion and diffusion, respectively.

48-bit subkeys are generated by the key expansion function (see the part C in Fig. 2.1). Although a 64-bit secret key word enters the key expansion function, 8 bits are removed by the Permutation Choice 1 (PC1) forming a 56-bit secret key divided into two 28-bit halves. These halves are further processes separately. Each half is shifted to left (SL) by one or two bits (depends on the round). The two shifted words are processed by the Permutation Choice 2 (PC2) generating a 48-bit subkey.

The DES deciphering operation uses the same round structure. The only difference is that subkeys are used in reversed order, i.e., subkeys $sk^{(16)}$, $sk^{(15}$, ..., $sk^{(2}$, $sk^{(1)}$ are used in rounds 1, 2, ..., 15, 16, respectively. More details about DES cipher can be found in the FIPS-46 standard [7].

2.2.1.2 Triple DES

DES was proved to be vulnerable and feasible attacks were demonstrated in 90-ties. For this reason, the FIPS-46 standard was expanded with a so called Triple Data Encryption Algorithm (TDEA).

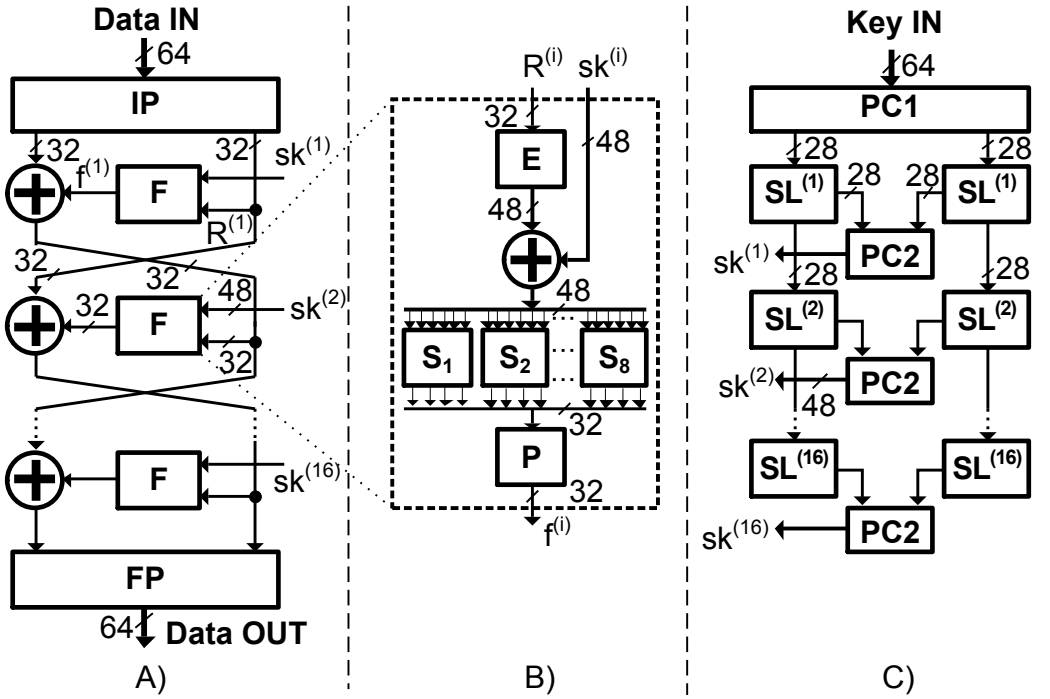

Figure 2.1: Structure of the Data Encryption Standard cipher (DES)

TDEA uses three DES ciphers connected in series. The ciphertext (CT) can be obtained from plaintext (PT) as follows: $CT = E_{K_3}(D_{K_2}(E_{K_1}(PT)))$. The decryption can be performed as follows: $PT = D_{K_1}(E_{K_2}(D_{K_3}(CT)))$. E represents a DES encryption operation while D represents a DES decryption operation. K_1, K_2 and K_3 represent three 56-bit secret keys which can be either independent (TDEA option 1), $K_1 = K_3$ are independent from K_2 (TDEA option 2), or all three keys are identical (TDEA option 3).

The TDEA option 1 is the strongest because the effective key length reaches 168 bits, followed by options 2 and 3. Unfortunately, TDEA operates with 48 rounds (three ciphers per 16 rounds) which can significantly decrease performance if a pipeline implementation is not possible.

2.2.1.3 Advanced Encryption Standard (AES)

Although the TDEA was not broken and was accepted as a standard in FIPS-46-3 [7], successful attacks on DES initiated a search for a new encryption standard. The Rijndael cipher was selected in a competition for Advanced Encryption Standard (AES) and standardized in 2001 as Federal Information Processing Standard 197 (FIPS-197) [8]. Nowadays, only AES is considered as sufficiently secure and recommended for data protection.

The structure of AES with a 128-bit key is depicted in Fig. 2.2 (for details on longer key versions see FIPS-197[8]). The ciphering is carried out in 11 rounds.

Data enters the cipher in 128-bit blocks. Each block is divided into 16 bytes (in_0, in_1, ..., in_{15}). The internal state of the cipher can be described by a matrix consisting of four rows and four columns as shown in Fig. 2.3. Four bytes of each column form a 32-bit word. The resulting data are organized 128-bit blocks.

The encryption algorithm uses four main operations that are applied on the state array: SubBytes, ShiftRows, MixColumns and AddRoundKey. The SubBytes transformation substitutes each byte in the state array with a different byte according to a substitution table (S-box tables can be found in the standard [8]). ShiftRows performs byte transposition of the state array as illustrated in Fig. 2.4. The MixColumns transformation operates on 32-bit words in columns (w_0, w_1, w_2, w_3). MixColums performs a modular multiplication of a state word, expressed in a polynomial form $s(x)$, with a constant polynomial $a(x) = \{03_{16}\}x^3+\{01_{16}\}x^2+\{01_{16}\}x+\{02_{16}\}$. More details on the modular multiplication can be found in [8]. The AddRoundKey operation is very simple, because it performs only bit-wise XOR operation between state bits and round key bits.

Unlike DES, the decryption algorithm in AES uses different operations: InvSubBytes, InvShiftRows, InvMixColumns and AddRoundKey. The InvSubBytes transformation substitutes each state byte with another byte according to a substitution table (InvS-box tables can be found in the standard [8]). InvShiftRows performs byte transposition of the state array as illustrated in Fig. 2.5. The InvMixColumns transformation operates on 32-bit words. It performs a modular multiplication of a state word polynomial $s(x)$ with a constant polynomial $a^{-1}(x) = \{0B_{16}\}x^3 + \{0D_{16}\}x^2 + \{09_{16}\}x + \{0E_{16}\}$. Another difference with the encryption algorithm is the necessity to precompute all round keys before the decryption can begin. During the decryption, round keys are used in the inverse order.

The byte substitution operations together with MixColumns provide very high confusion while ShiftRows together with MixColumns provide excellent diffusion.

The key expansion (see the part C in Fig. 2.2) generates 128-bit round key from the input secret key. This transformation is mainly based on modular addition (XOR). A non-linearity is introduced by the SubWord operation in Function (F). F operates on 32-bit words using left rotation on bytes, SubWord substitution and modular addition of an Rcon constant (see [8]).

2.2.1.4 Block Cipher Modes of Operation

Up to now, we have assumed only operations on one data block. However, when more data blocks have to be processed, so called block cipher modes of operation must be implemented. Block cipher modes use only a single key and some requires also an initializing vector (IV). If the last data block is smaller than the cipher block size, it must be extended by appropriate padding data. Block cipher modes of operation can provide confidentiality, authenticity or both.

Block cipher modes of operation were first standardized for DES cipher as FIPS-81 [9]. The standard included only ECB, CBC, OFB and CFB mode. After the standardization of AES in 2001, new standard SP800-38A [1] for block cipher modes

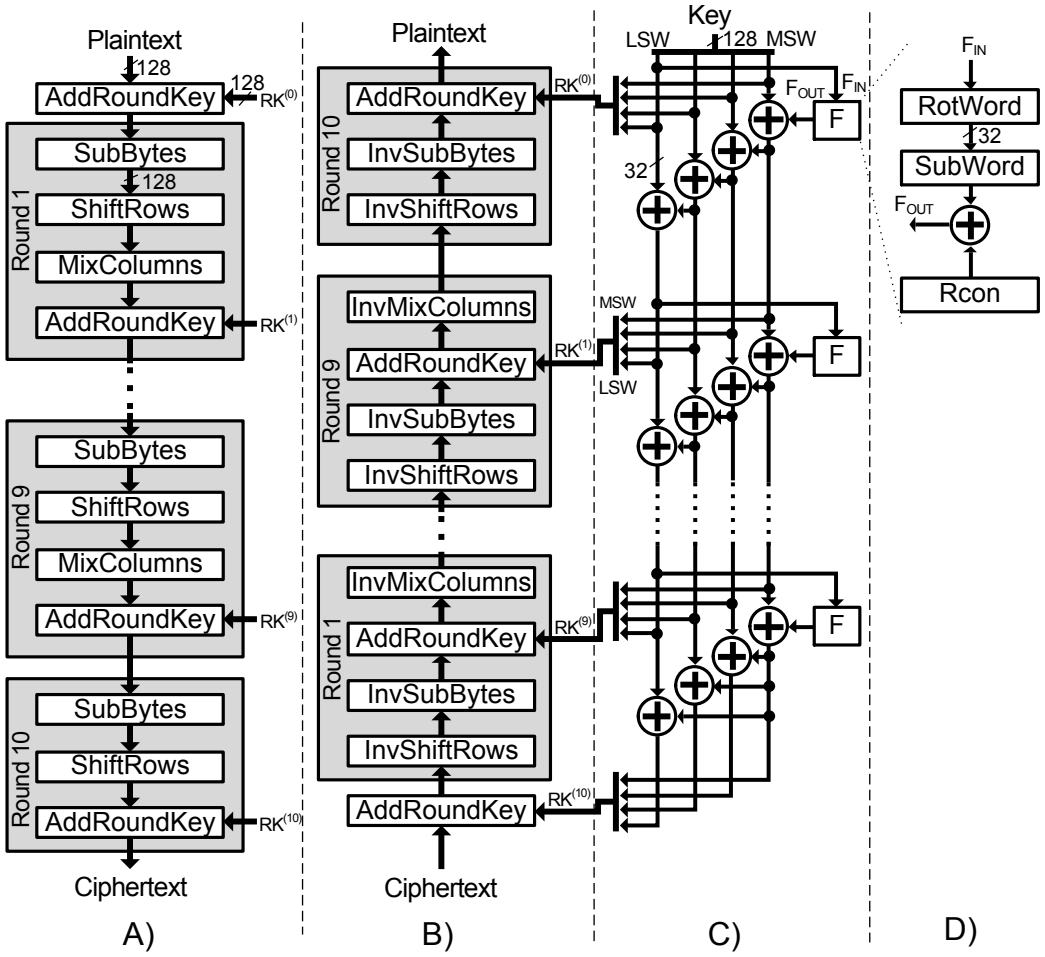

Figure 2.2: Structure of the Advanced Encryption Standard (AES) cipher with 128-bit key: Encryption (A), Decryption (B), Key expansion (C) with Function (D)

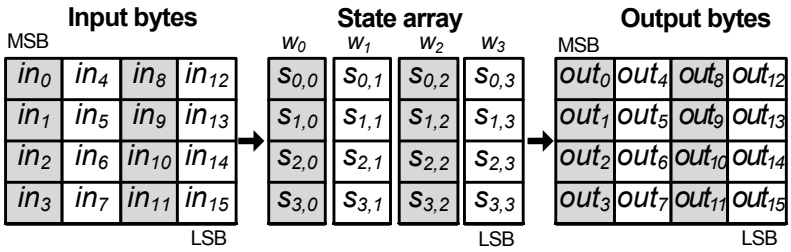

Figure 2.3: Division of input/output data blocks and internal state array

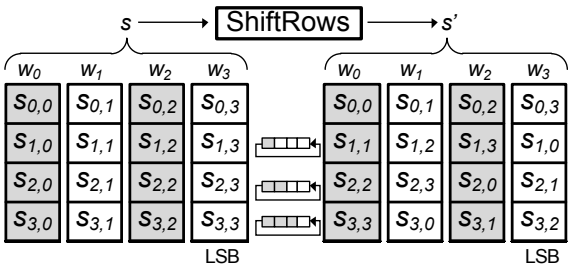

Figure 2.4: ShiftRows transformation performing left rotation of rows by one, two or three bytes

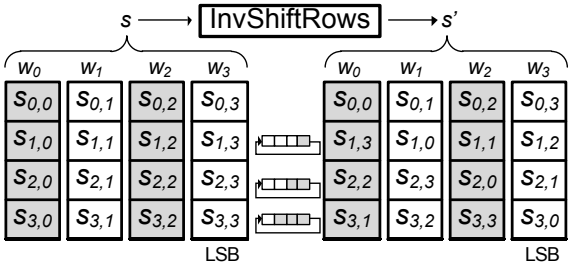

Figure 2.5: InvShiftRows transformation performing right rotation of rows by one, two or three bytes

was published. This standard contains the ECB, CBC, OFB, CFB and CTR mode.

However, the block cipher modes presented in SP800-38A [1] provide only confidentiality. A CMAC mode providing authenticity is described in SP800-38B [2].

Other modes providing both authenticity and confidentiality were standardized. The CCM mode is described in SP800-38C [10] and GCM in SP800-38D [11].

Next, we describe five most important basic block cipher modes operating on 128-bit blocks and providing confidentiality and one mode providing authenticity. We assume that the last data block is complete and does not require padding. For the sake of simplicity these block modes are presented using the AES cipher or decipher. These modes are also compared in Tab. 2.1.

Electronic Code Book (ECB) The most simple mode is the Electronic Code Book (ECB) block cipher mode (see Fig. 2.6) described in SP800-38A [1]. Encryption requires to apply the AES cipher on all data blocks using the same secret key. Decryption, on the other hand, uses the AES decipher with the same secret key.

The advantage of the ECB mode is its high speed because of the possibility to compute multiple blocks in parallel. Moreover, errors in one data block do not propagate to other ciphertext data blocks. The main disadvantage is the inability to hide data patterns. Especially this effect can be visible when enciphering images which contain bigger patterns with the same color. For this reason the ECB mode is recommended only for independent data blocks (e.g. key encryption).

Table 2.1: The comparison of basic block cipher modes of operation providing confidentiality [1] and authenticity [2]

	ECB	CBC	CFB	OFB	CTR	CMAC
Decipher is required	yes	yes	no	no	no	no
Error propagation	1[a,b]	all remain.[a], 2[b]	all remain.[a], 2[b]	1[a, b]	1[a, b]	MAC
Parallel execution	yes	no[a], yes[b]	no[a], yes[b]	no[a, b]	yes[a, b]	no
IV is required	no	yes	yes	yes	yes	no
Data pattern problem	yes	no	no	no	no	—

[a] Number of output blocks influenced by an error in an input block or IV in case of **encryption**.
[b] Number of output blocks influenced by an error in an input block or IV in case of **decryption**.

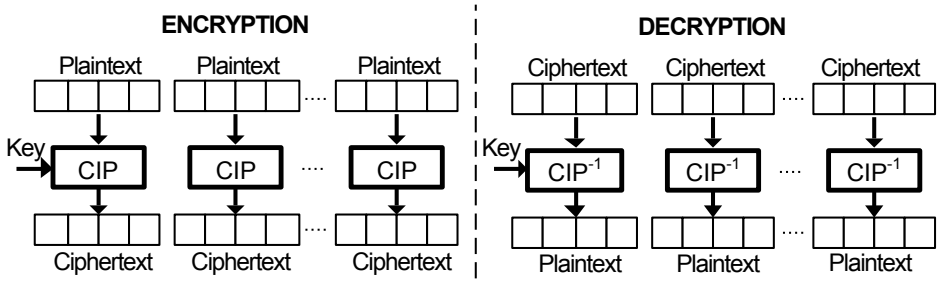

Figure 2.6: Electronic Code Book encryption block cipher mode of operation

Cipher Block Chaining (CBC) The Cipher Block Chaining (CBC) block cipher mode is illustrated in Fig. 2.7 and details can be found in SP800-38A [1]. Unlike the ECB mode, the CBC mode requires an Initialization Vector (IV). The ciphering starts by XOR-ing the first plaintext block with the IV. The result is encrypted by AES to form the first ciphertext block. The same ciphertext block is XOR-ed with the next plaintext block and the result is encrypted again to form another ciphertext block.

In case of decryption, the first ciphertext is decrypted by the AES decipher and XOR-ed with IV forming the first plaintext block. Next, the second ciphertext block is decrypted by the AES decipher and the result is XOR-ed with the previous ciphertext block forming the second plaintext block.

Interestingly, an error in the plaintext or IV influences all ciphertext blocks during encryption. If an error occurs in a ciphertext block or IV, this error influences only two consecutive blocks during decryption. Unlike the data pattern problem in ECB mode the CBC mode achieves indistinguishability of the ciphertext from the random bits.

Cipher Feedback (CFB) The Cipher FeedBack (CFB) block cipher mode is illustrated in Fig. 2.8 and details can be found in SP800-38A [1]. Similarly to the CBC mode, the CFB mode utilizes the IV. The IV is encrypted by AES and the

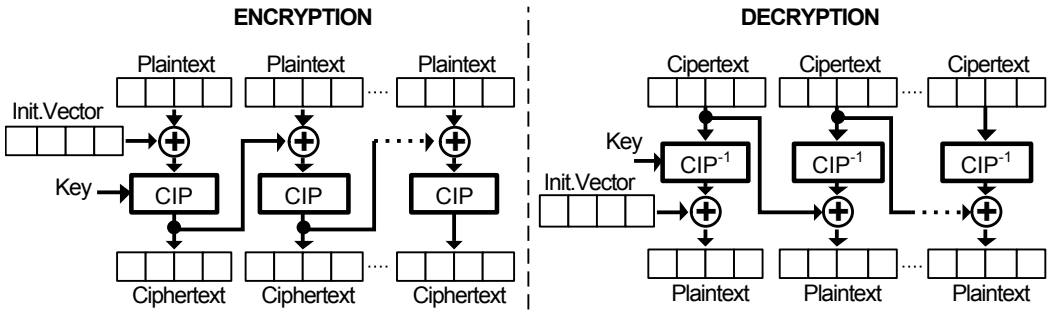

Figure 2.7: Cipher Block Chaining encryption block cipher mode of operation

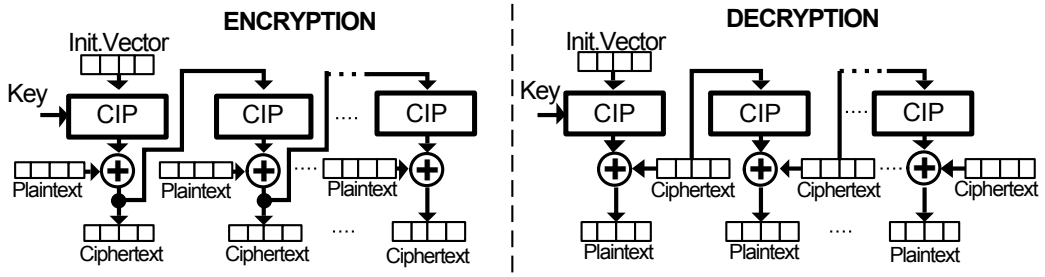

Figure 2.8: Cipher Feedback encryption block cipher mode of operation

result is XOR-ed with the first plaintext block to form the ciphertext. The resulting ciphertext block is encrypted by AES and the result is XOR-ed with the second plaintext block forming the second ciphertext block.

Unlike the ECB and CBC modes, the CFB mode uses AES cipher (not decipher) for decryption. This way the AES cipher can be shared by both encryption and decryption resulting in cheaper implementation. The IV is encrypted by AES and the result is XOR-ed with the first ciphertext block forming the fist plaintext block. The first ciphertext block is also encrypted by AES and the result is XOR-ed with the second ciphertext block forming the second plaintext block.

Similarly to CBC, an error in a plaintext block influences all remaining ciphertext blocks during encryption. On the contrary, an error in a ciphertext influences only the actual and subsequent plaintext blocks during decryption. Similarly to the CBC mode, the ciphertext output is indistinguishable from random bits. Although encryption is highly serial, decryption can be well parallelized resulting in much higher performance. An interesting property is a so called self-synchronization which enables to effectively synchronize decryption after a bit or byte error. However, this property holds only for CFB-8 and CFB-1 modes (working with 8-bit or 1-bit data blocks, respectively).

Output FeedBack (OFB) The Output FeedBack (OFB) block cipher modes is shown in Fig. 2.9 and details can be found in SP800-38A [1]. Unlike all previously described block cipher modes, the cipher in the OFB mode does not encrypt input

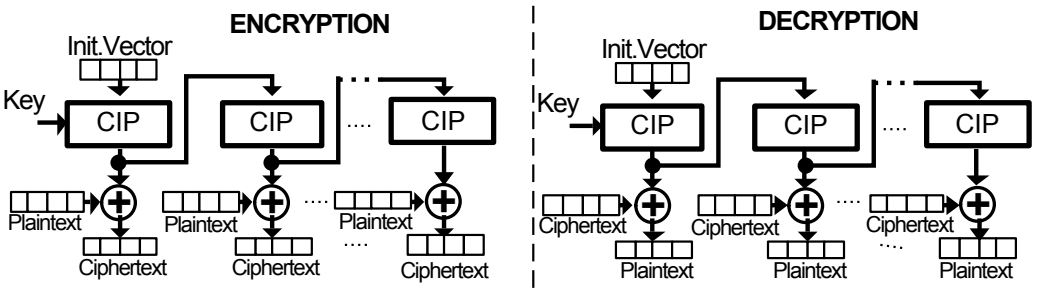

Figure 2.9: Output Feedback encryption block cipher mode of operation

data blocks but rather the IV and successive outputs of the cipher. The IV is first encrypted by AES and the resulting block is XOR-ed with the first plaintext block to form the first ciphertext block. The cipher output block is encrypted again and the result is XOR-ed with the second plaintext block to form the second ciphertext block.

Decryption is very similar. The generated cipher values are XOR-ed with ciphertext blocks forming plaintext blocks.

An error in an input data block propagates only to the resulting output block. However, if the IV is corrupted all output blocks are corrupted.

A parallel execution is not possible because the IV must be chained through all cipher blocks. However, if the IV is known soon enough prior to OFB execution and is not changed too often, the cipher outputs can be precomputed. These precomputed blocks only need to be XOR-ed with the corresponding plaintext or ciphertext. This XOR-ing operation can be performed in short time, and moreover, it can be parallelized.

Counter Mode (CTR) The Counter (CTR) block mode of operation is depicted in Fig. 2.10 and details can be found in SP800-38A [1]. Unlike the OFB mode, the CTR mode does not chain the ciphering but only encrypts the counter values. The counter must be initialized with the IV. The encryption starts by enciphering the IV by AES. The result is XOR-ed with the first plaintext block forming the first ciphertext block. Next, the IV is incremented and encrypted. The result is XOR-ed with the second plaintext block forming the second ciphertext block.

The decryption is almost the same as encryption. The only difference is that the AES output is XOR-ed with a ciphertext block forming a plaintext block.

The biggest advantage of the CTR over most block cipher modes is the possibility to precompute all cipher output blocks in a short time in parallel. These data can be simply XOR-ed with the plaintext or ciphertext in case of encryption or decryption, respectively. Because it is possible to parallelize both encryption and decryption the CTR mode achieves very high performance. If an error is introduced into an input plaintext or ciphertext block this error influences only one output block. However, if an error is introduced into the IV, all output blocks will be completely changed.

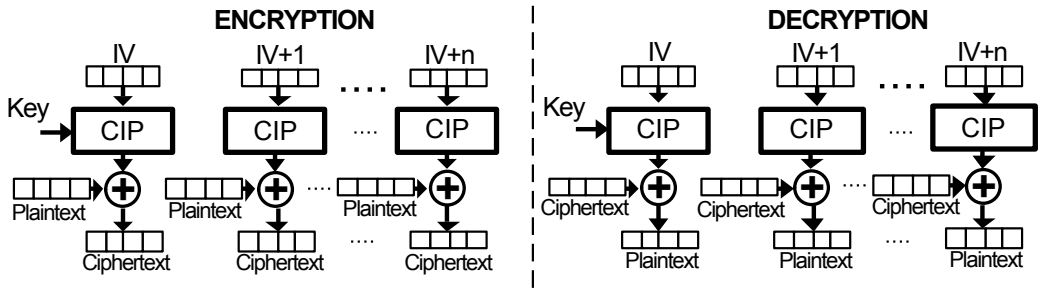

Figure 2.10: Counter encryption block cipher mode of operation

CBC Message Authentication Code (CMAC) Although all previously presented block cipher modes provide confidentiality, the verification of authenticity is not directly possible. The CBC Message Authentication Mode (CBC-MAC) can be used for authentication. A version of CBC-MAC mode called CMAC was standardized and published in SP800-38B [2]. The structure of CMAC is given in Fig. 2.11. For the sake of simplicity, the message is assumed to be divided into n complete 128-bit blocks and the resulting MAC is also a 128-bit value. For this reason, the structure presented in Fig. 2.11 is simplified for this particular case.

The first message block is encrypted by AES and the result is XOR-ed with the next message block. The result is encrypted again and so on. The only difference from the CBC-MAC is when processing the last message block. If we assume that the last (n-th) message block is complete, it must be XOR-ed not only with the previous encrypted data but also with a subkey K1 (see the part B in Fig. 2.11). The calculation of K1 involves a simple left shift and XOR-ing with a constant. The MAC is the result of the last encryption.

The most important property is the error propagation. If a 1-bit error is inserted into the message, all MAC bits are influenced. Moreover, length of the MAC word (i.e. 128 bits) ensures that it is almost impossible to find other message resulting in the same MAC. For this reason, CMAC perfectly guarantees authenticity of data and their owner (the holder of the key).

2.2.2 Stream Ciphers

Unlike the block ciphers, stream ciphers operate on individual bits. One interesting property of the stream ciphers is the zero error propagation. In other words, if a particular bit of the plaintext is corrupted, the error will influence only the corresponding ciphertext bit. For this reason, stream ciphers are used for infinitely long bit streams transported via a noisy channel. Stream cipher can achieve very high encryption speed. A disadvantage of stream ciphers is the small level of diffusion. The encryption of the plaintext bits involves a very simple operation (i.e. exclusive or). This transformation uses one plaintext bit and one bit of the keystream as its inputs and produces one ciphertext bit. The keystream can be random or it can be a pseudorandom sequence generated by a special key-dependent function. Since the

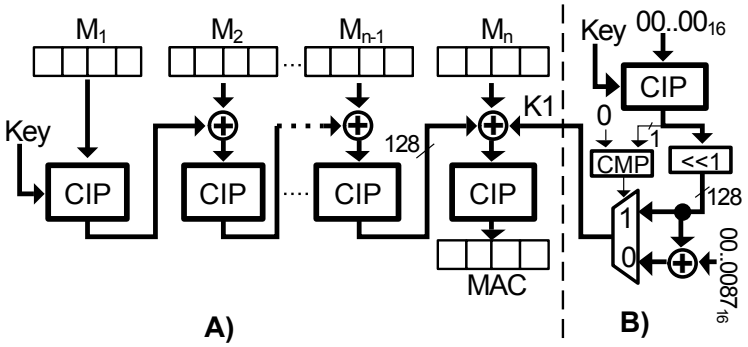

Figure 2.11: CMAC block cipher mode for authentication (part A) and a detailed computation of the K1 (part B)

keystream must not involve the secret key directly, keystream generation must start using an initial small keystream (called initialization vector or seed).

An interesting example of a stream cipher is the Vernam cipher which uses the exclusive-or operation as an encryption function. If the keystream sequence is random and it is never repeated (its length is the same as plaintext) it is called a one-time pad. A unique feature of the one-time pad cipher is its unbreakableness. The biggest disadvantage of the one-time pad is the key exchange which can involve physical transportation of the keystream material.

2.3 Hash Functions

A MAC can be generated using block ciphers but also using hash functions. A cryptographic hash function is a one-way function that processes a message and produces a hash (sometimes called message digest or digital fingerprint). Cryptographic hash functions must fulfill the following criteria as defined in [3]:

- **Preimage resistance** — for all pre-speficied outputs, it is computationally infeasible to find any input which hashes to that output
- **2nd-preimage resistance** — it is computationally infeasible to find any second input which has the same output as any specific input
- **collision resistance** — it is computationally infeasible to find any two distinct inputs with the same hash

Many cryptographic hash functions exist (i.e. MD4 [12], MD5 [13], SHA-1 [14], SHA-2 [15], etc.) and more details can be found in [3]. Important to mention is the Keccak algorithm – the winner of the SHA-3 competition organized by NSA [16]. For better understanding of our work we briefly introduce the MD5 cryptographic hash function.

2.3.1 MD5 Hash Function

The MD5 hash function is based on the Merkle-Damgård construction. The function operates with 512-bit message blocks and produces a 128-bit digital fingerprint. MD5 uses padding data P so that the message length L is divisible by 512. Concatenation ($||$ symbol) of a padding value P to a message M results in a complete message $M^{'}$:

$$\{M^{'}\} = \{M\}||\{P\} = \{M\}||\{100...00_2\}||\{L\} \tag{2.1}$$

The constant $100...00_2$ has as many zero bits as required to fill the last message block to 448 bits. The last 64 bits are reserved for message length information L.

The structure of the MD5 hash function is illustrated in the part A in Fig. 2.12. The MD5 computation is performed in four rounds and each round consists of 16 steps. A 128-bit hash state h_i is divided into four 32-bit words denoted as A, B, C and D. The initial value h_0 is an initialization vector which is defined as:

$$h_0 = 67452301EFCDAB8998BADCFE10325476_{16} \tag{2.2}$$

After entering the round block, four 32-bit words are processed in 16 steps. One such step is shown in the part B in Fig. 2.12. Depending on the round, one of the functions is applied on the B, C and D words. Functions use XOR ($\oplus$), AND (.), OR (+), and negation ($\bar{B}$) binary operators. Subsequently, a $\boxplus$ block performs the addition modulo 2^{32} operation. The X[k] is k-th 32-bit word of the processed message block M_i. The K[i] input represents a i-th 32-bit constant in table T containing 64 such words. The RL_s block performs a left rotation by s bits. The resulting 32-bit words are returned back to registers to start another step. When all 16 steps are completed new round is started. After completing all rounds, the four resulting words are added modulo 2^{32} to the previous four words of h_i to form h_{i+1}. If the message is finished the resulting 128-bit word represents a calculated hash value. However, if all message blocks have not been processed yet, the whole MD5 process repeats for the next 512-bit message block. More details on ordering of X[k] and K[i], the T table, and the s parameter can be find in the MD5 standard [13].

2.4 Generation of Random Numbers

Random number generators (RNG) are cryptographic primitives in cryptographic systems used for generation of ephemeral keys, initialization vectors, padding values, random masks for countermeasures, etc. RNG must meet certain basic requirements:

- To give unpredictable output data
- To guarantee good statistical properties of output data
- To be inherently secure, robust and resistant to attacks
- If possible, to test generated values on line by generator specific tests

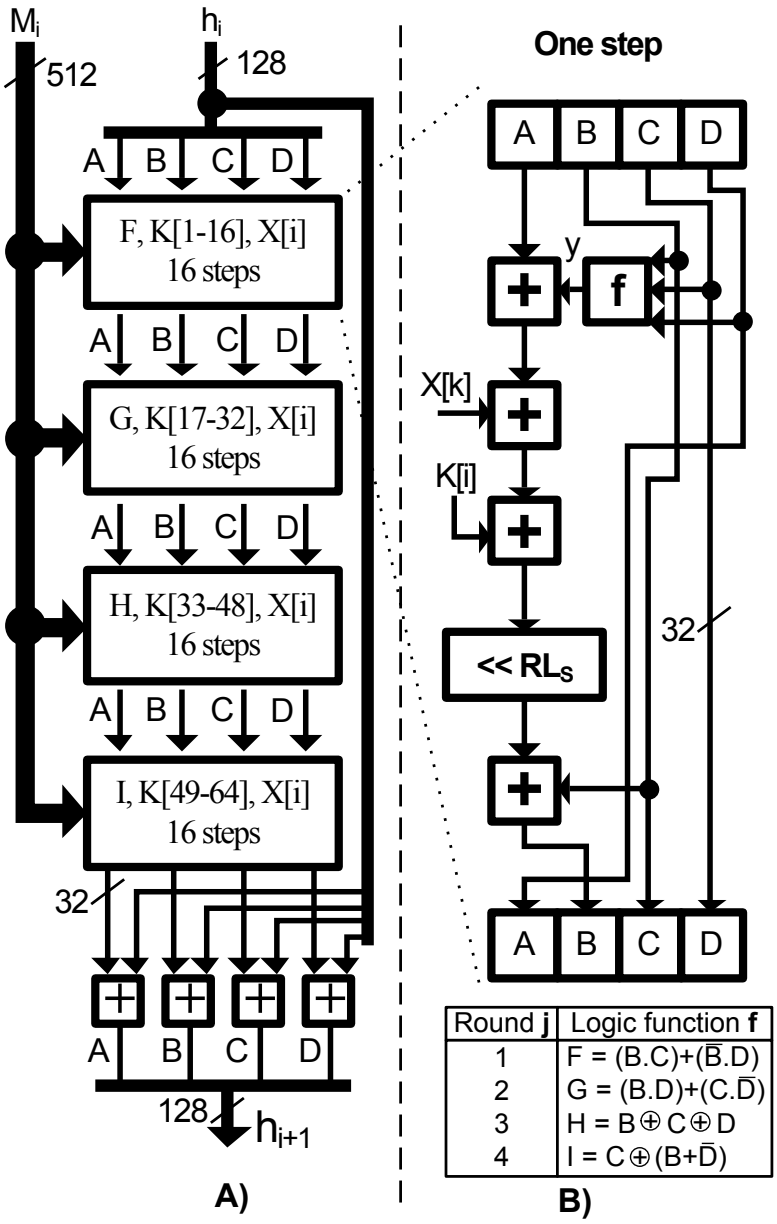

Figure 2.12: MD5 hash function structure (part A) and architecture of one step (part B)

2.4.1 Hardware Random Number Generators

Three basic RNG types are used in cryptography:

A Deterministic (pseudo-) random number generators (DRNGs or PRNGs)

B Physical (true-) random number generators (TRNGs)

C Hybrid random number generator (HRNGs)

PRNGs are very common because of their relatively straightforward structure and high performance. For example, PRNGs can be used for generation of a keystream used in stream ciphers. However, PRNG output is generated using exact mathematical formula, and if the generated sequence is not long enough, or the attacker has knowledge about the DRNG structure, attacks are feasible. Especially if the sequence is not long enough, it can be recorded and replayed afterwards.

TRNGs exploit analog physical processes as source of randomness. This physical process must be uncontrollable so that attacker cannot control the generation of output values, which are thus unpredictable. In general, TRNGs achieve lower performance than DRNGs. Their performance is influenced by the spectrum of the noise source and by the randomness extraction principle. The statistical characteristics depend on the noise source and the method used for extraction of randomness. However, the fluctuations of physical processes may affect statistical characteristics negatively. For this reason, TRNG output data may be cryptographically post-processed to maintain good statistical characteristics. Most common sources of randomness in logic devices are clock period jitter, setup & hold time violations, thermal noise, true dual port RAM write collision and others. More details can be found in [17].

HRNGs are a combination of the TRNG and PRNG. The TRNG repeatedly generates an unpredictable seed that initializes PRNG. PRNG generates random number at high speed, while TRNG prepares a new seed. HRNGs represent a tradeoffs between the predictability and speed.

Many different statistical tests for PRNGs exist and some of them were also adapted for TRNGs. Early in the testing phase, designers prefer to test TRNGs using FIPS-140-1 [18] or FIPS-140-2 [19] statistical tests. The small size of testing samples is an advantage of these tests. However, these tests are not sufficient and more complex tests are also necessary. Very common test suites are NIST SP800-22 [20] or DIEHARD [21].

The main disadvantage of the aforementioned statistical test suits is their inability to distinguish pseudorandomly generated data from truly random data. The response to this problem was the AIS-31 methodology that was proposed in 2001 and updated in 2011 [22], [23].

Many interesting RNG exist that can be implemented in logic devices. The most common TRNGs are based on ring oscillators [24, 25, 26]. The outputs of independent ring oscillators are usually sampled by a reference clock.

Sunar et al. proposed a TRNG composed of 114 ring oscillators (each includes 13 inverters) [27, 28]. Their outputs are XOR-ed and sampled by a D Flip-Flop (DFF). This principle was later improved by Wold and Tan by adding a DFF into output of each ring oscillator [29]. A first practical application of Sunar's TRNG was proposed by Schellekens et al. [30]. Other interesting TRNGs based on ring oscillators were proposed by Bucci et al. [24, 31], Kohlbrenner and Gaj [32], Tkacik [33] and many others. A very interesting TRNG called Transition Effect Ring Oscillator (TERO) was proposed by Varchola and Drutarovsky [34]. More detail on TRNGs and many other principles are comprehensively described in [17].

Next, we briefly present one PRNG and one TRNG that were used in this work.

2.4.2 LFSR-based PRNG

The Linear Feedback Shift Registers (LFSRs) are very simple PRNGs that can be easily implemented in hardware. Main advantages of LFSRs are the ability to produce sequences with large periods and very good statistical characteristics. LFSRs can be analyzed easily using algebraic techniques.

A LFSR consists of N memory elements, where each stores one bit. These elements are chained one after another. The second part of the LFSR is the feedback. A feedback part can be described by a linear equation which combines outputs of certain memory elements and produces a feedback value. The feedback value forms an input to the first memory element. If the LFSR feedback is described by a primitive polynomial, LFSR will generate a maximal sequence (every possible state except the zero state). Before the generation starts, all memory elements must be initialized. The initialization data is called a seed.

Two typical LFSR topologies exist: Fibonacci and Galois. In general, Fibonacci LFSRs requires slightly smaller chip area than Galois LFSRs. On the contrary, Galois LFSRs achieve higher speed because of smaller delays in the feedback chain. A 128-bit Galois LFSR is depicted in Fig. 2.13 and its feedback equation $y = f(x)$ is given in Eq. 2.3.

$$y = x^{128} + x^{99} + x^{59} + x^{31} + x^{9} + x^{7} + 1 \tag{2.3}$$

Notice that the position of every XOR element before a memory element is exactly indicated in Eq. 2.3 by the power of a corresponding polynomial term. For instance, an input to the 31st memory element, indicated as x^{31}, is calculated as XOR between the outputs of the 30th and 127th memory elements. More details on LFSRs can be found in [35].

2.4.3 PLL-based TRNG

Phase-Locked Loops (PLLs) are common hardware primitives used for generation of chip clock signals. Fischer and Drutarovsky have proposed to use the jitter of the clock signal generated by the PLL as a source of randomness [36] (see the part A in Fig. 2.14). This randomness can be extracted by a DFF which samples the PLL

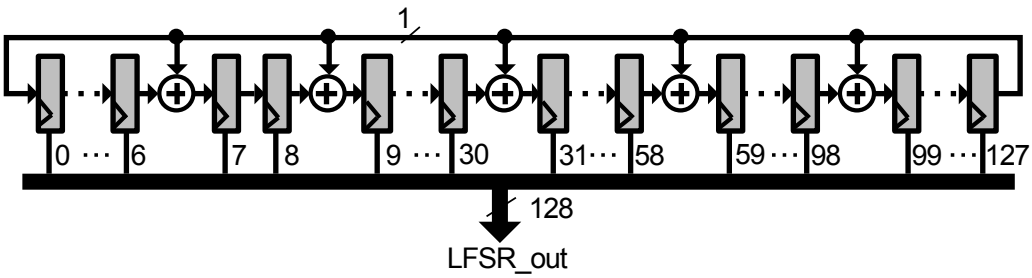

Figure 2.13: A 128-bit Galois Linear Feedback Shift Register capable of generating maximal length sequences

output clock signal clj on each rising edge of the PLL input clock signal clk. The relationship between the two clock signals can be described by Eq. 2.4 where K_M and K_D represent PLL multiplication and division factors, respectively.

$$F_{OUT} = F_{IN}\frac{K_M}{K_D} \tag{2.4}$$

The sampling of the clj signal by clk generates a patter in the DFF output with period T_Q is

$$T_Q = K_D T_{clk} = K_M T_{clj}. \tag{2.5}$$

This patters contains a random bit and its extraction requires XOR-ing all bits of the pattern by the Decimator. During the period T_Q, K_D rising edges of the clk signal occur, resulting in K_D bits in the pattern. One random bit can be present in the pattern only if the following condition is fulfilled:

$$\sigma_{jit} = MAX(\Delta T_{min}) = \frac{T_{clk}}{4K_M}GCD(2K_M, K_D) = \frac{T_{clj}}{4K_D}GCD(2K_M, K_D) \tag{2.6}$$

The GCD represents Greatest Common Divisor. In other words, the generator can be more sensitive to jitter if K_D is odd and as high as possible while the period T_{clj} is as small as possible.

The PLL-based generator can be significantly improved if two PLLs are used [37] (see the part B in Fig. 2.14). The PLL1 generates the clj_{rng} clock signal while the PLL2 generates clk_{rng} signal. User clock signal (clk_{usr}) can be also extracted from PLL2. More details on both PLL-based TRNGs can be found in [36, 37, 17].

An example of a configuration of a TRNG based on two PLLs for Xilinx Virtex-6 FPGAs is summarized in Tab. 2.2.

2.5 Key Management

One of the most important issues in secure communications is the management of confidential keys. Key management involves techniques, protocols and schemes for establishment and use of necessary keying material between authorized communication parties. Key management may involve, but are not limited to, the following tasks:

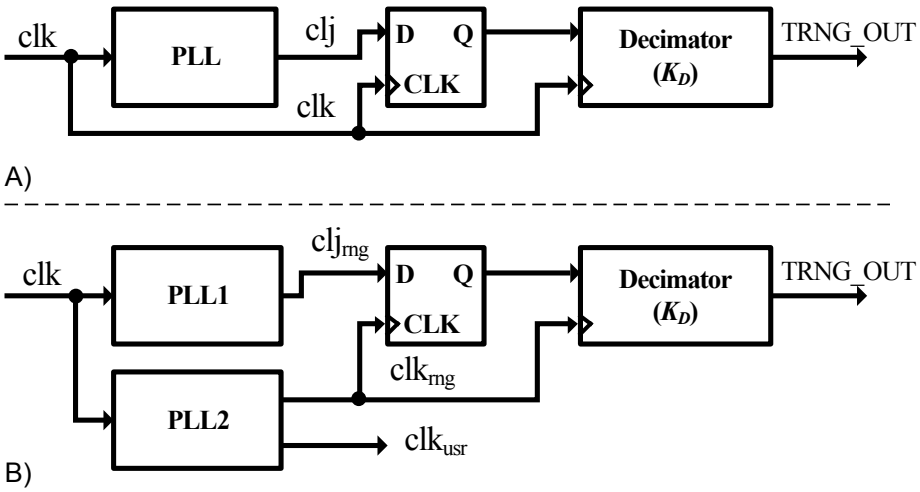

Figure 2.14: TRNG based on one (part A) or two PLLs (part B)

Table 2.2: Configuration of a TRNG based on two PLLs for Xilinx Virtex-6 FPGAs

	Name	F_{IN}[MHz]	K_M	Divider	K_D	F_{VCO}[MHz]	Output div.	F_{OUT}[MHz]
PLL1	clj_{rng}	200	55	7	—	1571.318	1	1571.318
PLL2	clk_{rng}	200	31	7	**217**	885.712	4.5	196.825
	clk_{usr}						9	98.412

1. **Pre-initialization** of long-term confidential keys, may involve manual distribution by a trusted entity
2. **Generation** of ephemeral (short-term) confidential keys
3. **Distribution** of ephemeral keys among all authorized communication parties
4. **Installation** of ephemeral keys involving their authentication in all authorized parties
5. **Updates or destruction** of ephemeral keys in case of their expiration or presence of security risks
6. **Storage** of both long-term and ephemeral keys in a secure isolated environment
7. **Use** of confidential keys for protections of other keys or data

Secure key management is a key management that is robust against various types of attacks (i.e. software attacks, protocol attacks, etc.).

Next, we explain key management security requirements as proposed by FIPS-140-2 [19], summarize assumed participating parties and the threat model, and propose a protocol ensuring secure key management.

2.5.1 Security Levels and Key Management

Key management is an essential element of every cryptographic module. Thus, security requirements proposed in FIPS-140-2 [19] for cryptographic modules have direct impact on key management and its security. Next, the four security levels as defined in FIPS-140-2 [19] are described. Note that security levels 3 and 4 require that secret keys are exchanged via a secure channel separated from data channels. If it is not possible, secret keys must be encrypted when exchanged to fulfill these criteria. Our aim is to propose such key management techniques where secret keys are encrypted when exchanged, and are isolated from data when stored or threated in clear. This way, security levels 3 and 4 can be reached.

Security level 1: provides the lowest level of security. Basic security requirements are specified for a cryptographic module (e.g., at least one Approved algorithm or Approved security function shall be used). No specific physical security mechanisms are required in a Security Level 1 cryptographic module beyond the basic requirement for production-grade components. An example of a Security Level 1 cryptographic module is a personal computer encryption board.

Security Level 1 allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an unevaluated operating system. Such implementations may be appropriate for some low-level security applications when other controls, such as physical security, network security, and administrative procedures are limited or nonexistent. The implementation of cryptographic software may be more cost-effective than corresponding hardware-based mechanisms, enabling organizations to select from alternative cryptographic solutions to meet lower-level security requirements.

Security level 2: Security Level 2 enhances the physical security mechanisms of a Security Level 1 cryptographic module by adding the requirement for tamper-evidence, which includes the use of tamper-evident coatings or seals or for pick-resistant locks on removable covers or doors of the module. Tamper-evident coatings or seals are placed on a cryptographic module so that the coating or seal must be broken to attain physical access to the plaintext cryptographic keys and critical security parameters (CSPs) within the module. Tamper-evident seals or pick-resistant locks are placed on covers or doors to protect against unauthorized physical access.

Security Level 2 requires, at a minimum, role-based authentication in which a cryptographic module authenticates the authorization of an operator to assume a specific role and perform a corresponding set of services.

Security Level 2 allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an operating system that

- meets the functional requirements specified in the Common Criteria (CC) Protection Profiles (PPs) listed in Annex B and

- is evaluated at the CC evaluation assurance level EAL2 (or higher).

An equivalent evaluated trusted operating system may be used. A trusted operating system provides a level of trust so that cryptographic modules executing on general purpose computing platforms are comparable to cryptographic modules implemented using dedicated hardware systems.

Security level 3: In addition to the tamper-evident physical security mechanisms required at Security Level 2, Security Level 3 attempts to prevent the intruder from gaining access to CSPs held within the cryptographic module. Physical security mechanisms required at Security Level 3 are intended to have a high probability of detecting and responding to attempts at physical access, use or modification of the cryptographic module. The physical security mechanisms may include the use of strong enclosures and tamper detection/response circuitry that zeroizes all plaintext CSPs when the removable covers/doors of the cryptographic module are opened.

Security Level 3 requires identity-based authentication mechanisms, enhancing the security provided by the role-based authentication mechanisms specified for Security Level 2. A cryptographic module authenticates the identity of an operator and verifies that the identified operator is authorized to assume a specific role and perform a corresponding set of services.

Security Level 3 requires the entry or output of plaintext CSPs (including the entry or output of plaintext CSPs using split knowledge procedures) be performed using ports that are physically separated from other ports, or interfaces that are logically separated using a trusted path from other interfaces. Plaintext CSPs may be entered into or output from the cryptographic module in encrypted form (in which case they may travel through enclosing or intervening systems).

Security Level 3 allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an operating system that

- meets the functional requirements specified in the PPs listed in Annex B with the additional functional requirement of a Trusted Path (FTP_TRP.1) and
- is evaluated at the CC evaluation assurance level EAL3 (or higher) with the additional assurance requirement of an Informal Target of Evaluation (TOE) Security Policy Model (ADV_SPM.1).

An equivalent evaluated trusted operating system may be used. The implementation of a trusted path protects plaintext CSPs and the software and firmware components of the cryptographic module from other untrusted software or firmware that may be executing on the system.

Security level 4: Security Level 4 provides the highest level of security defined in this standard. At this security level, the physical security mechanisms provide a complete envelope of protection around the cryptographic module with the intent

of detecting and responding to all unauthorized attempts at physical access. Penetration of the cryptographic module enclosure from any direction has a very high probability of being detected, resulting in the immediate zeroization of all plaintext CSPs. Security Level 4 cryptographic modules are useful for operation in physically unprotected environments.

Security Level 4 also protects a cryptographic module against a security compromise due to environmental conditions or fluctuations outside of the module's normal operating ranges for voltage and temperature. Intentional excursions beyond the normal operating ranges may be used by an attacker to thwart a cryptographic module's defenses. A cryptographic module is required to either include special environmental protection features designed to detect fluctuations and zeroize CSPs, or to undergo rigorous environmental failure testing to provide a reasonable assurance that the module will not be affected by fluctuations outside of the normal operating range in a manner that can compromise the security of the module.

Security Level 4 allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an operating system that

- meets the functional requirements specified for Security Level 3 and
- is evaluated at the CC evaluation assurance level EAL4 (or higher).

An equivalent evaluated trusted operating system may be used.

2.5.2 Participating Parties

For the sake of simplicity, we assume the following participating parties:

Trusted Entity represents an authority which generates and manually distributes long-term confidential keys or other necessary secret in a secure way to all participating parties.

Communication partners generate ephemeral keys and exchange them securely (using long-term keys). During the exchange, both confidentiality and authenticity of ephemeral keys must be guaranteed. Communication partners use ephemeral keys to protect data during their exchange.

Adversaries try to gain access to long-term or ephemeral keys. The attacks can be performed either locally or remotely.

2.5.3 Threat Model

Local attacks may include, but are not limited to, physical, side-channel and fault-injection attacks. On the contrary, software or protocol attacks may be carried out remotely.

Adversary that is passively observing the communication between communication partners is called eavesdropper. Key establishment protocols ensuring confidentiality of the transported keying material is a sufficient countermeasure against eavesdropping. More dangerous is the Man In the Middle (MIM) active attack, where the attacker interrupts the communication imperceptedly and pretends to be one of the communication partners to others. To protect against this threat, key establishment protocols must provide confidentiality and authenticity of exchanged keys.

Software attacks can be potentially dangerous, because they allow to remotely modify execution of software that operates with confidential data. Software attacks and protocol attacks including MIM are assumed in all systems presented in this work.

Side-channel attacks exploit data leaked from physical devices during execution of cryptographic algorithms. This leaked data can be used to recover confidential keys. This threat is assumed and countermeasures are presented in Chap. 5.

Bitstream attacks try to reverse-engineer, tamper or replace configuration data of programmable gate arrays. Protection of configuration data are assumed in all presented systems. Special attention is given to attacks on so called partial bitstreams that are assumed in Chap. 6.

2.5.4 Key Establishment Protocols

Before the secure data exchange can be initiated, all communication counterparts must agree upon and exchange keys securely. The solution to this key distribution problem are key establishment protocols. Key establishment protocols can be based on symmetric cryptography algorithms, asymmetric cryptography algorithms or their combination.

Key establishment may involve pre-initialization of a shared secret, key agreement or key transport, and key updates. Key pre-initialization requires manual distribution of a necessary shared secret to all parties. Afterwards, key transport or key agreement protocols can be used to establish an ephemeral key secret (i.e. session key). During key transport, one party creates a new session key and sends it to other parties. On the contrary, during key agreement a shared secret is derived by the parties as a function of information contributed by each, such that no party can predetermine the resulting value of the key [3]. To provide some degree of assurance regarding the identity of communication parties, an authenticated key establishment protocol can be used.

Basic key establishment protocols and their properties are summarized in Tab. 2.3.

Key establishment protocols based on symmetric cryptography algorithms require at least the following key hierarchy:

1. *Master keys* are the highest-level secret keys. They are pre-initialized in the system by a trusted entity. Master keys are transported and stored in clear and so their isolation and protection is of utmost importance.

Table 2.3: Basic key establishment protocols and their properties as classified in [3]

Type	Protocol (properties)
Key transport protocol based on symmetric encryption	Point-to-point key update (no server) Our protocol in Sec. 2.5.4.2 (no server, key authentication) Shamir's no key protocol (no server) Kerberos authentication protocol (server based) Needham-Shroeder shared-key protocol (server based) Otway-Rees (server based)
Key transport protocol based on asymmetric encryption	Basic PK encryption (1-pass) (no entity authentication) X.509 (2-pass) -timestamps (mutual entity authentication) X.509 (3-pass) -random (mutual entity authentication) Beller-Yacobi (4-pass) (mutual entity authentication) Beller-Yacobi (2-pass) (unilateral entity authentication)
Key agreement protocol based on symmetric encryption	Blom's symmetric key pre-distribution system
Key agreement protocol based on asymmetric encryption	Diffie-Hellman (entity authentication) ElGamal key agreement (key entity authentication) Station-to-Station (mutual entity authentication)

2. *Session keys* are ephemeral secret keys that are generated by a TRNG and are used for data protection. Session keys must be protected during their exchange by master keys. After their expiration, new session keys must be generated and exchanged to replace the expired ones.

2.5.4.1 Point-to-point Key Update Protocol

The Point-to-point key update protocol is a very simple key transport protocol that enables to exchange session keys SK. This protocol does not support key authentication. The protocol makes use of a pre-initialized master key MK and involves communication parties A and B.

A session key SK is generated by A, encrypted (E) using master key MK and sent to communication partner B. B posseses the same MK which is used for decryption of the received session key SK. E represents a symmetric-key cipher.

$$A \rightarrow B\text{:}\ \ E_{MK}(SK) \tag{2.7}$$

Although this protocol provides confidentiality, B cannot verify if the sender is really A or a man-in-the-middle adversary. For this reason, we propose a simple authenticated version of this protocol.

2.5.4.2 Authenticated Point-to-point Key Update Protocol

The man-in-the-middle attacks can be avoided only if the transported session key is also authenticated. Instead of only one pre-initialized master key, we propose two

master keys: encryption master key MK_{ENC} and authentication master key MK_{AUT}.

A session key SK is generated by A. Subsequently, SK is encrypted by MK_{AUT} forming a session key fingerprint FP_A. The session key together with its fingerprint are encrypted by MK_{ENC} and send to B. B decrypts the message using pre-initialized MK_{ENC} to obtain SK and FP_A. Next, B calculates the session key fingerprint FP_B by encrypting the SK with the pre-initialized MK_{AUT}. The authenticity of the SK is confirmed if the two fingerprints (FP_A and FP_B) are matching.

$$A \rightarrow B\text{:}\ \ E_{MK_{ENC}}\left(SK, E_{MK_{AUT}}(SK)\right) \tag{2.8}$$

To prevent reply attacks a timestamp or sequence number might be included in the encrypted portion.

2.6 Implementation of Cryptographic Hardware in FPGAs

2.6.1 Field-Programmable Gate Arrays (FPGAs)

FPGAs are more and more common not only for prototyping but also in customer devices. This trend can be observed especially in cryptography, where FPGAs constitute often a part of cryptographic modules, secure network routers, military devices and other devices requiring high security and hardware acceleration. FPGAs are suitable for cryptography for their high level of parallelism and flexibility.

When compared to Application Specific Integrated Circuits (ASICs), FPGAs are flexible, very cheap for small or middle production series and hardware updates are straightforward. On the other hand, logic designs implemented in FPGAs operate on much lower frequencies than those implemented in ASICs. FPGAs also consume more power.

In their nature, FPGAs are integrated circuits that are composed of a configurable logic fabric and wires interconnected with configurable switches. By configuring these FPGA primitives in a certain way, FPGA can perform hardware functions. The ability to perform any hardware function predestine FPGAs for rapid prototyping (hardware emulation and testing) of future ASIC circuits. The configuration data are transferred to FPGA as a sequence of configuration bits called FPGA bitstream.

In addition to configurable primitives, modern FPGAs contain additional embedded hardwired units. Usually, FPGAs contain the next hardwired blocks:

PLLs are essential parts in all digital circuits. PLLs may be used for multiplication/division of clock frequency, phase shifts, deskewing, jitter cancellation, generation of random numbers and others.

RAM blocks represent static memory blocks with sizes not exceeding several kbits (e.g. 36 kb). These memory blocks can be concatenated to form bigger

memory units. Recently, Dual Port RAMs (DPRAM) and True Dual Port RAMs (TDPRAMs) are embedded. DPRAMs include two interfaces, one for writing and one for reading. This allows simultaneous reads and writes of data stored in two different memory locations. TDPRAMs include two independent access ports to the same memory content. This way each port can read from or write to any location in the memory array. If the same memory location is accessed parallely it may result in collisions.

Digital Signal Processing (DSP) blocks are blocks usually implementing complex hardwired adders, subtracters and multipliers. DSP blocks can operate on much higher frequencies than equivalent implementations in logic.

Microprocessor usually requires significant amount of logic resources to be implemented in logic fabric. Thus, some FPGA devices may contain hardwired microprocessor (i.e. Cortex M1/M3, ARM7, PowerPC, etc.).

Configuration circuitry involves configuration controller and configuration interfaces including the JTAG interface. Moreover, symmetric block cipher and secure key storage may be embedded to ensure security of private configuration bitstreams.

Fast transceivers are special embedded blocks which may contain PLLs, serializers, deserializes, equalization circuits and other parts. Transceivers can be used to implement fast serial communication interfaces like gigabit ethernet, serial ATA, HDMI and others.

Other blocks e.g., PCI Express interface, Ethernet interface, Analog blocks, etc.

2.6.2 Development of Hardware Functions for FPGAs

With the introduction of computers to hardware design, traditional procedures were replaced by comfortable hardware design suites. Design starts by describing a hardware function using a specific hardware language. The most common are the following two languages:

VHDL language: VHDL stands for VHSIC Hardware Description Language, where VHSIC means Very-High-Speed Integrated Circuits. Interestingly, VHDL was originally developed by U.S. Department of Defense for simulation of complex digital systems.

Verilog HDL language: Verilog is an open standard which is based on C syntax. Verilog and VHDL have different syntax but very similar functionality. Only VHDL will be considered further on.

FPGA design flow

The FPGA design flow includes synthesis of the VHDL code to netlists, placement (allocation of FPGA logic fabric resources for the hardware design), routing (selection of FPGA interconnections) and generation of the FPGA programming files. Optionally, functionality of a hardware design can be simulated before being tested in an FPGA. Powerfull design tools are available from all FPGA vendors.

2.6.3 FPGA Classes and Families

Depending on how a configuration is stored in an FPGA, thee different FPGA classes can be distinguished: Flash-based, Antifuse-based or SRAM-based.

2.6.3.1 Flash-based FPGAs

In Flash-based FPGAs all configurable switches are constructed using a flash technology resulting in a non-volatile FPGA. As a consequence, such FPGAs can be configured many times and can retain their configuration even if they are disconnected from the power source. Moreover, flash technology allows to change the FPGA configuration when necessary. Flash-based FPGAs are predominantly produced by Microsemi (former Actel).

2.6.3.2 Antifuse-based FPGAs

Antifuse-based FPGAs contain one-time-programmable switches based on antifuse technology. Unlike Flash-based FPGAs, Antifuse-based FPGAs can be configured only once. On the other hand, these devices are fault tolerant and resistant to radiation, and so they are often used in air&space industry. These FPGAs are mostly produced by Microsemi.

2.6.3.3 SRAM-based FPGAs

Unlike previous two FPGA classes, SRAM-based FPGAs store configuration in a configuration RAM. However, RAM contents cannot be maintained if power is disconnected. For this reason, an external non-volatile memory is necessary. After each power-up, FPGA must load a configuration bitstream from an external configuration memory. SRAM-based FPGAs have the biggest FPGA market share and the most important produces are Xilinx and Altera.

Remark: To mitigate the disadvantage of SRAM-based FPGAs, some FPGAs may contain a configuration flash memory chip in the same package with the FPGA chip. After power-up, FPGA accesses the flash and loads the configuration from it. In fact, this solution only minimizes device costs and required space on the circuit board.

Considered FPGA Families

Further on we will consider only the following FPGA families: Microsemi Fusion [38], Microsemi SmartFusion [39], Altera Stratix-II [40], Xilinx Virtex-5 [41] and Xilinx Virtex-6 [42].

2.6.4 Technology Limits

The necessity to protect secret keying material stored within FPGA devices forces FPGA manufacturers to implement different protection techniques. One such technique proposed in TEMPEST specification [43] suggests to create red (containing sensitive plaintext data material, or secret keys in clear) and black (ciphertext, secret keys are encrypted) zones that should be implemented in two different chips. If they must be in the same package, there must by an air gap between the two chips to protect data in the red zone.

However, if a single chip FPGA is used for security applications, air gaps cannot be creates and tradeoffs must be searched. The first and the simplest idea is to separate red logic from the black logic in the FPGA. The logic blocks between the red and black zones must not be used, so that the logic is separated by an imaginary gaps formed from these unused blocks. This can be done in all aforementioned FPGAs using FPGA-specific design constraints.

Unfortunately, logic separation is necessary but not sufficient. In addition to logic separation, routing resources crossing from one zone to another must be limited to a certain extent and special routing macros must be used for crossing the borders. However, this is not possible in all FPGAs. Manual routing is not supported in Microsemi FPGAs. Altera Stratix-II supports manual routing to a certain extent. Xilinx Virtex-5 and -6 devices enable designers to select routing quite precisely. Moreover, set of constraints and flows is available to control routing automatically. To facilitate all this effort, Xilinx has proposed so called Isolation Design Flow.

2.6.5 Isolation Design Flow in Xilinx FPGAs

The Isolation Design Flow (a part of the Single Chip Crypto project) allows to control placement of logic, routing, insertion of trusted bus macros, external isolation of I/O bank, etc. The flow is supported by the Xilinx PlanAhead software. Note that a partial reconfiguration license is necessary for the isolation design flow in the PlanAhead tool.

First of all, VHDL design must be divided into security zones. Each zone must be synthesized separately forming separate netlists. All these netlists must be loaded into the PlanAhead tool. Each netlist is placed into a logic partition. The separation to partitions ensures that logic in each partition is synthesized separately. Next, a partition is placed into a physical block which encompasses logic and routing resources, BRAMs, DSPs, and other blocks. To isolate physical blocks and corresponding routing resources from each other, special constraints must be applied to each physical block. As a result, the logic is located inside the corresponding

physical blocks, routing resources are used only inside the physical blocks, and only authorized routes may cross the border via trusted bus macros.

The isolation fence between two neighboring zones is composed of unused CLB blocks. The fence must be at least one CLB wide.

The verification of all stringent isolation constraints can be carried out using Isolation Verification Tool (IVT). IVT was a result of cooperation between Xilinx and NSA. IVT can be used to verify if a fence between two neighboring physical blocks is wide enough, if unauthorized routing resources do not cross physical block border or isolation fence, if package pins corresponding to two different physical blocks are not adjacent or not in the same bank (powered from the same power pins), and many other constraints. If the verification process is successful, IVT tool generates a certificate. This certificate can be supplied with the device to certification authority as a proof.

More details on the Isolation Design flow and the Single Chip Crypto technology (SCC) are explained in [44] and [45].

2.7 Partial Hardware Reconfiguration and Security

In general, FPGA configuration modifications require to reconfigure the entire FPGA device. However, certain FPGA families support a so called partial reconfiguration which enables to modify only selected (reconfigurable) parts of the FPGA configuration and all other parts (static) remain intact. Moreover, partial reconfiguration can be performed while static parts are still operating (dynamic reconfiguration).

2.7.1 Partial Reconfiguration

The Partial Reconfiguration technology (PR) is supported by Xilinx Virtex-4, -5 and -6. Spartan-6 supports only difference-based partial reconfiguration, which allows only limited changes of the configured logic [46]. The partial reconfiguration flow is also supported by Altera Stratix V, Aria V and Cyclone V [47]. Since Altera design tools with support of partial reconfiguration are not available at the moment, we will further consider only Virtex-5 or Virtex-6 FPGAs.

2.7.1.1 Partial Reconfiguration Flow

The implementation of PR is similar to the isolation design flow. First, a VHDL design must be divided into blocks. Blocks that are intended to be reconfigurable must be synthesized separately. If more variants (*reconfigurable modules*) to the same reconfigurable block exist, each must be synthesized to its own netlist. These netlists are imported to the PlanAhead tool. All static logic netlists are placed into *static partitions*. All reconfigurable module netlists are placed to the same *reconfigurable partition*. Only one reconfigurable module can be active at a time.

Next, *physical blocks* are created and each partition is mapped to one of them. Physical blocks containing reconfigurable partitions are considered reconfigurable

and the other ones static. Very important are timing constraints. It is highly recommended to manually define timing constraints for signals and buses interconnecting static and reconfigurable partitions. When the design is prepared, it can be implemented to an FPGA. The design implementation phase involves a synthesis, logic optimizations, timing optimizations, placement and routing.

During placement phase, an additional *proxy logic* is placed into each reconfigurable partition. This logic is placed in such locations where signals or buses enter or leave the physical block containing a reconfigurable partition. The input and output of the reconfigurable partition formed by proxy logic are called *partition pins*. Partition pins are connected directly to *reconfiguration bus macros*, which interconnect reconfigurable physical blocks with static ones.

If no module is needed a black-box reconfigurable module can be generated and activated instead. Black-box reconfigurable modules contain only partition pins.

Finally, a complete FPGA bitstream and several partial bitstreams can be generated. Complete FPGA bitstream contains also one reconfigurable module (the one that was active). Each partial bitstream contains one reconfigurable module.

More information about partial reconfiguration flow can be found in [48].

2.7.1.2 Configuration of Partial Bitstreams

After power up, an FPGA loads a complete FPGA bitstream and activates the FPGA. Default reconfigurable module is a part of the complete bitstream, and so it is configured by default. When the active reconfigurable module is replaced, FPGA loads the partial bitstream to the reconfigurable partition and activates it.

Unlike complete bit files, partial bit files do not include the bitstream header.

Configuration of the partial bitstream can be controlled either via an external configuration interface (e.g. by an external microcontroller) or via an internal configuration interface (i.e. a static logic). External configuration interfaces are, e.g., the SelectMAP or JTAG interface. Internal configuration interface is called Internal Configuration Access Port (ICAP).

When the partial bitstream is configured using ICAP, a microcontroller or a state machine in the static logic is necessary to fetch a partial bitstream and control the configuration progress via ICAP.

2.7.2 Security Aspects of the Partial Reconfiguration

Although, encryption of complete bitstreams was introduced long time ago, partial bitstream encryption (using hardwired cipher) is supported only by the Virtex-6 family. However, a work-around exists for Virtex-5: an encrypted partial bitstream is transferred to the FPGA, decrypted by a cipher in static logic (not hardwired cipher) and configured as an unencrypted bitstream via ICAP.

When a partial bitstream is decrypted and authenticated using hardwired AES and HMAC units, the partial bitstream is activated only if authentication succeeds. However, if a partial bitstream is decrypted in the static logic and configured as

an unencrypted bitstream, authenticity and integrity of the partial bitstream is not guaranteed. After being configured, such corrupted bitstream may not only operate incorrectly but also modify static logic or even damage FPGA. For this reason, partial bitstream should also be authenticated in the static logic before entering ICAP. Other more risky approach is to access internal status registers after configuration and verify if the configured bitstream CRC passed.

Chapter 3

Crypto-processor with Secure Key Management

Contents

In this chapter, we explain why cryptographic software implementations must be protected at hardware level in order to be secure against software attacks. Then we present state-of-the-art hardware cryptographic architectures and explain their advantages and disadvantages. Subsequently, the novel separation rules, that enable cryptographic processors to support secure key management, are presented. These rules are demonstrated on the novel HCrypt crypto-processor.

3.1 Crypto-processors - State of the Art

Not so long time ago, information security was considered only by the military world. However, during the last 40 years we could witness slow integration of cryptography into our everyday life. Starting with the standardization of DES in 1976 [7], recommended for protection of classified US government documents, cryptography spread to telecommunications, avionics, banking and recently to Internet world. The increase of cryptographic processing forced developers to focus on a special hardware for cryptography. The typical example is the DES cipher, which was designed to be extremely efficient when implemented in hardware. With the introduction of first general-purpose processors (GPPs) and the rapid increase of their computational power, implementations of cryptographic algorithms in software became an affordable solution.

Nowadays, there are many cryptographic solutions, which differ in many aspects like throughput, flexibility, cost, security, time-to-market, etc. Not so long time ago, cryptographic algorithms were implemented either in software executed by hard GPPs (e.g. Intel Pentium), or in hardware implemented in ASICs. Implementations in hard GPPs are cheap and easy to update. In contrast, execution of complex cryptographic algorithms is very time-consuming therefore GPPs are not suitable for high-performance cryptographic applications. From the security point of view, GPPs are often vulnerable to hardware or software attacks. Although some monitoring software and firewalls can be used to increase security, GPPs are in general not suitable for security applications.

Designers have been searching for ways to overcome performance problems on GPPs while maintaining lower costs. The best solution is the addition of a coprocessor that will be responsible for the acceleration of high-performance cryptographic algorithms. However, coprocessors are not flexible enough, and when a cryptographic algorithm has to be updated the whole crypto-coprocessor has to be changed. Cryptographic coprocessors are usually not protected against any kind of attack, because their sole task is to provide the acceleration of computations. On the other hand, there have been commercial attempts to build a secure crypto-coprocessor called Trusted Platform Module (TPM) [49]. The TPM was designed to be resistant to various attacks and to provide root of trust, so that only authorized software could run on the GPP. However, security issues were discovered [50] and attacks were carried out [51] on the TPM.

In order to achieve higher performance, designers have implemented complex cryptographic algorithms in ASICs. ASICs are very attractive for the big product series, but the extremely high initial costs prevent their use for small series. Moreover, every detail of the design has to be carefully considered, in order to reduce the non-recurring engineering costs (i.e. one-time cost to research, develop and test the ASIC). Otherwise, high costs could prevent its introduction to the market. This increasing pressure initiated a search for alternative ways to implement cryptography. Hardware attacks are usually mitigated by using hardware countermeasures. However, when these countermeasures become obsolete updates are not possible.

In the last decade, cryptographic algorithms were more frequently implemented in FPGAs. FPGAs can be SRAM-based or flash-based. SRAM-based devices keep their configuration in a volatile configuration SRAM, so the device has to be configured after every power-up. In contrast, Flash-based FPGAs store their configuration in internal flash memory so device configuration does not have to be configured after power-up again. FPGAs are very suitable for many cryptographic algorithms [52], [53]. Because of their high parallelism, high-performance data security algorithms can be significantly accelerated when compared to software implementations. Moreover, most FPGAs can be reprogrammed therefore hardware updates are cheap and easy to perform in place or even remotely [54]. Furthermore, FPGA manufacturers have recently introduced partial dynamic reconfiguration features in high-performance FPGA families [55], [48]. These features allow to update only a part of the circuit remotely while the rest is operating without interruption. More aspects of the partial reconfiguration security will be discussed in Chap. 6.

With the spread of FPGAs, many new interesting architectures have been introduced in the data security field. These architectures either reused old strategies developed for GPPs or ASICs, or introduced completely new revolutionary designs exploiting all the FPGA features (i.e. reconfigurability, parallelism, hardwired DSP blocks, several clock domains, etc.). Huge number of fine-grain resources and embedded memory blocks enable an implementation of soft GPPs inside the FPGA. Consequently, cryptographic algorithms can be implemented in software and executed by a soft GPP (further just GPP). This allows designers to achieve very interesting cost-security-performance tradeoffs.

Next, we will present security drawbacks of cryptographic software implementations. Then, a classification of cryptographic hardware implementations will be presented and the security of the current crypto-processor architectures will be compared.

3.1.1 Security Issues of the Cryptographic Software Implementations

At the time when DES was introduction, the unprecedented strength of this cipher eliminated all the brute-force attack attempts, so the attackers were searching for other ways to get classified data. One such way was to attack the GPP software implementation by modifying the enciphering algorithm or replacing the secret key. In such case, just a simple swap of two consecutive instructions can lead to a transfer of the secret keys out of the processor. On the other hand, the instruction swap is relatively easy to detect. Better software attack was suggested by Kocher in 1996: "RAM cache hits can produce timing characteristics" [56]. In 2002 Page et al. carried out first studies about how to use the cache memory as a cryptanalytic side-channel [57]. In 2005 Percival et al. demonstrated a cache-timing attack on RSA [58]. In the same year a cache-timing attack on AES was demonstrated by Bernstein et al. [59]. This attack has been improved by Bangerter et al. [60], who has proclaimed that only 100 plain-text messages and 3 minutes of post-ciphering computation were

needed to recover the confidential key on Intel Pentium M 1.5GHz. Moreover, this attack is hard to detect because change of AES computation time is negligible.

As the number of attacks was rising, government organizations have asked for a better protection of the software itself. All these attacks led to numerous works in the countermeasure design research field. In 2006 Osvik et al. have proposed several countermeasure techniques [61]. Some other countermeasures have been proposed in [60], [59] and [62], however there are still certain drawbacks and tradeoffs that must be dealt with. Even though, no countermeasure could be completely resistant to cache-timing attacks. Each proposed countermeasure had a weakness, which can be exploited by an attacker to disclose the secret keys. Clearly, the protection of a critical software by another software, while both are running on the same GPP, is not a sufficiently secure solution. If a software has to be secure, it has to be protected at hardware level.

3.1.2 Cryptographic Hardware Architectures and their Security

Many architectural configurations can exist and the frontiers between them are not always clear. Considering aspects like the extent of modifications of the GPP architecture, hardware acceleration, capability to perform general-purpose tasks (independence), flexibility, size, etc. hardware cryptographic architectures can be classified as follows:

- Customized GPP
- Cryptographic processor (crypto-processor)
- Cryptographic coprocessor (crypto-coprocessor)
- Cryptographic array (crypto-array)

These four different architecture types are illustrated in Fig. 3.1. Next, we will explain differences between these approaches, compare their advantages and disadvantages and preset some work that has been done in this area. This chapter will concentrate mostly on cryptographic processors and Chap. 4 will be dedicated to cryptographic coprocessors. The cryptographic arrays are out of the scope of this work. More details on crypto-arrays can be found in [63].

3.1.2.1 Customized GPP

The typical implementation of the customized GPP is shown in Fig. 3.1 part a). Under customized GPPs one can understand a general-purpose processor that is modified in order to be capable of performing cryptographic tasks more efficiently. This GPP includes special functional units that are capable of executing dedicated cryptographic operations (i.e. AES substitution operation, Montgomery multiplication, etc.). Each cryptographic function is represented as a custom instruction. All custom instructions form a custom instruction set. Secret keys are stored in the data

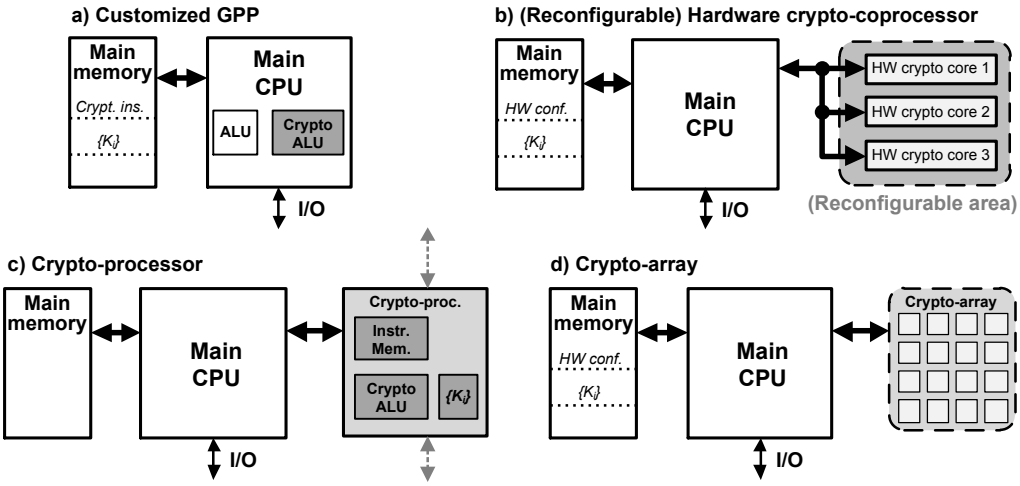

Figure 3.1: Four different types of the hardware cryptographic engines

memory and are treated as a regular data. For this reason, software attacks represent a considerable threat. This solution offers higher throughput than unmodified GPPs, because cryptographic operations are partly accelerated in hardware.

When designing a custom instruction set, the main problem is to find a set of elementary functions that could be used to implement different cryptographic algorithms (DES, AES, RSA, ECC, etc.). If cryptographic operations are broken to elementary ones in a too large extent (e.g. addition, binary operations, etc.), the performance will be comparable with the regular GPP's one. Therefore, the selection of a custom instruction set has to be carefully considered. Only then the resulting system can represent a well balanced software-hardware architecture capable of much higher performance than the pure software implementation. This fact has been demonstrated on a 32-bit Xtensa processor [64].

Several authors have proposed a customized GPP optimized for efficient AES computations [65], [66], [67], [68], [69]. Typical commercially available solutions are CryptoBlaze from Xilinx [70] or the AES New Instructions (AES-NI) incorporated in the new Intel processors [71]. Tab. 3.1 gives an overview of above mentioned customized GPPs. Tab. 3.2 gives their characteristics in more details.

A very interesting customized GPP is SecretBlaze proposed by Barthe et al. [72]. SecretBlaze is open-source and it is designed to be robust to side-channel attacks. Its flexible architecture allows to find tradeoffs between its size and security.

Unfortunately, no customized GPP architecture that would be resistant either to software or various hardware attacks was presented up to now. Secret keys are typically transferred unencrypted into the system together with regular data and the keys are stored unprotected in common data registers. Only a simple swap of two instructions can lead to their disclosure.

Table 3.1: Overview and performance of some GPPs customized for implementation of cryptographic algorithms

Name [ref]	Target technology	Cip. mode	Mbps/MHz	F_{max} in MHz	ASIC gates area
Custom Xtensa [64]	Xtensa 0.18 μm	AES-ECB	0.11	188	-
Sbox instruction [65]	Alpha 21264	AES-ECB	0.05	600	-
Inst. set ext. [66]	ASIC 0.13 μm	AES-ECB	0.57	250	16 KG 0.08 mm^2
CryptoBlaze [70]	CPLD CoolRunner	AES-ECB	?	?	-
Intel AES inst. [71]	Intel core 32 nm	AES-ECB	0.78	2670	-

Table 3.2: Summary of characteristics of customized GPPs

Name [ref]	Year publ.	Supported algorithms	Processing architecture	Key storage	Number of crypto cores	Hardware reconfig.	Main applic.
Custom Xtensa [64]	2002	DES, 3DES, AES, RSA	32-b Xtensa RISC cust. ALU	In main memory	1 crypto ALU	no	Wireless data security
Sbox instruction [65]	2000	3DES, IDEA, AES candid.	Sbox dedic. ALU	In main memory	1 Sbox look-up table	no	IPSEC, VPN
Inst. set ext. [66]	2005	AES	Cust. SPARC-V8 32b LEON-2	In main memory	1 Sbox, MixCols, ShiftRows unit	no	Embedded system data security
CryptoBlaze [70]	2003	AES RSA	Custom 8-b Xilinx PicoBlaze	In main memory	Sbox, GF multiplier unit	yes	FPGA system data security
Intel AES inst. [71]	2010	AES	Intel IA-32	In main memory	1 AES ALU	no	IPSEC VPN

3.1.2.2 Crypto-processor

The border between the crypto-coprocessor and crypto-processors is not always clear in the scientific publications [63] (see Fig. 3.1 parts b and c respectively). We consider the crypto-processor as a specific-purpose processor. This processor is programmable and contains one or more ALUs optimized for cryptography. Keys are stored inside the crypto-processor. It can also be connected to a GPP. On the other hand, the crypto-coprocessor contains one or several implementations of cryptographic functions. The crypto-coprocessor is not programmable but has to be controlled by a master GPP. Moreover, secret keys are not stored in the crypto-coprocessor, but rather in the GPP's register file.

Since crypto-processor is a special-purpose processor, its instruction set is usually more restricted than that of the GPP. For this reason it is not completely independent from the GPP and needs to be completed by a powerful GPP. Crypto-processor computational parts (ALUs) can be reconfigured in order to increase flexibility. Since a crypto-processor is more independent from the GPP than a crypto-coprocessor, more computations are moved from a GPP to a crypto-processor which leads to a better distribution of tasks in the multi-processor system. This task shar-

Table 3.3: Overview and performance of some crypto-processors

Name [ref]	Target technology	Cipher mode	Mbps/MHz	F_{max} in MHz	FPGA slices	FPGA RAM	ASIC gates area
CryptoManiac [74]	ASIC 0.25 μm	AES-ECB	1.42	360	-	-	1.93 mm^2
CCProc 1 core [76]	ASIC 0.13 μm	AES-ECB	1.62	250	-	-	93 KG 5.3 mm^2
CCProc 4 cores [77]	FPGA XCV4LX200	AES-ECB	6.40	95	18045	?	-
Cryptonite [78]	FPGA XCV2P30	AES-ECB	5.62	100	1748	32 KB	-
MCCP [79]	FPGA XCV4SX35	AES-CCM AES-GCM	4.43 9.91	192	8110	7 MB	-
HCrypt [80]	FPGA XCV6LX240T	AES-CFB	8.22	100	1422	148.5 KB	-

ing increases overall system throughput.

Many interesting contributions can be found in the crypto-processors research field: from simple processors to complex VLIW architectures. A crypto-processor aimed at construction of radio systems is presented in [73]. Several publications have been dedicated to complex VLIW (Very Long Instruction Word) crypto-processors. The crypto-processors capable of executing four 32-bit instructions in parallel are CryptoManiak [74], [75], or those developped under the CCProc Project [76], [77]. Other VLIW processor capable of executing two 64-bit instructions in parallel have been described in [78].

A very interesting concept was proposed by Grand et al. [79]. It is aimed at the software defined radio. This multi-core crypto-processor (MCCP) is capable of processing several communication channels at the same time. Unlike the previously mentioned architectures, MCCP contains a key memory for secure storage of secret keys. This memory can be initialized via a dedicated key channel. However, keys cannot be generated or exchanged securely.

Tab. 3.3 gives an overview of basic crypto-processor architectures. Tab. 3.4 gives their cryptographic characteristics in more details. Curiously, none of the mentioned architectures is able to provide secure key management. Secure key management includes the generation of truly random keys, their secure storage inside a dedicated key memory and their secure exchange. Our main objective is to search for such crypto-processor architectures that will support secure key management while maintaining high performance of cryptographic algorithms. We will further denote such crypto-processor as *secure crypto-processor*. In this chapter we present the novel HCrypt crypto-processor supporting secure key management that was published in [80], [63]. For the sake of completeness, our work is already included in Tab. 3.3 and 3.4.

3.2 New Rules for Securing Key Management

We have explained that in order to counter software attacks, secret keys must not be stored in clear in user data registers. For this reason, it is essential to physically

Table 3.4: Summary of characteristics of selected crypto-processors

Name [ref]	Year publ.	Supported algorithms	Processing architecture	Key storage	Number of crypto cores	Hardware reconfig.	Main applic.
CryptoManiac [74]	2001	AES, DES, 3DES	4-wide 4 Stage VLIW processor	Int. shared data memory	4 crypto ALUs by processor	no	IPSEC, VPN
CCProc [76]	2008	AES candid.	VLIW based processor	In main memory	4 Sbox clusters	no	Symm. encr. accelerator
Cryptonite [78]	2004	AES, DES, MD5	2*64-bit dedicated ALU	In main memory	Two dedicated ALUs	no	IPSEC, VPN
MCCP [79]	2011	AES, SHA, others	Multi-core processor	In dedic. register	2 - 8 reconf. crypto cores	yes	Software def. radio
HCrypt [80]	2010	AES ECB,... CBC-MAC,	2*128-bit dedic. ALU	In dedic. secure reg.	2 AES ciphers or deciphers	yes	Network secur., VPN

separate the place where the secret keys are stored from the place where user data are stored and processed. This separation must be achieved at several levels while the aspects of the system like security level, speed, size/cost and flexibility have to be considered. Next, we will present basic design rules increasing robustness of cryptographic processors against software attacks resulting in architecture that is secure by design. In order to fulfill the highest security requirements, the separation must be realized at four levels:

- protocol level
- system level
- architectural level
- physical level

We have presented the separation rules in [80], [81], [82] and [83]. Next, we will describe these levels and their principles in more details.

3.2.1 Separation at Protocol Level

High security of confidential keys can be achieved only with robust cryptographic protocols. Prior to data exchange, secret keys must be generated and securely exchanged between both communication counterparts. While being exchanged, their confidentiality and authenticity must be guaranteed. Indeed, if the keys were deciphered outside the security perimeter of the crypto-processor, they could be exposed to software attacks. Thus, secret keys have to be deciphered and authenticated in a protected part of the crypto-processor, and never leave this area in clear. These keys can be used for data enciphering/authentication but still inside the protected unit.

The protocol must clearly separate key management and data processing tasks. It must also determine how and by which blocks the tasks are performed. The separation of tasks can be provided at the instruction set level. Security critical tasks (i.e. key generation/transmission/reception, data enciphering/deciphering,

authentication, etc.) must be performed using dedicated instructions, while security uncritical tasks can be carried out by general-purpose instructions.

Finally, the key structure must be defined. If a hierarchical key structure is used, higher-level keys (i.e. master keys) are used to encipher and authenticate lower-level keys (i.e. session keys). The low-level keys are then used for data protection.

When considering side-channel attacks, the protocol must specify precisely the session key lifetime. If the session key is updated often enough, the attacker cannot gather enough information leakage to perform a successful attack. This simple approach can protect session keys very effectively.

3.2.2 Separation at System Level

The separation rules at system level are illustrated in Fig. 3.2. The rules suggest to create three zones: a data zone, a cipher zone and a key zone. The data are exchanged between the data zone and the cipher zone across the data bus (in black in Fig. 3.2). Encrypted session keys are also transported through this data bus when being exchanged with other communication counterparts. No way for accessing the key memory from the data zone must exist.

The data zone comprises data registers, ALU and blocks, which are responsible for performing block cipher modes and the packet management. No element from the data zone must have access to secret keys in clear.

Keys are stored in clear in a dedicated memory located in the key zone. The key memory has a hierarchical structure and is separated from the data zone by the cipher. All the keys are transferred between the cipher and the key memory via the key memory bus (in grey in Fig. 3.2), except for master keys that are introduced into the memory via separated input during device initialization. This bus must be completely separated from the data bus (in black) interconnecting the data zone with the cipher zone. It is essential that physical paths letting secret keys to pass in clear from the key memory bus to the data bus must not exist. This condition is very important from the security point of view, because it guarantees the separation of the key and data zones.

Before enciphering/deciphering data blocks and keys, the cipher is initialized with a selected key (a session key or higher level key) via the cipher key bus (in white in Fig. 3.2). The key selection is controlled by the processor's control unit through the control bus. The key address space must be completely covered – no unused key address can remain.

The principle of creation of security zones is independent from the type of enciphering algorithm – any symmetric key block cipher can be used.

3.2.3 Separation at Architectural Level

To achieve efficient separation of security zones, the data bus, the key memory bus and the cipher key bus cannot cross more than one security fence. Bus multiplexers directing the flow of data must be arranged in such a way that, even if their control

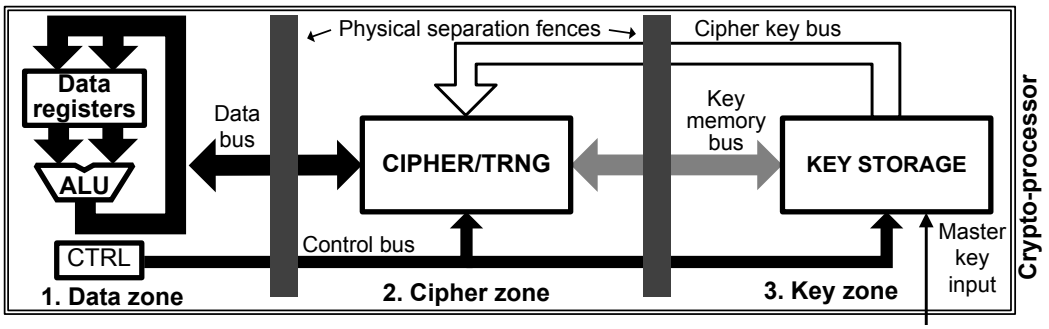

Figure 3.2: Separation of key storage and data storage at system level for crypto-processor

is violated, no physical path can be created for keys to escape from the key zone. Note that in order to send keys to the data output bus, there is no other solution than to send them via the cipher.

Interfaces between partitions must be designed to respect special constraints: data buses must be unidirectional and communications must be controlled only by the crypto-processor's control unit. A straightforward interface design simplifies implementation of bus macros in partial reconfiguration or physical isolation design flows.

3.2.4 Separation at Physical Level

For the highest security, separation at physical level has to be implemented at two levels:

- logic placement level
- routing level

3.2.4.1 Physical Separation at Logic Placement Level

In order to physically separate the security zones and to maintain this separation after the synthesis, each zone must be placed in a separate logic partition which is mapped to a separate physical block. These physical blocks must not overlap, otherwise this separation cannot be guaranteed after synthesis.

3.2.4.2 Physical Separation at Routing Level

Higher level of physical separation, and thus security, can be achieved if besides physical separation of functionnal blocks, the routing resources do not cross freely the boundaries of these blocks. For this reason, an empty area (insulation fence) is created at the border of the two neighboring zones (see Fig. 3.2) placed in two different physical blocks. Only selected signals can cross this fence. This counter-measure 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. data zone). This physical insulation principle is recommended by NSTISS in its Red/Black installation guidance [43]. This guideline is followed in Xilinx SCC design tools [44]. Moreover, the use of both separation at logic placement and routing level is suggested by NSA in [45].

3.3 Crypto-processor Design

It is clear that secret keys can be effectively protected within the device only if hardware countermeasures are implemented. Software countermeasures have not proved to be effective. In this chapter, we will explain the work concerning the design of the unique hardware architecture that can guarantee secure key management. The stringent separation rules are the corner-stone of the novel HCrypt crypto-processor concept. HCrypt is a specific-purpose processor optimized for execution of cryptographic algorithms that supports secure key management. However, the separation rules established for HCrypt can also be applied in any other crypto-processor design. Next, we will explain the structure of the unique HCrypt crypto-processor, discuss its security, present implementation results and finally test its operation. HCrypt was first presented in [80] and [63]. However, the presented version was designed only to prove the concept and was not optimized for performance. Further on, we will present only the optimized version which will be denoted as HCrypt.

When designing a specific-purpose cryptographic processor with secure key management, the objective is to physically separate registers and buses carrying keys from those carrying data. The proposed architecture must ensure that secret keys cannot leave processor unencrypted. For this reason, two sets of registers should be implemented, so that keys and data would be stored separately. In order to achieve higher performance, the processor has a 128-bit datapath.

HCrypt exchanges data with external/internal environment using input/output FIFOs. The processor is able to implement a serial communication protocol using packets. It permits to analyze and create packets efficiently. Data processing operations needed for implementation of cipher modes are carried out in a special-purpose ALU that constitutes an important part of the processor.

For security reasons, the key lifetime should be limited (the same key should be used only for a limited amount of data). One of adopted solutions is to use two hierarchical key levels: master keys and session keys. The session keys are generated inside the system and are used for data enciphering. In order to enable data deciphering, a session key has to be exchanged with the communication party. It is therefore enciphered by the master key (shared with the party) and sent together with the encrypted data. In order to generate session keys inside the system, a true random number generator is embedded in the crypto-processor.

Although the cipher/decipher blocks should be included in the processor's datapath, they are considered as trusted black boxes in our project. In our implementation the AES cipher and decipher were used for testing purposes.

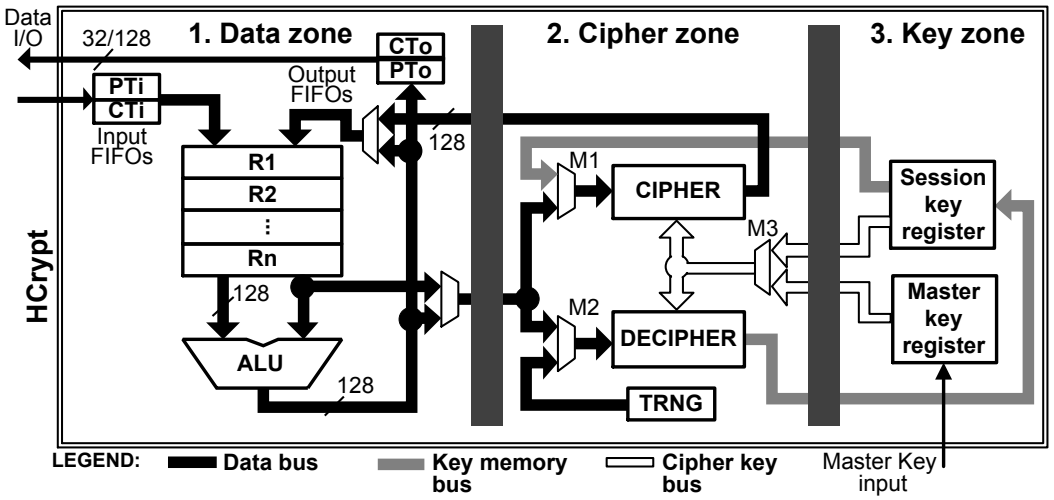

Figure 3.3: HCrypt architecture divided into data, cipher and key zones

Considering many design constraints like performance, size, security, flexibility and programmability necessitates making many crucial decisions resulting in unique balanced hardware architecture. Moreover, a specific-purpose processor with its specific-purpose instruction set requires development of essential software tools (i.e. assembler, etc.), which rises the complexity of the whole project even more. Next, we will present the hardware architecture of the HCrypt crypto-processor, the instruction set, the programmer's model and the software development tools.

3.3.1 Hardware Architecture

The HCrypt crypto-processor consists of three security zones as depicted in Fig. 3.3.

3.3.1.1 Data Zone

The datapath is the most area- and performance-critical part of the crypto-processor. It consists of data busus, data registers and an ALU. Data registers are implemented in four 32 bits wide TDPRAM RAM blocks in parallel. They are accessible through two 128-bit data ports. The ALU has a 128-bit datapath. It carries out all arithmetic and logic operations (XOR, CMP, ROL, CLR, INC, DEC, ADD, etc.) necessary for implementation of most encryption block cipher modes. The authentication modes are not taken into account yet, but can be implemented easily in the future. The HCrypt datapath is optimized for Xilinx Virtex-5 and Virtex-6 FPGAs.

There are four input/output FIFOs in the current version of the processor. They serve as input/output data ports, which convert 32-bit data words into 128-bit words used inside the crypto-processor and vice versa. Additionally, the FIFOs allow to interface external clock domains with the internal clock domain of the crypto-processor. The use of externally clocked 32-bit interface is very beneficial if HCrypt is interconnected with other 32-bit processor system running on its own

clock frequency.

The control logic is responsible for instruction fetching, decoding and execution. Instructions are carried out in 1, 2, 3, 4 or more clock cycles, depending on the type and complexity of the operation. The pipelining is not implemented, since much more resources would be required and speed increase would not be adequate. However, the pipelining can be implemented in future versions of HCrypt. On the other hand, special two-cycle instructions can be used to start cipher or decipher and then continue the execution of subsequent instructions while cipher/decipher is operating. This feature can be excellently used for an implementation of block cipher modes, where some instructions like branching, exclusive or, decrementation, etc., can be executed in parallel with the ciphering. This way one full CFB loop can be performed in 14 clock cycles only. Moreover, when the cipher/decipher output is required by an instruction (e.g. XOR) the instruction waits until cipher output is available. Instructions are 32-bit wide. Current instruction set consists of 29 instructions.

The program is stored in the program memory that is initialized by an external 32-bit interface. The program memory is implemented as one DPRAM block capable of storing 1024 instructions long HCrypt software. The program memory can be expanded if necessary (by concatenating two or more memory blocks).

3.3.1.2 Cipher Zone

Note that multiplexers included in the datapath are organized in such a way, that unintentional or intentional transfers of unencrypted keys outside the crypto-processor would never be possible.

Any trusted symmetric-key enciphering algorithm with 128-bit datapath can be implemented in HCrypt. For this particular implementation AES standard has been chosen. The cipher and decipher have two separate inputs (data and key) and one output (data). Due to iterative nature of the datapath (folded datapath), enciphering or deciphering is carried out in 11 clock cycles. All data and key inputs/outputs are registered in the cipher/decipher.

Although a TRNG is implemented for the final testing, a PRNG was used during HCrypt development to simplify the placement and routing processes. However, PRNG cannot be used in the final HCrypt version for its predictability and must be replaced by a TRNG as for instance PLL-based TRNG presented in [36].

3.3.1.3 Key Zone

Key memory bus and key registers for the key management are shown in Fig. 3.3. The master key (aimed at session key enciphering/deciphering) is stored in the Master key register. This register is capable of storing two 128-bit keys: one for enciphering and one for deciphering. This register is implemented in the logic area of the FPGA. Before the operation, the master key has to be initialized in the crypto-processor via a 32-bit separated dedicated bus by the trusted entity.

The session key registers are implemented using two 128-bit DPRAM blocks. Session keys can be generated inside the processor using the embedded TRNG or they can be received by the crypto-processor in an enciphered form (protected by the master key) and subsequently deciphered using the master key. If generated inside the processor, the session key has to be enciphered using a master key and sent out according to the protocol.

3.3.2 Implementation of HCrypt in FPGA

The physical separation can be achieved only if each zone of the crypto-processor is placed in a separate logic partition, where each partition is mapped into an isolated physical block. Furthermore, each physical block has to be isolated from all other blocks by at least one line of unused grounded CLB blocks. Buses crossing this isolation area must be routed through dedicated trusted bus macros which guarantee preservation of required physical isolation. However, all these advanced design flows are highly dependent on the target FPGA family. In search of the suitable candidate for HCrypt crypto-processor Xilinx Virtex-6 FPGA has been chosen, since it supports all mentioned design flows. For this reason this FPGA is further considered in this work. Other options include older Virtex families and recently the physical isolation flow has become available for the Altera Cyclone III LS family [84]. Nevertheless, the HCrypt crypto-processor can be ported to these FPGAs easily.

3.3.2.1 Physical Separation at Logic Placement Level

The HCrypt crypto-processor implementation is divided into three logic partitions (i.e. data, cipher and key) using the Xilinx PlanAhead development tool. Each partition is placed in a separate physical block. The shape and location of the physical blocks is carefully chosen so that sufficient number of logic, routing and hard-wired resources is present in the block. Moreover, it is recommended to slightly enlarge physical blocks to simplify and shorten the placement and routing processes.

3.3.2.2 Physical Separation at Routing Level

As explained before, high security can be achieved only if crypto-processor zones are carefully isolated from each other. The SCC design flow enables this isolation and implementation is carried out using the Xilinx PlanAhead development tool. First, a reset global signal is routed between physical blocks through Trusted Bus Macros (TBM) and a manually placed buffer is required for every such partition output. Global clock network is the only exception allowed to cross isolation fence without the use of TBMs. Second, special constraints for the physical blocks are set in the PlanAhead tool to ensure that only the selected wires cross the isolation fence.

The isolation effectiveness has to be verified using Xilinx IVT. This tool verifies if all stringent isolation constraints are met, and if so, IVT generates a special isolation certificate. This certificate can be further used as a proof of correct isolation design.

The HCrypt has been designed using the SCC flow, tested using IVT and finally the isolation certificate has been successfully obtained.

3.3.3 Programming Means

When a secure crypto-processor is developed, not only a careful hardware design but also a software development flow must be considered. Besides common instructions, the instruction set takes advantage of the processor specific structure. Namely, it uses different instructions for handling ordinary data and keys. Also, arithmetic and logic instructions are optimized for implementing common cryptographic functions and modes.

3.3.3.1 Instruction Set

The instruction set is divided into four main groups:

- Data transfer instructions: data manipulation between data registers and input/output FIFOs
- Instructions for key generation and transport between ordinary data registers and key data registers
- Data processing instructions: optimized for implementation of cipher modes and key management
- Control instructions: aimed at the program control (branching)

The most important instructions of the proposed instruction set with respect to this grouping are shown in Tab. 3.5.

This instruction set natively supports basic block cipher modes (i.e. CFB, OFB, CTR, CBC-MAC, etc.). The operation of the crypto-processor can be illustrated on the program examples in Tab. 3.6 that realize various basic encryption modes. In this table, PTI represents PlainText Input, PTO PlainText Output, CTI CipherText Input and CTO CipherText Output.

CBC Encryption Block Cipher Mode Example The code illustrated in Tab. 3.6 (part A) enables to encipher the packet consisting of N 128-bit data blocks (N is given in the first data block in the packet) in the CBC mode (see NIST SP 800-38A, p. 10 [1]). However, the CBC mode requires decipher unit in case of deciphering. Although HCrypt includes decipher unit, it is aimed only for protection of session keys and not data. For this reason the CBC block cipher mode cannot be fully implemented. On the other hand, the CBC mode implementation on HCrypt can be used for authentication (i.e. the CBC-MAC authentication mode) where only the cipher unit is necessary.

Table 3.5: The dedicated instruction set

A) Data transfer instructions

Instruction	Cycles	Description
init #, rx	5	Initialize Rx with a 128-bit constant
mov PTI, rx	1	Get one block from plain-text input FIFO and put it to Rx
mov rx, PTO	2	Put data from data reg. Rx to the plain-text output FIFO
mov CTI, rx	1	Get one block from cipher-text input FIFO to data register Rx
mov rx, CTO	2	Put data from data register Rx to cipher-text output FIFO
mov rx, ry	2	Move the contents of data register Rx to data register Ry
mov32 rx, ry	7	Move the least significant 32 bits from Rx to Ry and clear all higher Ry bits

B) Instructions for key generation and manipulation

Instruction	Cycles	Description
genk kx	≥ 12	Generate the key and save it in the key data register Kx
getk rx, ky	13	Decipher encrypted key from data reg. Rx to key reg. Ky
putk kx, ry	13	Encipher the key from key reg. Kx to data reg. Ry
autk rx, kx	13	Validate key from key reg. Kx if its fingerprint matches one in Rx

C) Data processing instructions

Instruction	Cycles	Description
enc rx, kx, ry	13	Encipher the contents of Rx with key Kx and save it in Ry
xor rx, ry, rz	2	Rx XOR Ry and save the result to Rz
add rx, ry, rz	3	Rx + Ry and save the result to Rz
inc rx, ry	3	Increment Rx and save the result to Ry
dcr rx, ry	3	Decrement Rx and save the result to Ry
clr rx	1	Clear Rx
cmp rx, ry	2	Compare Rx with Ry, set the flag Z accordingly
cmp32 rx, ry	2	Compare the least significant 32 bits of Rx and Ry, set the flag Z accordingly
rol32 rx, ry	2	Rotate 128-bit word in Rx by 32-bits to the left and store in Ry

D) Control instructions

Instruction	Cycles	Description
nop	1	No operation
goto addr.	1	Unconditional branching
bneq addr.	1	Conditional branching if not equal (Z = 0)
bdie addr., PTI	1	Branch if input plain-text FIFO is empty
bdie addr., CTI	1	Branch if input cipher-text FIFO is empty
bdof addr., PTO	1	Branch if output plain-text FIFO is full
bdof addr., CTO	1	Branch if output cipher-text FIFO is full
call addr.	1	Save return address to stack, branch to function
ret	1	Branch to return address loaded from stack

Table 3.6: Block encryption modes

A) CBC encryption mode

```
        mov     PTI, r3         ;Get N from PTI→R3
        mov     PTI, r1         ;Get IV from PTI→R1
        mov     r1, CTO         ;Send IV to CTO
cbce:   mov     PTI, r2         ;Get data block from PTI→R2
        xor     r1, r2, r2      ;R1 XOR R2→R2
        enc     r2, kx, r1      ;Encipher R2→R1
        mov     r1, CTO         ;R1→CTO
        dcr     r3, r3          ;Decrement R3 (R3 - 1→R3)
        bneq    cbce            ;Branch to cbce if not zero
```

B) CFB encryption mode

```
        genk    kx              ;Generate new session key
        putk    kx, r1          ;Encipher session key and put it to R1
        mov     r1, CTO         ;Send session key to CTO
        mov     PTI, r3         ;Get N from PTI→R3
        mov     PTI, r1         ;Get IV from PTI→R1
        mov     r1, CTO         ;Send IV from R1 to CTO
cfbe:   enc     r1, kx, r2      ;Encipher R1→R2
        mov     PTI, r1         ;Get plain-text block from PTI→R1
        xor     r2, r1, r1      ;R2 XOR R1→R1
        mov     r1, CTO         ;Put R1 to CTO
        dcr     r3, r3          ;Decrement R3 (R3 - 1→R3)
        bneq    cfbe            ;Branch to cfbe if not zero
```

CFB Encryption Block Cipher Mode Example The code illustrated in Tab. 3.6 (part B) enables to generate the session key that is then used to encipher the packet consisting of N 128-bit data blocks in the CFB mode (see NIST SP 800-38A, p. 12 [1]). In this case, CFB is fully supported since for both enciphering and deciphering operations cipher unit is sufficient.

3.3.3.2 FlexASM Assembler

HCrypt's complex instruction set makes writing programs in machine code more difficult. Long machine code is only hardly readable. For the sake of simplicity, HCrypt software can be written in a simple assembly code. This code can be translated to a machine code using an assembler tool. In order to address this issue, we developed a processor-specific assembler (compiler) tool.

The whole compilation process is illustrated by the flow chart depicted in Fig. 3.4. FlexASM is a two-pass assembler. In the first pass, all labels in the code are examined, and if not previously found, labels are stored in a table. In the second pass, labels are replaced by their numeric representations and instructions are translated

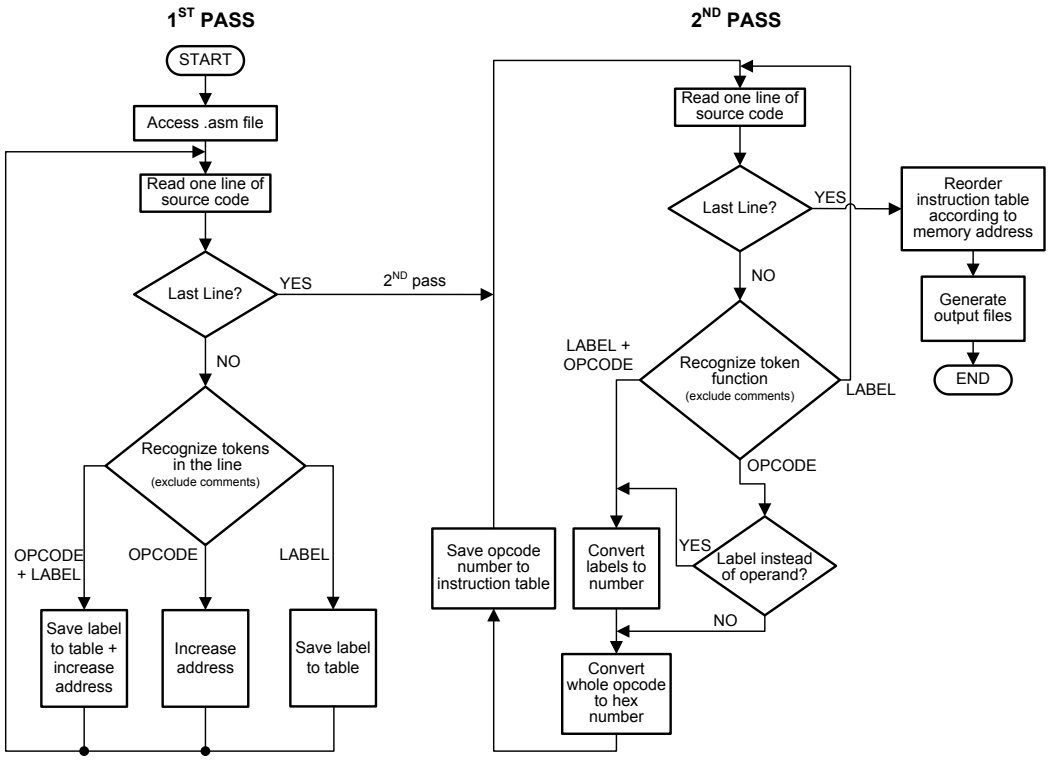

Figure 3.4: FlexASM two-pass compilation flow chart

into the machine code. During initial scanning phase, source code lines are read and divided into tokens and unnecessary parts (i.e. white spaces, comments) are removed. A token is a sequence of characters that are treated as a single unit. Consequently, tokens are examined and treated according to their function (e.g. label are stored in the label table, instruction mnemonics are translated into their opcode representation, etc.). During the second pass, translated instruction words are stored in the instruction table. Finally, the instruction table is reordered according to the corresponding memory address and output files are generated.

Unlike other assemblers, FlexASM has a programmable instruction set which is defined in a separate text file and passed to FlexASM as an input argument. The example of an instruction set definition file is illustrated in Tab. 3.7. Each instruction must be defined by corresponding necessary parameters like a mnemonic, opcode, number of operands (#opr) and a definition of each operand. Optionally, comments can be inserted into the file preceded by the semicolon character (;). Each operand is defined by two values: its size (in number of occupied bits) and its shift (position in an instruction word).

For example, the *enc* instruction has three operands. Operand A occupies 8 least significant bits since its shift is 0. Operand B occupies also 8 bits and it is shifted by 8 bits to the left. Finally, operand C occupies also 8 bits and it is shifted by 16 bits to the left. In this example, the operand A represents register containing

Table 3.7: Instruction set definition file example

```
; ========== CONTROL ========
; MNEMONIC   OPCODE      #OPR   OPA   OPB   OPC   ; COMMENT
nop          0x00000000   0     0  0                    ; no operation
goto         0x01000000   1    12  0                    ; unconditional jump
; ...
; ===== ALU OPERATIONS =====
xor          0x10000000   3     8  0   8  8  8  16  ; A xor B -> C
clr          0x11000000   1                 8  16  ; 0 -> C
cmp          0x12000000   2     8  0   8  8        ; if A=B, 1->Z else 0->Z
; ...
; === SECURITY OPERATIONS ===
enc          0x30000000   3     8  0   8  8  8  16  ; encrypt data(A) with SK(B) -> C
; ...
```

input data to be ciphered with the session key. The session key register is defined by operand B. After the enciphering operation is finished, results are stored into destination register defined by operand C.

Second input argument is the HCrypt source code file. The compilation can be started by the console command *FlexASM.exe <inst_set.txt> <source_code.asm>*. FlexASM translates the assembly code into the machine code, while generating *.raw*, *.c*, *.hex*, *.lst* and *.tab* output files. The *.raw* file is a text file that contains one 32-bit instruction word per line. This format is very suitable since the file can be easily read by a test bench file during HCrypt simulation. If HCrypt software is initialized by an external processor, HCrypt software can be a part of the processor source code. For this reason, *.c* file containing an array of all instruction words is also generated by the FlexASM. Afterwards, this array can be copied&pasted to the processor's c-code source file. If HCrypt source code is a part of an FPGA bitstream, a RAM containing HCrypt software can be initialized with a *.hex* file compatible with the Intel-Hex standard. For debug purposes, list (*.lst*) and table (*.tab*) files are generated too. The list file contains a compilation report. The label table and the instruction table are summarized in the table file.

The FlexASM assembler was written in the standard ANSI C language and compiled using Borland Builder C++ 6.0. Due to its ANSI C compliance, FlexASM can be compiled by any other compatible compiler under any operating system. The assembler is a console application since it does not have a graphic user interface.

Although FlexASM was developed to facilitate work with the HCrypt crypto-processor, its high flexibility allows its use with other embedded processors. Moreover, FlexASM source code is freely available and thus easily modifiable if further flexibility is required. These features make it a very convenient tool for rapid low-level software development.

3.3.4 Communication Protocol

In order to achieve secure key management, a robust cryptographic protocol is essential. The cryptographic protocol has to clearly define all necessary steps to ensure that session keys are exchanged and authenticated correctly. The session key exchange but also data has to be protected by the encryption, and the data authenticity has to be guaranteed. The communication protocol implementation is supported by the HCrypt software and all security critical operations are carried out by dedicated instructions.

We present an example of the cryptographic protocol for communication between Alice (A) and Bob (B) (see Fig. 3.5) in order to illustrate efficiency of the proposed structure. The key exchange part of this protocol is based on the authenticated point-to-point key update protocol described in Sec. 2.5.4.2. In practice, any other common cryptographic protocol can be implemented.

Tasks 1, 2, 3, 8 and 9 are performed solely in the protected area (cipher and key zone) and tasks 5, 6, 7 are executed only in the unprotected area (data zone). Tasks 4 and 10 represent iterative implementation of encryption modes (EM) performed partly by the unprotected area (registering and xor-ing of subsequent data blocks) and partly by the protected area (enciphering E and deciphering E^{-1}). In this protocol, we assume that a Trusted Entity TE pre-initialized the same enciphering (MK) and the same authentication (AMK) master keys to both devices, and that the keys are saved in their master key registers (protected area). The key exchange protocol is based on the symmetric key cryptography. In the first step, the device starting the communication (A), generates a new Session Key SK: the unprocessed session key CSK is generated by the TRNG, then it is post-processed cryptographically in the decipher using the master key MK and saved in clear in the session key memory (Task 1 in Fig. 3.5). Next, the session key SK is enciphered using master key MK and transferred to the unprotected area (Task 2). Finally, a digital fingerprint FP_A is generated by enciphering SK using the authentication master key AMK (Task 3).

When both the session key SK and its fingerprint FP_A are generated, Task 4 can be executed in a loop: data blocks ($DATA_i$) are sent from the data zone to the cipher zone, where they are enciphered using SK and sent back to the data zone as $CDATA_i$. ALU combines input and output blocks according to the encryption mode algorithm (EM) and computes $MCDATA_i$. Finally, packet P containing the enciphered session key CSK, its fingerprint FP_A, and enciphered data blocks $MCDATA_i$ is created (Task 5). The packet is sent to device B (Task 6). The crypto-processor on the side B receives the packet P (Task 7) and extracts the enciphered session key CSK and its digital fingerprint FP_A. The key is then sent to the cipher zone, where it is deciphered using the master key MK and stored in the session key memory (Task 8). A fingerprint FP_B of the session key SK using the master authentication key AMK (Task 9) is generated and sent back to the data zone, where it is compared with the received fingerprint FP_A (Task 10). If FP_A and FP_B are the same, the session key is authenticated and can be used for

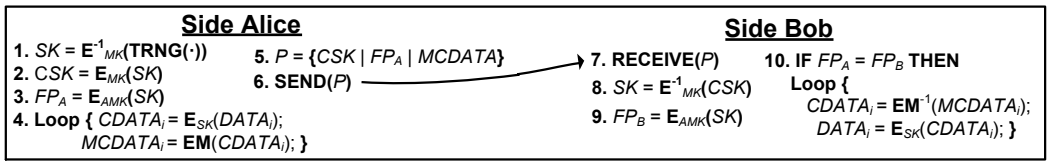

Figure 3.5: Authenticated key update Communication protocol between two devices

data enciphering/deciphering (the loop in Task 10).

The packet, compatible with the communication protocol, can have the straightforward structure as depicted in Fig. 3.6. Packets sent to HCrypt for processing are called *Command packets* and packets created by HCrypt and sent out are called *State packets*. Structure of both packet types is very similar.

Each packet begins with a *Preamble*. The Preamble is a 32-bit special character sequence which enables HCrypt to recognize the packet header and synchronize to it. Afterwards, a 32-bit *Command* word is examined. The command word can specify if a session key is a part of the packet and needs to be deciphered or a session key is not a part of the packet and needs to be generated. The command word can also specify whether data have to be enciphered or deciphered. The command word is followed by a 32-bit *Block mode* word that specifies which cipher block mode (e.g. ECB, CFB, CBC-MAC, etc.) HCrypt has to perform on the packet data. The last 32-bit word in the packet header which specifies how many 128-bit data blocks are in the packet. The header is followed by the 128-bit enciphered *Session key* (if specified in the *Command* word). Session key needs to be deciphered to be used for data protection. Since session key authenticity has to be provided, additional 128-bit *Session key fingerprint* is included in the packet. If a session key is authenticated, it can be stored in the session key register and used for data protection. These initial packet information is followed by 128-bit *Data blocks* which are processed according to the header settings. Finally, the packet is terminated by a 128-bit *Footer* word, which can be optionally used for the packet integrity check.

As a reaction to the command packet, HCrypt responds back by generating the state packet. Besides the preamble and number of 128-bit data packets, packet header contains two 32-bit state words specifying what kind of operation has been carried out on data (i.e. enciphering/deciphering, block cipher mode). State words also indicate if the operation was successful. Besides the 128-bit header word, all other parts of the packet are the same as in command packets.

Since the packet header can be broken into four 32-bit words, each containing a different piece of information, HCrypt includes some necessary instructions capable of operating on 32-bit words (i.e. 32-bit left rotation, 32-bit move operation and comparison of 32 least significant bits).

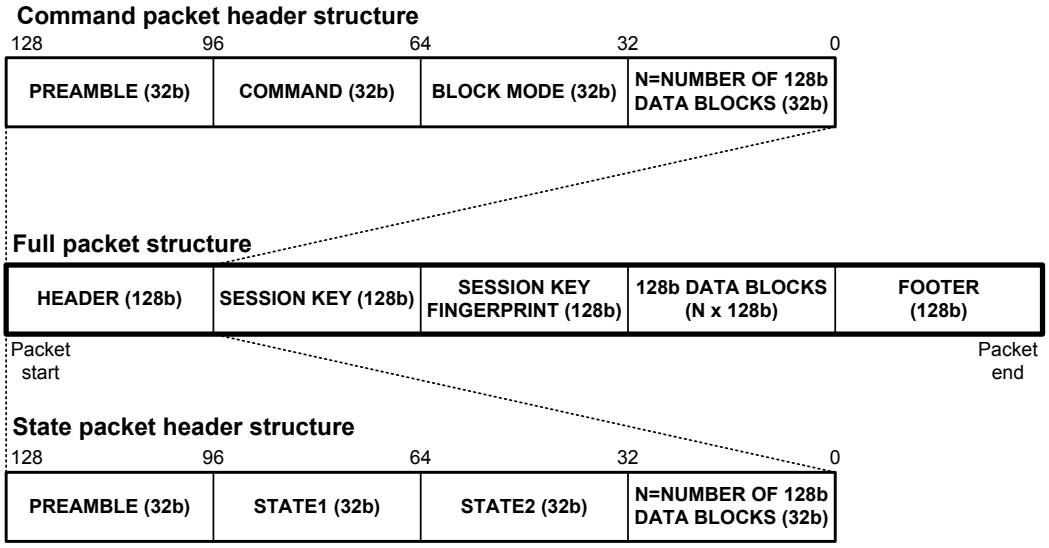

Figure 3.6: Structure of the packet supported by HCrypt

3.4 Implementation Results

HCrypt was described in VHDL language and implemented in Xilinx Virtex-6 XC6VLX240T FPGA (Xilinx ML605 board) using Xilinx ISE Web ver. 12.4. The system utilizes just fine-grain FPGA resources and embedded RAMs/FIFOs. The functionality was simulated using the Xilinx ISim tool. Subsequently, timing simulation as well as hardware tests were carried out. During hardware tests, a small Cypress USB module was used for interconnecting HCrypt with a PC. Next, we present the resource utilization report and results of the simulation and the hardware test.

3.4.1 Cost Evaluation

Utilization and distribution of HCrypt resources is illustrated in Tab. 3.8. These resources do not include USB interface since its size is negligible and it was used only during tests in hardware. Note that the AES decipher (see Fig. 5.2) requires 50% more logic resources than the cipher (see Fig. 5.1). This difference is a result of an additional 128-bit XOR block and a more complex InvMixColums block (see InvMixColumns decompositioncan in [85]) present in the AES decipher.

3.4.2 Simulation

The HCrypt simulation is very important, because it can easily uncover errors in the processor's complex design and also helps the user to debug the HCrypt software. Cycle-accurate simulation enables to estimate HCrypt's maximum theoretical data throughput for current software by calculating the number of clock cycles needed for critical parts of the code (i.e. loops). If big data packets are processed, time

Table 3.8: Utilization of resources in XC6VLX240T

	Slices		RAM kb	
TOTAL	1422	100.0%	1188	100.0%
→ **Crypto-Proc.**	877	61.7%	828	69.6%
→ **AES Cipher**	216	15.2%	180	15.2%
→ **AES Decipher**	329	23.1%	180	15.2%

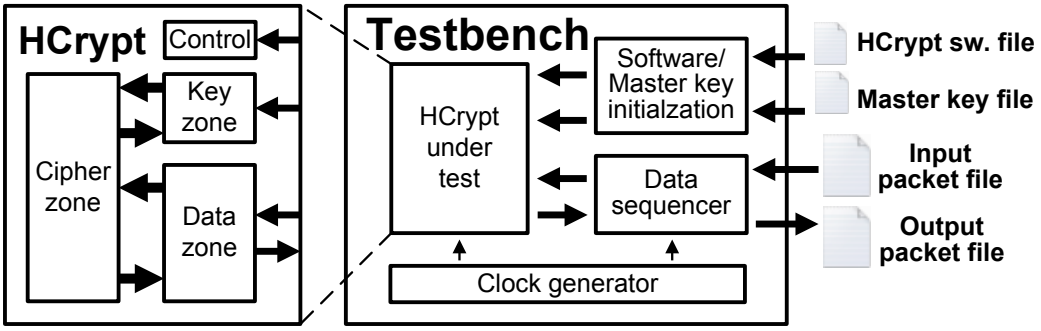

Figure 3.7: HCrypt simulation procedure

spent to process overhead information (i.e. header, footer, session key and session key fingerprint) is negligible to the time spent to process data. In this case, it is the most important to optimize critical loops for data processing. On the other hand, data throughput can decrease significantly if the packet size is small and the overhead impact is significant. In this case not only data processing loops, but also all header and footer processing need to be optimized.

HCrypt simulation was performed using the Xilinx ISim tool. First, simulator read and compiled all HCrypt VHDL files including a test bench. Subsequently, simulation was carried out and selected HCrypt signals were displayed in the waveform. During the simulation process, information messages were displayed in the simulator transcript window.

The HCrypt simulation procedure is illustrated in Fig. 3.7. The corner-stone of the HCrypt simulation is a complex *Test bench*. First, the test bench read *HCrypt software file* containing HCrypt instruction words and initialized HCrypt program memory. Note that the software file has a .raw file format and it was generated by the FlexASM tool. Second, the *Master key file* was accessed and the master key was transferred into the master key register. When the initialization of both HCrypt software and its master key was finished, HCrypt started to execute its software. HCrypt operation was simulated by reading packets from the *Input packet file* and writing resulting packets to the *Output packet file*. Moreover, HCrypt instruction flow and timing information were given in a form of messages that were displayed in the simulator transcript window.

Tab. 3.9 shows how many clock cycles are required to carry out different packet processing operations. In this example, data are processed using the CFB block

Table 3.9: Number of clock cycles required for the packet processing

	Packet overhead				Data
	Header decoding	Key decryption	Key authentication	Other	One 128-bit CFB loop
Clk. cycles	34	12	13	30	14

Table 3.10: Dependence of maximum throughputs on number of 128-bit data blocks in the packet

128b data blks.	1	2	4	6	8	10	12	14	16
Packet data in kb	0.125	0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0
Packet speed in Mb/s	535.4	572.2	626.0	663.4	691.0	712.1	728.8	742.3	753.5
Data speed in Mb/s	107.1	190.7	313.0	398.1	460.6	508.6	546.6	577.4	602.8
Packet overhead in %	80	66.6	50.0	40.0	33.3	28.6	25.0	22.2	20.0

128b data blks.	20	32	50	64	100	128	256	512	1024
Packet data in kb	2.5	4.0	6.25	8.0	12.5	16.0	32.0	64.0	128.0
Packet speed in Mb/s	771.0	801.9	824.0	833.4	846.4	851.7	861.5	866.7	869.3
Data speed in Mb/s	642.5	712.8	762.9	784.4	813.8	825.8	843.3	859.9	865.9
Packet overhead in %	16.7	11.1	7.4	5.9	3.8	3.0	1.5	0.8	0.4

cipher mode (see Sec. 2.2.1.4). Here, the CFB loop constitutes a critical loop.

The simulation clock frequency was set to 100 MHz. Using this accurate timing information, maximum packet and data throughputs could be calculated. The maximum throughputs are displayed in Tab. 3.10. The maximum packet throughput (with packet overhead), when processing 1024 data words in the packet, reaches 869.3 Mb/s if HCrypt operates at the clock frequency of 100 MHz. The data throughput (without packet overhead: header, session key, fingerprint and footer) reaches 865.9 Mb/s when processing 1024 data words.

3.4.3 Hardware Tests and Benchmarks

Although simulation is very important for debugging initial design errors, the hardware test can definitely prove the system functionality and show the real system performance. The hardware tests setup is depicted in Fig. 3.8. HCrypt together with the USB interface was configured inside an FPGA which was interconnected with the Cypress USB module. The USB module communicated with the PC containing the Cypress USB driver and a benchmark application.

Two frequency domains were present in the FPGA. HCrypt was operating at the 100 MHz internal clock frequency as well as all its internal parts including hte AES cipher and decipher units. The USB interface was operating at the 48 MHz clock frequency. Synchronization between the USB interface and HCrypt clock domains was provided by the HCrypt Input/Output FIFOs.

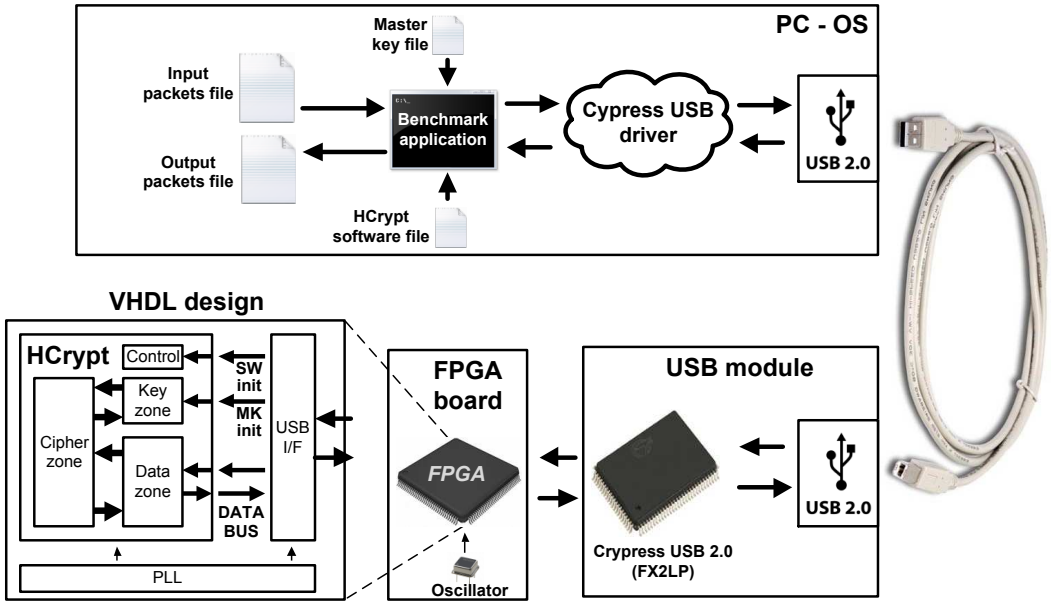

Figure 3.8: HCrypt hardware test setup

The whole testing process started by initializing HCrypt with its software and the master key. The HCrypt software, generated by the FlexASM assembler, was stored in the *HCrypt software file* on the PC. The benchmark application accessed this file and transferred all instruction words through the USB module to the FPGA. Subsequently, the USB interface received instruction words and initialized HCrypt program memory. When the program initialization was finished, the benchmark application accessed the *Master key file* and transferred the master key to the FPGA.

When the master key was present in the master key register, HCrypt started to execute its program code and waited for input packets. The PC application read the *Input packet file* and sent packets to HCrypt input FIFO. HCrypt decoded the packet header, extracted and deciphered session keys and carried out 128-bit CFB block cipher mode on packet data (1024 word of 128-bit size each). Subsequently new packet containing results was sent back to the PC. The benchmark application read the resulting packet, verified it and saved it to the *Output packet file*.

Unfortunately, the maximum bandwidth of the USB interface was far below the HCrypt's throughput. For this reason, the whole packet was first stored in the FPGA embedded RAM and only then it was sent to HCrypt. Results were stored in the same embedded RAM memory and then sent to the PC. The USB interface entity included a small counter for embedded testing which was active during HCrypt computations. The counter value was sent to the PC for the speed evaluation.

In order to reach the clock frequency of 100 MHz, extra timing constraints were required in the PlanAhead to shorten critical paths. Finally the maximum data throughput reached 824.7 Mb/s and the packet throughput reached 827.9 Mb/s.

3.5 Discussion

The main difficulty concerning the portability of the proposed VHDL code is related to the arrangement of FPGA-specific embedded memory blocks. As mentioned above, HCrypt was mapped to Xilinx Virtex-6 device where each TDPRAM block has two 32-bit input and output ports. However, in Altera Stratix II FPGA each TDPRAM block has two 16-bit input and output ports, thus twice more memory blocks have to be instantiated in order to maintain the same width of the datapath (128-bit). Moreover, when adapting to Microsemi Fusion or SmartFusion devices, where each embedded TDPRAM block has two 8-bit input and output ports, 16 memory block have to be instantiated to maintain the same width of the datapath.

Resource distribution report showed that the most logic resources are dedicated to the ALU. On the contrary, the most embedded RAM blocks were used for Input/Output FIFO instantiations, in order to interconnect 32-bit external buses with 128-bit internal datapath with different clocks.

It has been observed that current AES cipher and decipher blocks constitute the longest critical path, therefore optimization of the cipher can result in better performance. Some long critical paths were also reported in the ALU (i.e. adder/subtracter unit), thus this part has to be optimized in the future as well.

The obtained maximum real data throughput of 824.7 Mb/s was close to the maximum theoretic data throughput of 865.9 Mb/s. The reason could be in the inaccurate simulation model of I/O FIFOs which does not match exactly the real behavior of the embedded FIFO block.

Crypto-processor solution offers high flexibility. A straightforward software modification enables rapid implementation of cipher modes. Higher flexibility can be achieved only by hardware modifications which are not easy to carry out. However, partial reconfiguration technique can be used to modify or replace the whole HCrypt security blocks (i.e. cipher/decipher) thus raising flexibility even more.

In the first HCrypt version, only software attacks on the key management were considered. However, side-channel attacks represent a serious threat nowadays and must not be neglected. For this reason, countermeasures needed to be implemented in next HCrypt versions. We deal with this threat in Chap. 5 and propose countermeasures, raising the resistance to side-channel attacks for no extra cost.

3.6 Conclusions

In this chapter, we have shown how crypto-processors supporting secure key management can be constructed. Keys can be safe inside the device only if they are well separated from the data processing part. Thus, for achieving secure key management, novel separation rules have been proposed. These rules suggest separation at protocol, system, architectural and physical level. If they are carefully respected, confidential keys will be protected against software attacks in the future devices, so these devices will become secure by design.

The separation rules are the corner-stone of the new HCrypt crypto-processor. HCrypt has a 128-bit wide datapath, embedded AES cipher, AES decipher and a TRNG, and an instruction set optimized for block cipher modes and packet management. HCrypt is divided into data, cipher and key zone each having different security privilege. The most secure is the key zone, where secret keys are stored in clear. HCrypt protects data using the generated session keys. The session keys must be securely exchanged prior to data exchange using the master keys. Master keys are transferred to HCrypt via dedicated 32-bit bus. HCrypt operation, security, programmability have been demonstrated and throughput have been measured. Hardware tests indicated data throughput of 824.7 Mb/s, while HCrypt exchanged session keys and processed data in packets using 128-bit CFB block encryption mode.

Important issue during the HCrypt design was the development of programming means. First, software code written in the machine code became long and hardly readable. Second, frequent changes in the HCrypt instruction set made software modifications difficult. For this reason a novel flexible assembler tool – FlexASM has been developed. Advantage of FlexASM is the possibility to define instruction set in a text file.

Presented separation rules consider only software attacks against key processing and storing. However, side-channel attacks represent also a considerable threat and must not be neglected. These attacks will be further examined and an extension to security rules, considering also side-channel attacks, will be presented and demonstrated in Chap. 5.

If data throughput is very high, not only session keys, but also master keys require to be regenerated from time to time, so that an attacker will not be able to collect enough leaked information for a successful attack. We suppose that master keys are installed to the device by a trusted entity. However, frequent master key changes raise significantly device transportation costs. Indeed, the master key exchange is an open problem and can be presumably simplified using asymmetric cryptography in the future.

The work presented in this chapter was publised in [80], [63]. In order to support public discussion about these interesting issues and to motivate public to further testing of separation rules, HCrypt VHDL project is freely available at SecReSoC project web page: *http://labh-curien.univ-st-etienne.fr/secresoc/*

Chapter 4

Crypto-coprocessor with Secure Key Management

Contents

In Chap. 3 we have shown that there are no customized GPP or crypto-processor systems that support the secure key management. For this reason we have proposed original separation rules and demonstrated them on the novel HCrypt crypto-processor. However, sometimes it could be useful to use a powerful GPP in conjunction with a crypto-coprocessor. Next, we will examine the existing crypto-coprocessor architectures considering different aspects and try to adapt the previously proposed separation rules to the design of a novel crypto-coprocessor.

4.1 Crypto-coprocessors - State of the Art

Cryptographic coprocessor is a module outside of the GPP that accelerates cryptographic computations. This module is controlled by the GPP. Cryptographic functions inside the module are implemented in a very efficient way and so the throughput is very high. Secret keys are not stored in the crypto-coprocessor memory, but are stored as data in the processor data registers or main memory. If the crypto-coprocessor is implemented in FPGA, parts of the crypto-coprocessor can be reconfigured during the system operation. This technique increases the flexibility of the crypto-coprocessor while reducing its size.

In Chap. 3 we have dealt with crypto-processors. In order to differentiate between terms crypto-coprocessor and crypto-processor, we define crypto-coprocessors as follows: A crypto-coprocessor contains one or several implementations of cryptographic functions. The crypto-coprocessor is not programmable, but has to be controlled by a master GPP.

A lot of work has been done in the field of crypto-coprocessors. One of the first DES crypto-coprocessors was published by Verbauwhede et al. [86]. It contained four write-only key registers and supported four basic block cipher modes, MAC generating functions and a Random Generation Function (RGF). The first Rijndael crypto-coprocessor (i.e. processes up to 256-bit data blocks) was proposed by Kuo et al. [87] and after silicon fabrication it ran faster than estimated [88].

A single-core AES crypto-coprocessor connected to the LEON processor has been described in [89], [90] and [91]. This work has given one very interesting conclusion. Although hardware acceleration is significant, the communication interface creates bottlenecks and thus reduces system performance. The CryptoBooster crypto-coprocessor is dedicated to acceleration of the IDEA algorithm [92].

Multi-core AES crypto-coprocessors controlled by the MOLEN processor [93] are given in [94] and [95]. Another multi-core high-performance AES crypto-coprocessor, presented in [96], is composed of several AESTHETIC crypto-coprocessors described in [97]. A secure and non-secure version of AES crypto-coprocessor were implemented in silicon and their resistance to side-channel attacks was compared in [98]. Very interesting is the SAFES multi-core crypto-coprocessor proposed by Gogniat et al. [99]. SAFES concentrates not only on performance but also on overall system security. This security is provided by special units like Bus monitor, Clock monitor, Channel monitor, Power monitor etc.

The comparison of these crypto-coprocessor architectures can be found in Tab. 4.1 (the implementation differences) and in Tab. 4.2 (cryptographic engine differences).

Although this work is concentrated on the symmetric keys cryptography, some crypto-coprocessor systems were proposed for asymmetric key cryptography [100], [101] or combination of both symmetric and asymmetric key cryptography [102], [103].

However, all the mentioned crypto-coprocessors were oriented on performance. Not much work has been dedicated to protection of critical data from software attacks. Before entering crypto-coprocessor, the secret keys pass through the main

Table 4.1: Different implementations of crypto-coprocessors

Name [ref]	Target technology	Cipher mode	Mbps/MHz	F_{max} in MHz	FPGA slices	FPGA RAM	ASIC gates area
AES Processor [89], [90]	ASIC 0.18 μm	AES-ECB	11.60	295	-	-	73 KG 0.73 mm^2
CryptoBooster [92]	FPGA XCV1000	IDEA-ECB	16	33	?	?	-
AESTHETIC 1 core [97]	ASIC 0.25 μm	AES-ECB AES-CBC	12.80	66	-	-	200 KG 6.29 mm^2
AESTHETIC 3 cores [97]	FPGA XCV2V6000	AES-ECB AES-CBC	36.80	50	27561	0	-
CrCU [94]	FPGA XCV2VP30	AES-ECB	0.59	100	847	216 KB	-
AES-MS 2 cores [95]	FPGA XCV2VP30	AES-ECB AES-CBC	25.60	100	2161	432 KB	-
SAFES [99]	FPGA XCV2VP30	AES-ECB	10.60	37.8	2192	0	-
Rijndael [87]	ASIC 0.18 μm	Rijndael-ECB	18.60	100.0	-	-	173 KG 3.96 mm^2
Rijndael2 [88]	ASIC 0.18 μm	Rijndael-ECB	15.23	154.0	-	-	173 KG 3.96 mm^2
AES-WDDL [98]	ASIC 0.18 μm	AES-ECB	14.69	69.0	-	-	596 KG 5.95 mm^2
DES coproc. [86]	ASIC 2.4 μm	DES-ECB	2.5	12.0	-	-	18 Ktrans. 25.0 mm^2

GPP. During this transfer, the secret keys are temporarily stored in the GPP data registers. This is the right moment for the attacker to exploit. If he succeeds to modify the GPP software, the keys can be easily transferred out of the GPP. Clearly, security perimeter must not exceed the crypto-coprocessor border and as a consequence secret keys must never enter the GPP in clear. Next, we will present *secure crypto-coprocessor* supporting secure key management.

4.2 New Rules for Securing Key Management

The separation rules presented in Sec. 3.2 can also be applied on cryptographic coprocessors. In order to counter software attacks, secret keys must not be stored in clear in the GPP data registers. For this reason it is essential to physically separate the place where the secret keys are stored from the GPP where data are stored and processed. Next, we explain the differences, from the crypto-processor case, that must be considered when applying separation rules on crypto-coprocessors to guarantee secure key management on GPP systems.

4.2.1 Separation at Protocol Level

The goal of the protocol is to define how to protect secret keys during their generation, storage and exchange. Exchanged keys can be considered protected only if their confidentiality and authenticity is guaranteed. Indeed, if the keys were deciphered in the GPP using a software decipher, they could be exposed to software attacks. Thus, secret keys have to be deciphered and authenticated in a dedicated unit, outside the GPP in a crypto-coprocessor, and never leave this unit in clear.

Table 4.2: Summary of crypto-coprocessors' characteristics

Name [ref]	Year publ.	Supported algorithms	Processing architecture	Key storage	Number of crypto cores	Hardware reconfig.	Main applic.
AES processor [89], [90]	2004	AES-ECB, CCM, CBC-MAC	Cust. SPARC-V8 32b LEON-2	Embedded key register	1 full hardware AES engine	no	IPSEC, VPN
CryptoBooster [92]	1999	IDEA, DES	Hardware reconf. core	Session memory	1 full hardware blk. cip. engine	yes	Network security
AESTHETIC [97]	2009	AES-ECB, CBC	Host process. + hw. AES acceler.	Embedded key generator reg.	1 to 3 full AES engines	yes	Network security
CrCU [94]	2006	AES-ECB, CBC SHA-128/256	Molen process.+ several AES cores	Embedded key register	CrCU number not limited	no	Trusted computing
AES-MS [95]	2008	AES-ECB AES-CBC	Molen processor + two AES cores	Embedded key register	2 AES cores	no	VPN
SAFES [99]	2008	AES, SHA, others	Reconf. hardw. accelerator	Embedded key register	4 parallel crypto primitives	yes	Embedded system security
Rijndael [87]	2001	Rijndael-ECB	hw. Rijndael full core	Embedded key register	1 full Rijndael hw. engine	no	Embedded system security
Rijndael2 [88]	2003	Rijndael-ECB	hw. Rijndael full core	Embedded key register	1 full Rijndael hw. engine	no	Embedded system security
AES-WDDL [98]	2006	AES-ECB, OFB, CBC-MAC	hw. AES, WDDL impl.	Embedded key register	1 full AES hw. engine	no	Embedded system security
DES coproc. [86]	1991	ECB, CBC, OFB CFB, MAC, RGF	hw. DES core, mode logic	Embedded key registers	1 or 3 DES cores in series	no	Trusted computing

The protocol must clearly separate key management and data processing tasks and determine by which blocks the tasks are performed. In fact, security-critical tasks (i.e. key generation/transmission/reception, data enciphering/deciphering, authentication, etc.) must be performed using dedicated crypto-coprocessor instructions, while security-uncritical tasks can be provided by general purpose instructions.

The higher-level keys must never leave the protected area inside inside the crypto-coprocessor, while the lower-level keys can leave this area in encrypted form and be processed by the GPP.

4.2.2 Separation at System Level

The principle of the separation at the system level is illustrated in Fig. 4.1. Unlike the crypto-processor system concept, the data zone is replaced by a complete GPP outside of the crypto-coprocessor. The remaining cipher and key zones are included inside the crypto-coprocessor and are the same as presented in Chap. 3, so the same principle holds for the both cases.

4.2.3 Separation at Architectural Level

Similarly to the crypto-processor, in case of the crypto-coprocessor the same rules have to be applied for the separation at the architectural level. Moreover, interfaces between partitions must be designed to respect special constraints: data buses must be unidirectional and the communications must be controlled only by the GPP. A straightforward interface design simplifies implementation of bus macros in partial reconfiguration or physical isolation design flows.

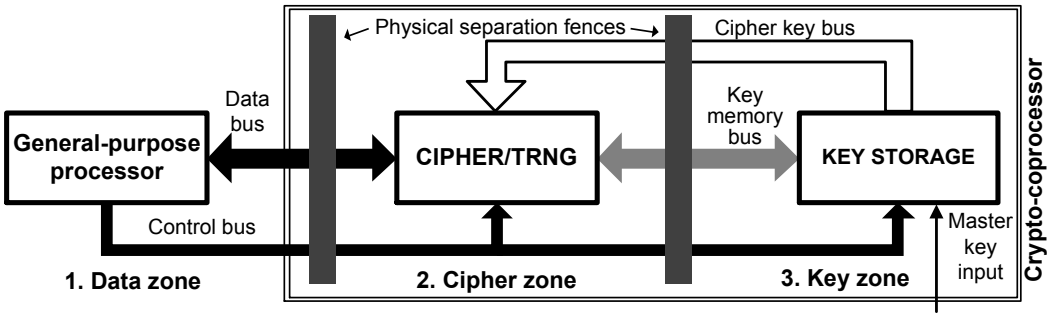

Figure 4.1: Separation of key storage and data storage at system level for crypto-coprocessor

4.2.4 Separation at Physical Level

The way how to separate secret key storage from the GPP at physical level is the same as in the case of crypto-processors.

4.3 Extension of Separation Rules to Crypto-coprocessors

The presented HCrypt crypto-processor is a very good choice when the system is developed from the scratch. On the other hand, when the system cost and compatibility is considered, a GPP in conjunction with a cryptographic coprocessor represents a very interesting tradeoff. In this case GPP can perform all general tasks and the crypto-coprocessor performs security related tasks only (i.e. enciphering, deciphering, key management). Furthermore, stringent separation constraints are required only within the crypto-coprocessor unit and no GPP operation can lead to a disclosure of confidential keys stored in the crypto-coprocessor. This new concept can be used for managing the keys by any GPP in a secure way.

Next, we present the crypto-coprocessor that could be interfaced with any GPP IP implemented in FPGA. This particular secure crypto-coprocessor supporting secure key management will be further denoted as *HCrypt-C*.

The GPP in conjunction with the HCrypt-C crypto-coprocessor is depicted in Fig. 4.2. Before interconnecting HCrypt-C with the GPP, HCrypt-C interfaces have to be adapted to the GPP interface. For this reason, a bus translation unit (further referred as a *wrapper*) has to be inserted between HCrypt-C and the GPP interface. However, for every GPP interface a different wrapper block has to be implemented. In case of different GPP and HCrypt-C interface bus widths, the wrapper is responsible for synchronization of transfers on both sides. Accordingly, control instructions from the GPP interface have to be translated to HCrypt-C control signals.

Next, we describe three most common interface types and discuss their advantages and disadvantages.

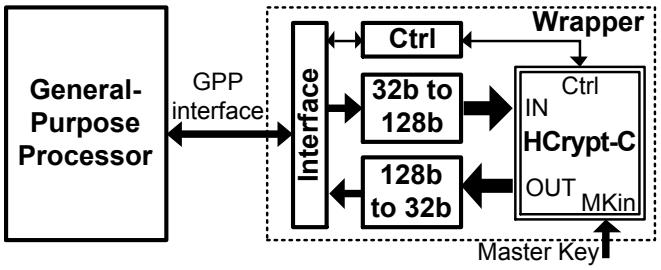

Figure 4.2: Cryptographic system containing the HCrypt-C crypto-coprocessor interconnected with the GPP through the wrapper block

4.4 Interfaces between GPP and the HCrypt-C Crypto-coprocessor

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

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 HCrypt-C crypto-coprocessor 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 communication, 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 HCrypt-C 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 [104], can be used.

Examining the organization of current systems, one can distinguish the following three types of interconnections between the GPP and the coprocessor, while considering aspects like latency, width, topology, security, cost, etc.

4.4.1 Internal Processor Bus

In this case, the crypto-coprocessor is included in the processor data-path, so it becomes its module. Data pass to this module directly from registers as operands. Results are returned to the registers. The module is controlled directly by the processor control unit through a dedicated control bus. The advantage of this solution is in its minimal latency which leads to very high performance. Unfortunately, the crypto-coprocessor is a part of the processor's critical path, and therefore it can slow the whole system down. A high security level is naturally achieved by the point-to-point connection nature.

4.4.2 Dedicated Coprocessor Bus

Here, the crypto-coprocessor 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 topology. Considering the connection between a single crypto-coprocessor and a processor, the connection has a point-to-point nature therefore high security level is maintained.

4.4.3 Peripheral Bus

In this way of interconnection, the crypto-coprocessor is connected to the processor through a bus using a point-to-multipoint connection. In the best case, the crypto-coprocessor 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, a point-to-multipoint bus is less suitable for security applications because data transfers between the security unit and the processor can be potentially eavesdropped by other units connected to the same bus.

4.5 Design of the Crypto-coprocessor/Processor Pairs

The stringent separation rules are the corner-stone of the novel HCrypt-C crypto-coprocessor concept. The implementation of HCrypt-C is illustrated in Fig. 4.3 and three separated security zones (the processor interface, the cipher, and the key zone) can be clearly distinguished. HCrypt-C has a 128-bit wide datapath. Similarty to HCrypt, three buses are used: a data bus (in black), a key memory bus (in gray) and a cipher key bus (in white). Separation rules are strictly applied: key buses never pass through the processor interface zone and data buses never pass through the key zone. Secret keys can never leave the key zone without passing through the cipher.

As presented in Sec. 4.2, any enciphering algorithm can be used in HCrypt-C. In order to validate the system architecture, we use a 128-bit AES, because it is the most common currently used algorithm.

Similarly to the HCrypt crypto-processor concept, keys are organized in two hierarchical levels: high-level master keys and low-level session keys. Session keys are generated inside HCrypt-C by a TRNG and post-processed by the decipher core or received from the processor and deciphered and authenticated using master keys. For the sake of simplicity, a PRNG has been implemented, but any TRNG principle can be used in the real application. Session keys are used only for data enciphering/deciphering (using cipher modes) and authentication (e.g. using CBC-MAC mode).

Special care has been taken (i.e. each zone is placed to separate physical block,

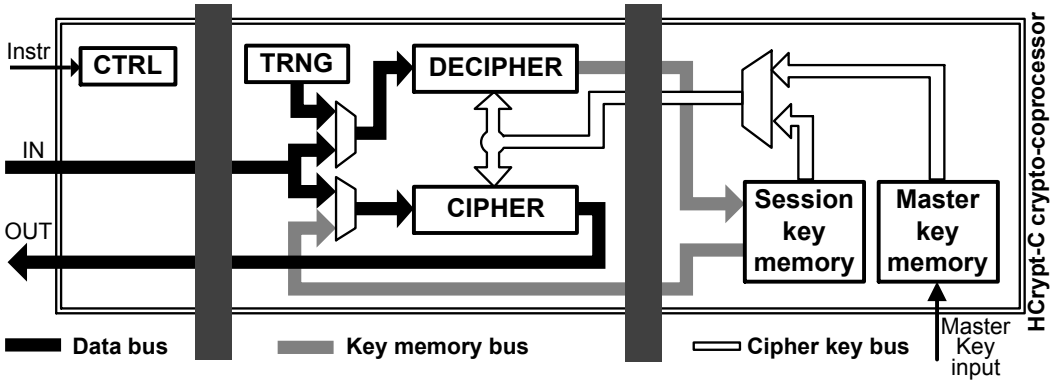

Figure 4.3: HCrypt-C crypto-coprocessor implementation

synthesis attributes, etc.) to force synthesis tools to respect physical separation of zones so that design maintains its natural division even after synthesis.

Since HCrypt-C complies with stringent separation rules, it is secure by design, and no software or protocol attacks can result in disclosure of secret keys. For this reason, any protocol can be implemented, while the key protection remains the same. Of course, the proposed solution will not resist protocol attacks that do not target secret keys, such as service denial attacks. These should be dealt with at the software level, which is beyond the scope of this work.

Next, we present three different ways how the HCrypt-C crypto-coprocessor can be interconnected with GPPs and we illustrate this concept on thee different processor implementation examples:

1. Altera NIOS II – Internal processor bus interconnection is used

2. Xilinx MicroBlaze – Dedicated coprocessor bus interconnection is used

3. ARM Cortex M1 – Peripheral bus interconnection is used

4.5.1 Altera NIOS II GPP with HCrypt-C Crypto-coprocessor

This kind of GPP and HCrypt-C interconnection was implemented in the Altera NIOS II processor, as illustrated in Fig. 4.4. HCrypt-C communicates with NIOS II using a wrapper. Control instructions are passing to the wrapper directly from the processor control unit. In this case, all HCrypt-C operations are implemented as custom instructions of the processor. NIOS II custom instruction technique enables users to include a custom unit into the processor's datapath. This way, the processor's critical path is extended by the datapath of HCrypt-C, what affects the processor maximum clock frequency.

Special constraints are applied on the design, so that all three security zones will remain separated from each other after the synthesis and placement.

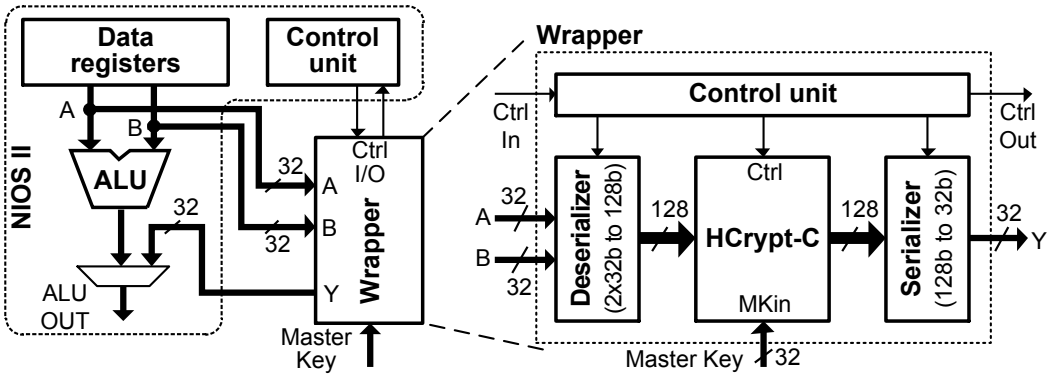

Figure 4.4: NIOS II interconnected with the HCrypt-C crypto-coprocessor wrapper via the internal processor bus

4.5.1.1 Wrapper Structure

The communication link between NIOS II and HCrypt-C, the wrapper interface, is depicted in the right part of Fig. 4.4. Data are moved in 32-bit words from the NIOS II registers into deserializer which outputs full 128-bit word. Since two NIOS operands are available, 64 bits of data can be transported to the HCrypt-C deserializer at once. Results are stored in the Serializer of which the output is returned back to NIOS II registers in a form of four 32-bit words. Both the serializer and deserializer are implemented in logic.

Since control signals can be provided to HCrypt-C directly from the processor's control unit, only a simple control unit is necessary to derive HCrypt-C control signals. HCrypt-C operation can be monitored using status flags provided by the wrapper control unit.

4.5.1.2 Software

The NIOS II software was written in the ANSI C code using the Eclipse development environment. All NIOS II custom instructions are listed in Tab. 4.3 and a 128-bit CFB deciphering mode example is illustrated in Tab. 4.4.

4.5.2 Xilinx MicroBlaze GPP with HCrypt-C Crypto-coprocessor

In contrast with the previous solution, the instruction set does not need to be customized if the HCrypt-C crypto-coprocessor is connected to the processor's register file via a dedicated coprocessor bus. Interconnection of HCrypt-C with the processor using the coprocessor bus is depicted in Fig. 4.5. The system consists in the Xilinx MicroBlaze processor, HCrypt-C and the wrapper. The MicroBlaze architecture features the high-performance 32-bit Fast Simplex Link (FSL), aimed at interfacing external modules with the processor registers. According to this standard, the FSL bus has to be routed via FIFOs. Despite the fact that the FIFOs insert additional

Table 4.3: HCrypt-C crypto-coprocessor instructions in NIOS II

A) Data transfer instructions

Instruction	Description
void W0_WR (int D0)	Send 32 most significant bits D0 to HCrypt-C deserializer
void W1_WR (int D1)	Send 32 bits D1 to HCrypt-C deserializer
void W2_WR (int D2)	Send 32 bits D2 to HCrypt-C deserializer
void W3_WR (int D3)	Send 32 least significant bits D3 to HCrypt-C deserializer
void W01_WR (int D0, int D1)	Send 64-bit upper half D0D1 to HCrypt-C deserializer
void W23_WR (int D2, int D3)	Send 64-bit lower half D2D3 to HCrypt-C deserializer
int W0_RD (void)	Read 32-bit most significant word from HCrypt-C serializer
int W1_RD (void)	Read 32-bit word from HCrypt-C serializer
int W2_RD (void)	Read 32-bit word from HCrypt-C serializer
int W3_RD (void)	Read 32-bit least significant word from HCrypt-C serializer

B) Instructions for data protection and key management

Instruction	Description
void genk (int Kx)	Generate the key and save it to session key register Kx
void getk (int Kx)	Decipher encrypted key from deserializer to session key register Kx
void putk (int Kx)	Encipher the key from session key register Kx to serializer
void enc (int Kx)	Encipher data from deserializer to serializer with ses. key in Kx

latencies, they separate the processor and HCrypt-C clock domains, so that HCrypt-C can run at a higher clock frequency than the processor. This way the processor's critical path is not extended by the critical path of HCrypt-C.

4.5.2.1 Wrapper

The wrapper interfacing HCrypt-C to the FSL buses is illustrated in the right part of Fig. 4.5. Data are moved to and from HCrypt-C wrapper using two separate 32-bit FSL buses (i.e. FSL0 and FSL1). Unfortunately, the FSL standard does not define the control interface, so before each operation, a 32-bit control instruction word has to be sent to HCrypt-C via the FSL FIFOs. Extraction of the control instruction word from the FIFO output (in FSL0) and inclusion of the 32-bit state word to the FIFO input (in FSL1) is performed by the FSL slave units. To recreate a 128-bit data word in the deserializer, four 32-bit words have to be transported via an FSL bus. In total at least five 32-bit bus transfers have to occur in order to move one 128-bit word from the MicroBlaze data register to the deserializer leading to lower performance than in the case of NIOS II. In contrast to NIOS II, only one 32-bit word can be sent to the wrapper at a time resulting in decreased performance.

Table 4.4: Simplified CFB deciphering block mode example on NIOS II

```
Blks = CTI; // Read number of data blocks

// Send encrypted session key to HCrypt-C
W01_WR(CTI0, CTI1); // Encrypted session key (high 64b)
W23_WR(CTI2, CTI3); // Encrypted session key (low 64b)

// Decipher session key and store it to session key register
GETK(0x00000000); // Decipher session key to register SR0

// Load IV from input Cipher-text FIFO
CT0 = CTI; // Load the most significant 32b word
CT1 = CTI; // Load 32b word
CT2 = CTI; // Load 32b word
CT3 = CTI; // Load the least significant 32b word

// CFB data deciphering loop
for (i=0; i<Blks; i++)
{
  // Encipher cipher-text with SR0
  W01_WR(CT0, CT1); // Send data cipher text hword to HCrypt-C
  W23_WR(CT2, CT3); // Send data cipher text lword to HCrypt-C
  ENC (0x00000000);    // Encipher cipher text using key in SR0
  DW0 = W0_RD();       // Read out results from HCrypt-C
  DW1 = W1_RD();       // Read out results from HCrypt-C
  DW2 = W2_RD();       // Read out results from HCrypt-C
  DW3 = W3_RD();       // Read out results from HCrypt-C

  // Load new cipher-text from input Cipher-text FIFO
  CT0 = CTI;
  CT1 = CTI;
  CT2 = CTI;
  CT3 = CTI;

  // Send XOR results to output Plain-text FIFO
  PTO0 = DW0 ^ CT0;
  PTO1 = DW1 ^ CT1;
  PTO2 = DW2 ^ CT2;
  PTO3 = DW3 ^ CT3;
}
```

4.5.2.2 Software

The MicroBlaze software was written in the ANSI C code using the Eclipse development environment. Contrary to the NIOS II system, no custom instructions have to be defined. The FSL bus can be accessed using only two MicroBlaze instructions: *putfslx* and *getfslx*. By using these two instructions, all the HCrypt-C commands can be described as short functions. A simplified example of the 128-bit CFB decryption block mode implemented on MicroBlaze is shown in Tab. 4.5.

Table 4.5: Simplified CFB deciphering block mode example on MicroBlaze

```
// Definition of instructions
#define iWR2DES() (0x00000000)
#define iRDSER() (0x00010000)
#define iENC(sreg) (0x00030000 + sreg)
#define iGETK(sreg) (0x00040000 + sreg)
...
// Definition of instruction function macros
#define W0123_WR(w0,w1,w2,w3) ({ \
    putfslx(iWR2DES(),0, FSL_NONBLOCKING_CONTROL); \
    putfslx(w0, 0, FSL_NONBLOCKING); putfslx(w1, 0, FSL_NONBLOCKING); \
    putfslx(w2, 0, FSL_NONBLOCKING); putfslx(w3, 0, FSL_NONBLOCKING); \
})
#define W0123_RD(w0,w1,w2,w3)({ \
    putfslx(iRDSER(), 0, FSL_NONBLOCKING_CONTROL); \
    getfslx(w0, 0, FSL_DEFAULT); getfslx(w1, 0, FSL_DEFAULT); \
    getfslx(w2, 0, FSL_DEFAULT); getfslx(w3, 0, FSL_DEFAULT); \
})
...
```

```
Blks = CTI; // Read number of data blocks

// Send encrypted session key to HCrypt-C
W0123_WR(CTI0, CTI1, CTI2, CTI3);

// Decipher session key and store it to session key register SR0
putfslx(iGETK(0), 0, FSL_NONBLOCKING_CONTROL);

// Load IV from input Cipher-text FIFO
CT0 = CTI; // Load the most significant 32b word
CT1 = CTI; // Load 32b word
CT2 = CTI; // Load 32b word
CT3 = CTI; // Load the least significant 32b word

// CFB data deciphering loop
for (i=0; i<Blks; i++)
{
  // Encipher cipher-text with SR0
  W0123_WR(CT0, CT1, CT2, CT3); // Send data cipher text to HCrypt-C
  putfslx(iENC(0), 0, FSL_NONBLOCKING_CONTROL);
  W0123_RD(&DW0, &DW1, &DW2, &DW3); // Read results from HCrypt-C

  // Load new cipher-text from input Cipher-text FIFO
  CT0 = CTI;
  CT1 = CTI;
  CT2 = CTI;
  CT3 = CTI;

  // Send XOR results to output Plain-text FIFO
  PTO0 = DW0 ^ CT0;
  PTO1 = DW1 ^ CT1;
  PTO2 = DW2 ^ CT2;
  PTO3 = DW3 ^ CT3;
}
```

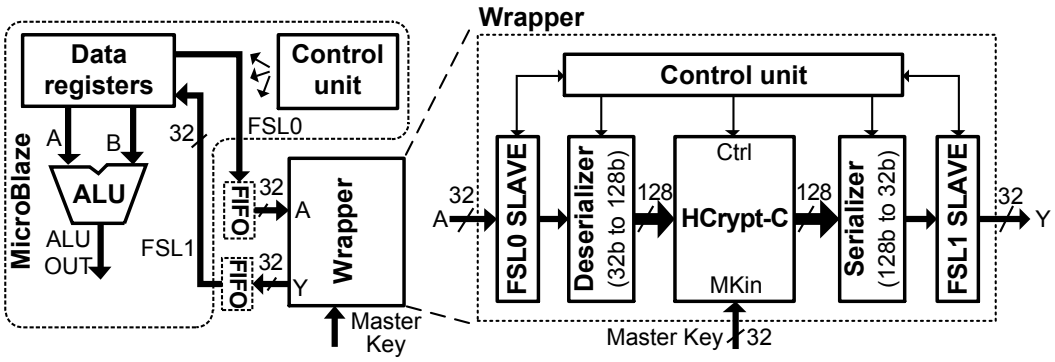

Figure 4.5: MicroBlaze interconnected to the HCrypt-C crypto-coprocessor wrapper via a dedicated processor bus (i.e. FSL)

4.5.2.3 Physical Isolation of Security Zones

According to presented separation rules higher security can be achieved if not only a physical separation at the logic placement level but also a physical isolation at the routing level is implemented. The Xilinx Virtex-6 supports the Secure Chip Crypto technology, which enables to implement the physical isolation of physical blocks on a single chip.

First, system is partitioned into the processor, cipher and key zones while each zone is compiled independently resulting in one separate netlist group per zone. Next, each netlist group is placed in a separate logic partition and each partition is mapped into a separate physical block using the Xilinx PlanAhead software. One must ensure that enough resources are present in physical blocks and it is recommended to slightly enlarge physical blocks to simplify the placement and routing. The FPGA floorplan containing all the three physical blocks is illustrated in Fig. 4.6. More detailed floorplan can be acquired using Xilinx FPGA editor.

Before concluding that the physical isolation has been achieved, a design verification has to be carried out using the Xilinx IVT.

4.5.3 ARM Cortex M1 GPP with HCrypt-C Crypto-coprocessor

The most common general solution for interfacing GPP with HCrypt-C is to access the crypto-coprocessor via a point-to-multipoint peripheral bus. This communication is less secure than the point-to-point communications mentioned before, however, it is available for all GPPs. For example, this is the only solution that can be applied in Microsemi FPGAs featuring the Cortex M1 processors and the AHB bus [105] as it is illustrated in Fig. 4.7. Like in the previous two examples, the system is divided into the protected cipher and key zones, enclosed inside HCrypt-C, and the unprotected processor zone.

Although the AHB bus does not include an instruction interface, an address bus can be used to pass instructions to HCrypt-C in parallel with data. On the other hand, the AHB bus is shared among several often accessed bus slaves (program flash

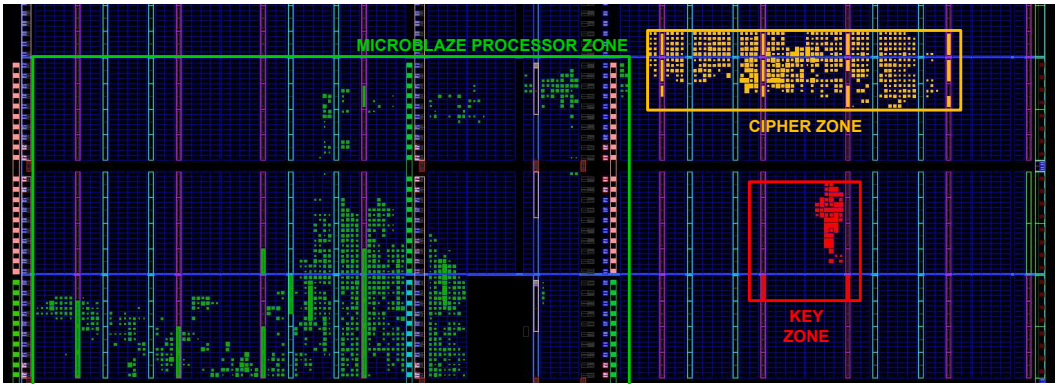

Figure 4.6: Floorplan of MicroBlaze divided into processor, cipher and key zones placed in isolated physical blocks (cipher and key zones are part of HCrypt-C)

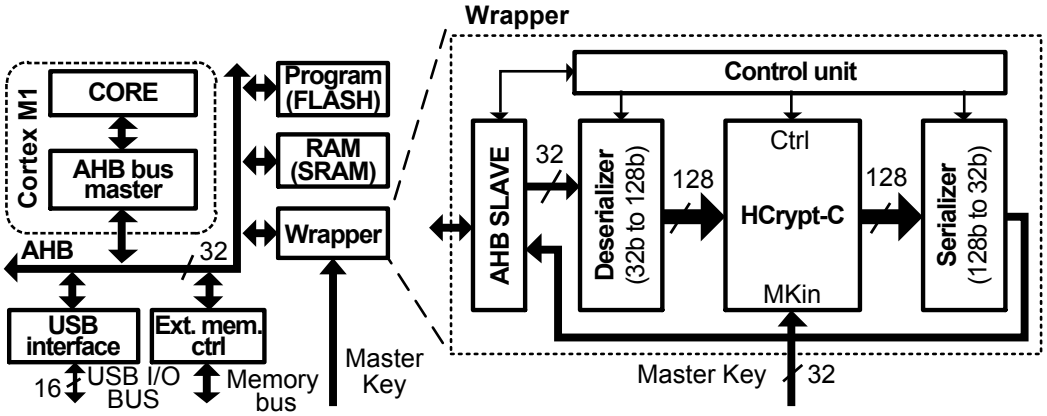

Figure 4.7: Cortex M1 interconnected to HCrypt-C wrapper via peripheral bus

memory, RAM, etc.), therefore data exchange rate with HCrypt-C is decreased.

Special constraints and separate netlists must be generated in order to keep all three security zones separate from each other. Physical isolation is not supported by Microsemi FPGAs and so routing cannot be controlled. This way only separation at the logic level is possible.

4.5.3.1 Wrapper

The wrapper interfacing the HCrypt-C crypto-coprocessor to the AHB bus is illustrated in the right part of Fig. 4.5. The HCrypt-C wrapper includes the AHB slave unit which reads out data targeted for HCrypt-C and decodes HCrypt-C instructions from the AHB address bus. This way, in contrast to the MicroBlaze system, no control words have to be send via the data bus. On the other hand the AHB bus is shared among the program flash memory, RAM memory and other peripherals leading to lower performance. Since only one 32-bit data word can be transferred into the HCrypt-C wrapper at a time, four 32-bit words in series have to be transported

to the deserializer to recreate a 128-bit data word.

4.5.3.2 Software

The Cortex M1 software was written in the ANSI C code using the Eclipse development environment. Contrary to the NIOS II system, no custom instructions have to be defined. Since the HCrypt-C wrapper is connected as a slave peripheral and instructions are moved via an address bus, every HCrypt-C instruction is viewed as one virtual register in the processor's memory space reserved for HCrypt-C. When an operation is executed, a certain virtual register has to be addressed (to pass an instruction via the address bus). For this reason all HCrypt-C instructions are described as short macros in the c code. A simplified example of the 128-bit CFB decryption block mode implemented on Cortex M1 is shown in Tab. 4.6.

4.6 Implementation Results

The three processors extended by HCrypt-C crypto-coprocessor containing the AES cipher, decipher and TRNG 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 HCrypt-C crypto-coprocessor extension were implemented in Xilinx ML605 evaluation kit featuring Virtex-6 device XC6VLX240TFF1156. For synthesis and mapping, ISE version 12.4 was used. The isolation of the three security zones was verified by Xilinx IVT and no isolation violations were reported. The Cortex M1 system and its extension were implemented in the Microsemi Fusion embedded development kit board featuring the Microsemi Fusion device M1AFS1500-FGG484. The project was compiled and mapped to the selected device using Libero version 8.5 SP2. Besides the processor and HCrypt-C crypto-coprocessor extension, a small block containing a 16-bit data interface to the external Cypress USB device CY7C68013A was embedded in all systems. It was used only for testing purposes and it does not constitute an inherent part of the system. For this reason it is not included in further resource utilization reports. Next, we present the resource utilization report and results of the hardware tests.

4.6.1 Cost Evaluation

The implementation results concerning the logic area and the memory requirements are presented in Tab. 4.7. The area is expressed in a number of occupied ALMs for the Altera family, Slices for the Xilinx family and Tiles for the Microsemi family. For comparison, we recall that one ALM in the Altera Stratix II family contains two 4-input LUTs and two Flip-Flops (FFs). One Slice in the Xilinx Virtex-6 family contains four 6-input LUTs and eight FFs. One Tile in the Microsemi Fusion family contains either one 3-input combinatorial function or one FF. Therefore, the results

Table 4.6: Simplified CFB deciphering block mode example on Cortex M1

```
// Definition of instructions
#define SM_ADDR 0x40000000
#define iWR2DES() (0x00000000)
#define iRDSER() (0x00010000)
#define iENC(sreg) (0x00030000 + sreg)
#define iGETK(sreg) (0x00040000 + sreg)
#define iSTATUS(reg)(0x00070000 + reg)
...
// Definition of instruction function macros
#define SMWR(INSTR,VALUE)(*((uint32_t volatile *)(SM_ADDR + INSTR)) = (VALUE))
#define SMRD(INSTR) (*((uint32_t volatile *)(SM_ADDR + INSTR)))
#define W0123_WR(w0,w1,w2,w3) ({ \
    SMWR(iWR2DES(),w0); SMWR(iWR2DES(),w1); \
    SMWR(iWR2DES(),w2); SMWR(iWR2DES(),w3); \
})
#define W0123_RD(w0,w1,w2,w3) ({ \
    w0 = SMRD(iRDSER()); w1 = SMRD(iRDSER()); \
    w2 = SMRD(iRDSER()); w3 = SMRD(iRDSER()); \
})
...
```

```
Blks = CTI; // Read number of data blocks

// Send encrypted session key to HCrypt-C
W0123_WR(CTI0, CTI1, CTI2, CTI3);

// Decipher session key and store it to session key register SR0
SMWR(iGETK(0), 0);

// Load IV from input Cipher-text FIFO
CT0 = CTI; // Load the most significant 32b word
CT1 = CTI; // Load 32b word
CT2 = CTI; // Load 32b word
CT3 = CTI; // Load the least significant 32b word

// CFB data deciphering loop
for (i=0; i<Blks; i++)
{
  // Encipher cipher-text with SR0
  W0123_WR(CT0, CT1, CT2, CT3); // Send data cipher text to HCrypt-C
  SMWR(iENC(0), 0);
  while(SMRD(iSTATUS(0))==0); // wait for ENC to finish
  W0123_RD(&DW0, &DW1, &DW2, &DW3); // Read results from HCrypt-C

  // Load new cipher-text from input Cipher-text FIFO
  CT0 = CTI;
  CT1 = CTI;
  CT2 = CTI;
  CT3 = CTI;

  // Send XOR results to output Plain-text FIFO
  PTO0 = DW0 ^ CT0;
  PTO1 = DW1 ^ CT1;
  PTO2 = DW2 ^ CT2;
  PTO3 = DW3 ^ CT3;
}
```

Table 4.7: Utilization of FPGA resources by tree processors with the HCrypt-C crypto-coprocessor containing the AES cipher

	NIOS II		Cortex M1		MicroBlaze	
	ALMs	**RAM kb**	**Tiles**	**RAM kb**	**Slices**	**RAM kb**
System total	2531	243.9	15053	216.0	1954	1206.0
→ **Processor**	1204	187.9	9433	104.0	1350	774.0
→ **HCrypt-C**	1327	56.0	5620	112.0	604	432.0
Ext. overhead	110.2%	29.8%	59.6%	107.7%	44.7%	55.8%

cannot be directly compared. The memory requirements are given in kbits for all technologies. For clarity, we present the results for the processor and for its HCrypt-C crypto-coprocessor extension separately.

4.6.2 Hardware Tests and Benchmarks

System functionality can be proved and real performance obtained only by hardware tests. The setup used for the hardware tests is explained in Fig. 4.8. In each test a corresponding processor with its HCrypt-C crypto-coprocessor were used. In contrast to the HCrypt hardware test, the GPP software resided either in the RAM memory filled during the FPGA configuration process (in case of NIOS II and MicroBlaze) or in the embedded Flash memory filled during the configuration process (in case of Cortex M1). The processor system together with the USB interface were configured inside an FPGA which was interconnected with the Cypress USB module. The USB module communicated with the PC containing the Cypress USB driver and a benchmark application.

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 the USB interface. The communication protocol, packet structure and testing procedure are similar to those used for the HCrypt tests (see Sec. 3.3.4). The whole testing process started when the benchmark application accessed the *Master key file* and transferred the master key to the FPGA. When the master keys were present in HCrypt-C, the PC application read the *Input packet file* and sent data packets to the GPP data input. 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 HCrypt-C. 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. The benchmark application read the resulting packet, verified it and saved it to the *Output packet file*. 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.

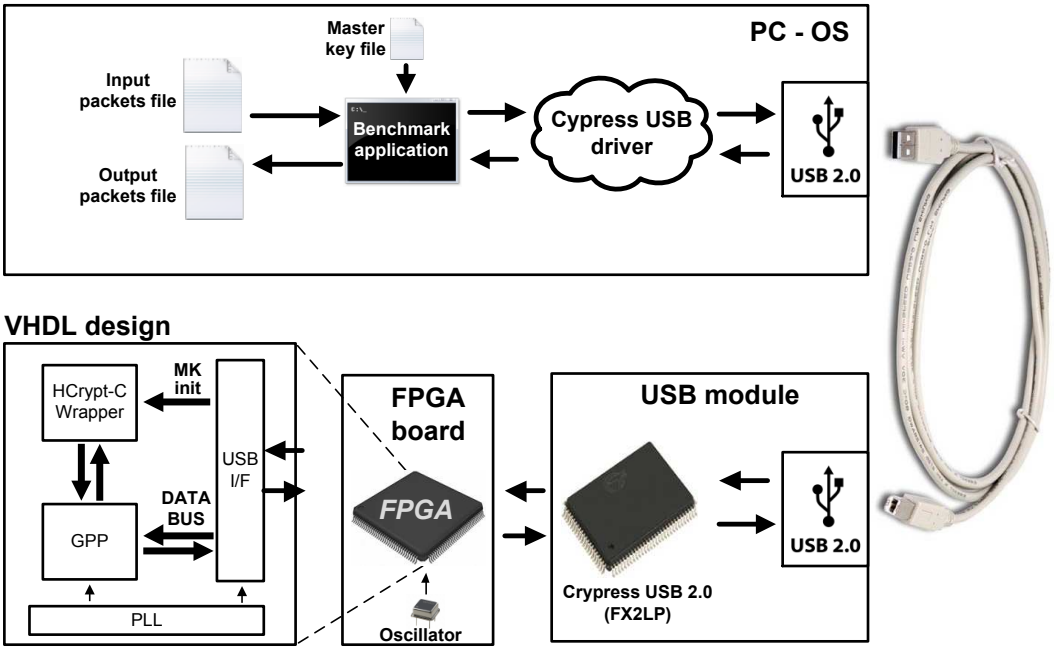

Figure 4.8: General-Purpose Processor system hardware test setup

4.7 Discussion

Area requirements for NIOS II, MicroBlaze and Cortex M1 extensions from Tab. 4.7 seem to be different. This is due to differences between ALMs, Slices and Tiles and also because of the size of processors. In the Altera FPGA, the HCrypt-C area is similar to that of the NIOS II (1327 vs. 1204 ALMs giving 110%). However, since HCrypt-C included both AES cipher and decipher cores, we can conclude that the sole security extension cost due to the separation of zones is negligible. On the other hand, the MicroBlaze processor occupies bigger area and the HCrypt-C overhead is only 45%. The separation cost remains negligible. In Cortex M1 system, HCrypt-C represents a 60% overhead, therefore its cost remains negligible.

The MicroBlaze processor with its HCrypt-C extension achieves 73% of the throughput of the NIOS II. This is caused by the FSL bus protocol (data and control words), compared to the straightforward custom instruction implementation in NIOS II. This difference could be reduced in the MicroBlaze system if data were transferred to HCrypt-C using DMA. The FSL bus would serve only for transporting instructions to HCrypt-C. Moreover, the Cortex M1 processor with its HCrypt-C 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 HCrypt-C to the AHB bus is multiplexed with the 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 HCrypt-C, sev-

eral 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 HCrypt-C unit has to be interfaced with the NIOS II processor using the 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, HCrypt-C is closely tied to the processor and thus the system maintains high performance. Connecting HCrypt-C to the MicroBlaze processor is simpler at the hardware level, but more complex at the protocol level: more complex state machine has to be implemented in the wrapper to pick up a control instruction from the data stream transported via the Fast Simplex Link. Because of this fact, the overall throughput of the system is significantly lower. Connecting HCrypt-C via a peripheral bus is the most general solution. Once the data interface is designed, the HCrypt-C can be reused with any processor featuring a common peripheral bus. However, in this case, the throughput of the system is the lowest.

Timing violation attacks that were carried out (i.e. overclocking causes timing violations in control logic) revealed a vulnerability in the CBC decryption mode (see Fig. 4.9). In fact, the Decipher data output was connected to HCrypt-C output via a multiplexer. Such architecture violated separation rules, because an intentional or unintentional fault in a multiplexer control signal could redirect session keys in clear out of the cipher zone. This backdoor was eliminated by disconnecting the Decipher output from the output of HCrypt-C. Nevertheless, the CBC decryption mode is often used nowadays and its secure implementation remains an open problem. After eliminating the backdoor, timing violation attacks against all processor implementations were not successful. The implemented protocol and architectural separations of the key and processor zones were therefore shown to be efficient. To further increase the security level, the security zones were physically isolated in the MicroBlaze case. However, the physical isolation flow is an advanced flow since very hard constraints have to be met and searching for tradeoffs requires expert knowledge. If the physical block sizes and their arrangement are not chosen well, the placement and routing phases can be computed for several hours. Despite all these obstacles, the physical isolation between the security zones was achieved and a verification certificate was generated as a proof.

The principle presented in this report can be extended to any GPP, since we showed that the HCrypt-C crypto-coprocessor can be accessed as an ordinary peripheral. However, we can expect that the best solution would be to use an open source GPP and to include HCrypt-C in the processor's data path as it was the case with the NIOS II processor.

4.8 Conclusions

In this chapter we have shown how a crypto-coprocessors supporting secure key management can be constructed. For cost-critical applications requiring also ex-

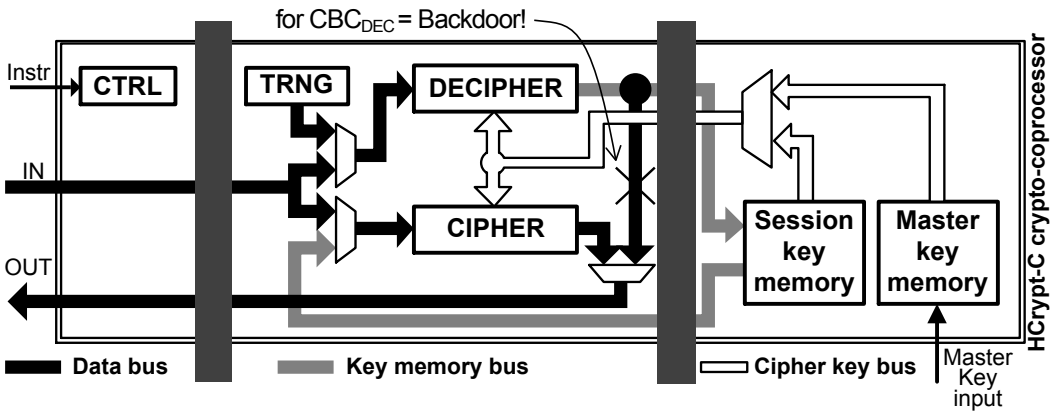

Figure 4.9: Security module with CBC decryption mode backdoor

ecution of general-purpose instructions we proposed HCrypt-C crypto-coprocessor extension to any GPP. This crypto-coprocessor respects all stringent separation rules thus enabling secure key management on GPPs. The HCrypt-C crypto-coprocessor is divided into the data, cipher and key zone each having different security privileges. This crypto-coprocessor has the 128-bit wide datapath, embedded AES cipher, decipher and TRNG. Similarly to HCrypt, master keys are transferred via a dedicated 32-bit bus. HCrypt-C can be interconnected with a processor using an internal processor bus, a dedicated coprocessor bus or a peripheral bus. All three bus types were compared and demonstrated on the three processor implementations. Altera NIOS II, Xilinx MicroBlaze and ARM Cortex M1 were extended by HCrypt-C. The operation, security and throughput of all the three processor systems were tested. The tests indicated that the data throughput of NIOS II was 25.1 Mb/s, MicroBlaze was 18.4 Mb/s and Cortex M1 was 12.2 Mb/s, while each system exchanged session keys and processed data in packets using the 128-bit CFB block encryption mode.

Flexibility of the crypto-coprocessor can be significantly increased if some of its parts are upgradeable (i.e. cipher, TRNG, etc.). The solution to this issue is a partial reconfiguration technology. This technology allows to reconfigure the whole cipher zone. However, with the implementation of the partial reconfiguration new security challenges arise. These challenges will be further examined and solutions will be proposed in Chap. 6.

The presented processors with the HCrypt-C crypto-coprocessor extension were published in [81], [82], [83] and [106].

Chapter 5

Protecting Crypto-processors Against SCA at Macroarchitecture Level

Contents

Most of contemporary cryptographic algorithms are robust against statistical cryptanalyses (i.e. linear, differential) as well as algebraic cryptanalyses. However, problems can arise when these algorithms are implemented in physical devices. Essentially, physical processes related to computations of cryptographic algorithms within the device can be partly observed from outside the device and can be considered as a valuable leaking information. New methods have to be constantly developed to suppress the impact of these physical phenomena and thus to protect the valuable information stored and processed inside the cryptographic devices.

This chapter will first explain what are side-channel attacks and what work has been done so far to protect the cryptographic devices from them. Next, a detailed security analysis of the AES cipher will be provided. The analysis of the AES cipher is essential for the HCrypt security analysis. Contrary to the threat model in previous chapters, the HCrypt security analysis will be focused not only on the software attacks, but also on the side-channel and fault injection attacks. Considering these analyses, we will propose the novel zero-cost countermeasures against SCA for crypto-processors and crypto-coprocessors. The zero-cost countermeasures will be included in an extended version of separation rules and further demonstrated on the HCrypt2 implementation.

5.1 Side-Channel Attacks

Side-channel attacks are passive non-invasive physical attacks on physical cryptographic systems which exploit different information leakage sources with the aim to disclose the secret keys processed within the cryptographic device. The leaked information is directly related to changes of physical quantities like time, power, intensity of electromagnetic field, light intensity, etc. Therefore the sources of information leakage are mainly:

- computational time of a cryptographic algorithm,
- power consumption,
- electromagnetic radiation,
- acoustic vibrations,
- light emission

Side-channel attacks were not given much attention until 1996 when Kocher et al. first published *timing attacks* [56]. Authors suggested to measure the computation time of cryptographic algorithms in order to gain necessary information about the secret keys that were involved in the computation. Another breakthrough came in 1998 when Kocher et al. proposed to use the dynamic power consumption of the electronic circuit as a side-channel for attacking cryptographic device [107]. Dynamic power consumption of a CMOS cell depends on its input and output state transitions ($0 \rightarrow 1$, $1 \rightarrow 0$). Two power analysis methods were presented: *Simple Power Analysis* (SPA) and *Differential Power Analysis* (DPA) [107]. Electromagnetic analysis attacks (EMA) are very similar to power attacks. EMA attacks were first demonstrated in 2001 by Gandolfi et al. [108] and Quisquater et al. [109]. The sources of information leakage are the dynamic changes of electromagnetic field related to changes of current passing through a CMOS cell during transitions. Contrary to the power analysis attacks, an antenna moving over the circuit's surface allows to precisely select the area where the most information is leaked from the circuit.

5.1.1 Power Analysis Attacks

The discovery of the power analysis attacks led to numerous successful attempts to reveal secret information from various kinds of security devices. Smart card security breaches were demonstrated in [110], [111], [112], etc. The first successful attack against the ASIC hardware implementation of AES was performed by Ors et al. [113]. In 2003 Standaert et al. have warned that cryptographic implementation in FPGAs can be also attacked [114]. Soon after, in 2004, his team has demonstrated that DES and AES implementations in FPGAs are also easily breakable [115], [116]. Since then, many attack improvements have been proposed.

The power analysis attack begins with the power consumption measurement. The most common way is to connect a shunt resistor to power or ground lines of the measured circuit and to record the voltage drop along the resistor by a digital oscilloscope. The power traces recorded by the oscilloscope are subsequently processed on the PC.

In addition to recorded traces containing the key information leakage, the attacker must select an appropriate power leakage model that involves key hypotheses. The correct key hypothesis can be recognized from the power traces using a distinguisher.

5.1.1.1 Leakage Models

The second phase of the attack is to describe transitions inside the circuit by an appropriate leakage model. The leakage model serves to estimate the change of power consumption of the device according to expected data-dependent transitions within the circuit. The estimated power consumption can be compared to the measured power consumption. The attack can be successful only if the similarity is high enough. Several leakage models exist and the most common are the Hamming Weight (HW) and the Hamming Distance (HD) models.

Hamming weight power model

The simplest leakage model is the *Hamming weight model*. This model assumes that the power consumption is proportional to the number of register bits set to logic one. For example if an 8-bit register is set to the state $(1100\ 0101)_2$, the corresponding Hamming weight is 4. However, the HW model is not sufficiently precise, because the power consumption depends more on number of bit transitions than the actual state. Therefore, the attacker will probably use the HW model if only one of two consecutive register states is known. If both consecutive register states are known, the attacker can use more accurately the Hamming distance power model.

Hamming distance power model

This model exploits bit transitions causing dynamic power consumption. For its higher accuracy it is more preferred than the HW model. The HD of two consecutive states a and b can be described as

$$HD\,(a, b) = HW\,(a \oplus b) \tag{5.1}$$

where $\oplus$ denotes bit-wise exclusive-or operation. HD can be also understood as a number of bit transitions. For instance if the state before the transition a = $(1100\ 0101)_2$ and the state after the transition b = $(0101\ 1111)_2$ then the corresponding HD equals to $HW\,[(1001\ 1010)_2] = 4$.

5.1.1.2 Distinguishers

The last phase of the attack is to find the dependency between the leakage model and information contained in the measured power traces. This operation is performed by algorithms called distinguishers. The role of distinguishers is to distinguish the correct key hypothesis from all other incorrect ones. The strength of the particular distinguisher is in its ability to find the correct key hypothesis using as few power traces as possible. The first proposed distinguishers were the *Simple Power Analysis* and *Differential Power Analysis* described by Kocher et al. in [107].

The success of power analysis attacks increased due to new improvements. Generalized multiple-bit DPA was proposed by Messerges et al. [117], Template Attacks by Chari et al. [118], Correlation Power Analysis (CPA) by Brier et al. [119], Partition Power Analysis by Le et al. [120], Mutual Information Analysis by Gierlichs et al. [121] and more.

5.1.2 Countermeasures

The discovery of DPA quickly uncovered vulnerabilities of nearly all cryptographic devices of that time and led to desperate research effort, both academic and industrial, to protect the cryptographic systems. This was the time when the first countermeasures against power analysis attacks came to light. Today many countermeasures have been presented reaching from a simple key update to complex dual-rail pre-charge logic. A very good studies comparing some countermeasures were presented by Mangard [122] and Güneysu et al. in [123]. Next, we present the most common countermeasures.

5.1.2.1 Key Update

The attack can be successful only if the attacker can collect enough power traces corresponding to a certain number of encipherings using the same secret key. However, if the key is updated often enough, the attacker may not be able to acquire a sufficient number of power traces. The solution is to use an ephemeral secret key (i.e. session key), which is regenerated after its expiration time. On the other hand, this countermeasure requires other means for distribution of the new generated session key. Both the symmetric-key and asymmetric-key cryptographic algorithms can be used for this purpose. The symmetric-key algorithms require another higher-level secret key (i.e. master key) to secure the distribution of the session keys. The asymmetric-key algorithms require a pair of keys (i.e. private and public keys) to protect session key exchange. In both cases, it is not worth for the attacker to attack the ephemeral session keys, but rather to concentrate his effort on disclosing the master or private key. To thwart attacks on these highly confidential keys, other countermeasures must be applied in the device. The key update countermeasures were considered in [124], [125].

5.1.2.2 Generation of Noise

The objective of using noise generators is to introduce data independent noise to power consumption. Additional noise reduces the signal to noise ratio and forces the attacker to collect more power traces. One principle suggests to configure all the unused FPGA logic blocks to shift registers. Constant shifting will increase power consumption and thus the noise level. Another way is to exploit write collisions of embedded TDPRAMs [126]. A high consumption can be caused also by short circuits in FPGA switch boxes [127]. However, it has been shown that efficiency of the noise generators is very small [123]. Depending on the applied principle, the noise generators can be very expensive in terms of circuit resources. For their low effectiveness and high price, they are not recommended.

5.1.2.3 Random Execution Length

An interesting approach for countering side-channel attacks is to insert random dummy cycles during the execution of a cryptographic algorithm. This way, the attacker must use more complex filtering to remove all misalignments caused by randomization. However, practical attacks exist [128], [129]. Random length of the execution can also be achieved if the clock period is not constant. This interesting approach was proposed and tested in [123]. Author suggests to exploit features of modern digital clock management units like clock multiplexing and multi-phase shifting. Although these approaches may not significantly increase number of required resources, the extra penalty is the speed decrease.

5.1.2.4 Data Masking

Masking is probably the most common countermeasure in current devices. The aim of the masking is to insert randomness into security-critical computations in such a way that data are changed and so the leaked information will completely disappear in the obtained power traces. Masking can be performed on the bit level [130] and word level. However, masking on the word level is vulnerable to high-order power analysis attacks [117]. Masking involves application of a mask m to data word a using a masking operator (i.e. modular addition, modular multiplication, exclusive or, etc.). For instance, if the masking operator is the exclusive or, the masked data has the form $a_m = a \oplus m$. Only the masked data can be processed together with the secret key. Before the output data is ready, it has to be unmasked. Masking can also involve a random pre-charge phase. This phase involves initialization of all registers in the circuit with random values before and after the data word a was processed. This interesting approach prevents the attacker to use the Hamming distance model, since one of the two consecutive states is random. However, a new mask has to be used per every data word in order to prevent second order power analysis attacks. This can be an issue, since random number generators are usually not able to generate truly random numbers with throughput higher than

1-2 Mb/s. Despite its high effectiveness, masking is usually very expensive and reduces significantly the performance of the cipher [131], [62].

5.1.2.5 Data hiding

Unlike the masking, the data hiding technique does not change the data. The main idea of data hiding is to make the power consumption as constant as possible for different input data. This can be achieved with the so called Dual-Rail Pre-charge logic (DRP). The DRP replaces classic single-rail cells by dual-rail cells, where each input of the dual-rail cell is composed of two complementary wires (carrying data value by one wire and its inverse value by the other wire). The DRP cell output is also composed of two complementary wires carrying an output value and inverted output value. In addition to dual-rail cells, complementary wires have to be implemented as symmetrically as possible in the circuit to provide constant power consumption independent from the processed data [132], [133], [134]. Implementation of symmetrical routes is not straightforward in ASICs, but almost impossible in FPGAs. Moreover, if a pre-charge phase is implemented, each time the circuit is switched into the pre-charge phase, both wires are initialized either with one or zero value (no complementarity during the pre-charge).

The DRP techniques were studied in [135], [136]. Other variants including masking, three phase dual-rail, balanced cell DRP, etc. were proposed in [130], [134], [137]. Although very resistant to power analysis attacks, the DRP increases the implementation of the cipher to at least a double size. When masking is implemented together with DPR, the overhead can reach 450% [138]. Because of the speed and size penalties, the DRP remains very expensive for chip manufacturers, so its cost-effectiveness is considerably low.

We have shown that none of the today's countermeasures is perfect and tradeoffs between the security, size, speed and power consumption have to be always searched to achieve sufficient cost-effectiveness. All presented countermeasures try to hide key information in the power leakage. However, the attacker needs to construct also a good power leakage model to be able to distinguish the correct key hypothesis with the same success rate. Interestingly, no countermeasure has been proposed that would prevent the attacker from constructing an effective power leakage model up to now. We believe that there are ways to reorganize the architecture in a such way that the attacker would need more power traces to recover the secret key with the same success rate. Next, we explain these countermeasures.

5.2 Crypto-processor with Zero-cost Countermeasures against SCA

5.2.1 Introduction

The first version of the HCrypt crypto-processor [80] (it will be called HCrypt1 throughout this chapter) is presented in Chap. 3. The main feature of the HCrypt1

crypto-processor obtained at the macroarchitecture level was the resistance against software attacks targeting disclosure of secret keys. It was supposed that the countermeasures against hardware attacks such as side-channel attacks and Fault Injection Attacks (FIA) could be implemented independently from the HCrypt1 macroarchitecture inside embedded cipher and decipher blocks at the microarchitecture level. Since both cipher and decipher processed session keys using the master key, it was necessary to protect both cipher and decipher against SCA and FIA. It turned out that efficient countermeasures could be the main obstacle for protection of the crypto-processor against hardware attacks.

The aim of the work described in this chapter is to propose a new evolution of the HCrypt1 macroarchitecture maintaining high level protection against software attacks and increasing robustness against SCA and FIA. One of the main advantages of the proposed architecture illustrated on the new HCrypt2 crypto-processor is that besides high macroarchitecture level protection against software and side-channel attacks, it simplifies implementation of common data hiding and data masking techniques as protection against SCA at microarchitecture level.

Unfortunately, HCrypt1 security does not depend only on its macroarchitecture. The SCA exploits physical phenomena related to HCrypt microarchitectural properties. However, the internal structure of the cipher and decipher units must be considered when analysing HCrypt microarchitectural properties. For this reason, the cipher and decipher cannot be considered as black-box units any longer, and so their ciphering and deciphering algorithms must be selected. For the sake of simplicity, we chose the AES standard [8], and so the AES cipher and AES decipher will be considered further on in this chapter.

5.2.1.1 Evaluation of the AES Core Security Against SCA

The aim of the security evaluation is to determine how difficult it is for the attacker to recover the secret key used for the enciphering. We assume that the attacker does not have any knowledge about the transistor/cell netlist of the AES cipher. On the other hand, we suppose that the attacker knows the cryptographic algorithm to be attacked and its implementation.

The goal of the attacker is to obtain the secret key. For this reason, he will try to exploit data-dependent power consumption. In order to construct the data-dependent power consumption hypotheses, key-dependent intermediate results are needed. The most common technique for the construction of hypotheses, considering the data-dependent power consumption, is to use HW or HD power models.

Furthermore, we assume that the attacker has an access only to a limited amount of the leaked information. Thus, the adversary is not allowed to fix some bytes to simplify the cryptanalysis, and no viable methods can be used to decrease the complexity of the hypotheses creation.

Finally, we suppose that the adversary will attack the AES cipher implemented in FPGA, and that S-boxes are implemented using embedded RAM blocks. We will suppose that the adversary can choose either HW or HD power model, but he prefers

HD power model, because it fits the real power consumption better. Let us assume that the attacker can attack either the first or the last AES round key. In each case, we will analyze the complexities of the HD and HW power model construction.

The further security evaluation of the AES cipher and decipher units requires a short description of both architectures.

AES cipher architecture

The architecture of the AES cipher is illustrated in Fig. 5.1. High throughput is provided by a 128-bit wide datapath. The folded structure of the datapath performs one enciphering in 11 clock cycles. 16 parallel 8-bit substitution tables (S-boxes) are implemented in 8 TDPRAM blocks. These embedded RAMs are synchronous, and so their internal registers serve also to register round state value. ShiftRows operation is implemented solely by routing resources. Four parallel 32-bit MixColumns units are implemented in logic. Input plain-texts and output plain-texts are registered using DFF. Round keys are generated by the Key expansion unit, which has also 128-bit folded datapath and computes one round key per one clock cycle.

Let us denote ShiftRows (SR) and InvShiftRows (SR^{-1}) operations applied to the byte i, j of the AES state as

$$SR\,(i,j) = (i, j - i\,(mod\ 4)) \tag{5.2}$$

$$SR^{-1}\,(i,j) = (i, j + i\,(mod\ 4)) \tag{5.3}$$

Let the input data register contains the m-th plaintext block $pt^{(m)}$. This block is XORed with the round key $k^{(0)}$ producing the state $x^{(0)} = pt^{(m)} \oplus k^{(0)}$. Since the round counter is initialized to zero ($r = 0$), the sum propagates to the S-box input. The SubBytes operation (SB) is carried out at the beginning of the next clock cycle ($r = 1$), producing $SB(x^{(0)})$. Subsequently, after ShiftRows, the MixColumns operation (MC) is applied. The output of the MixColumns $w^{(r)} = MC\left(SR\left(S(pt^{(m)} \oplus k^{(0)})\right)\right)$ is selected by the multiplexer (since $r = 1$), XORed with the round key $k^{(1)}$, producing a new state $x^{(1)} = w^{(1)} \oplus k^{(1)}$. The state $x^{(1)}$ propagates to the S-box input, and the SubBytes operation is applied again in the subsequent clock cycle, etc.

In the last round ($r = 10$), the cipher-text can be described as

$$ct^{(m)} = x^{(10)} = w^{(10)} \oplus k^{(10)} = SR\left(SB(x^{(9)} \oplus k^{(9)})\right) \oplus k^{(10)} \tag{5.4}$$

where $ct^{(m)}$ is registered in the output data register (the MixColumns function is not performed in the last round). At the same time the round counter resets to $r = 0$, and the input register outputs the next plaintext block $pt^{(m+1)}$. Input and output registers are enabled only during the last clock cycle ($r = 10$).

Known cipher output cipher-text attack

We recall that the attacker needs to know two consecutive values of certain register output in order to construct usable HD power hypotheses. Let us assume that

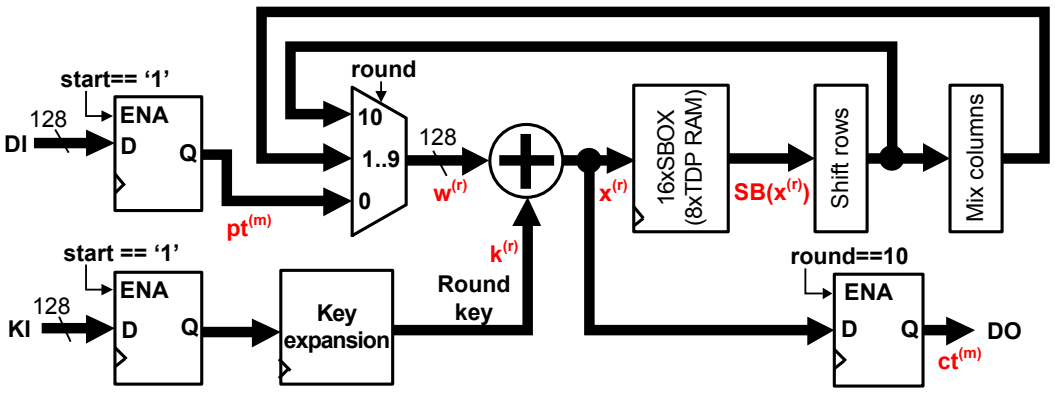

Figure 5.1: Architecture of the AES cipher with the 128-bit folded datapath

the attacker has access only to the cipher output (cipher-texts). The best register candidates are the internal registers of the embedded RAMs implementing S-boxes. The RAM registers are used to register the RAM inputs (control, address and data) before being used to access the memory matrix. These registers are the best source of leakage, because they are all producing the key-dependent dynamic power consumption triggered by the clock rising edge. Since S-box RAM input $x^{(r)}$ is mirrored to the output of its internal registers after the clock rising edge, we have to construct the hypotheses for the S-box input $x^{(r)}$. The key expansion will not be considered in this analysis.

In order to construct the HD power model, we need to find two consecutive values of the S-box register. The value of the S-box input $x^{(10)}$ in the last round ($r = 10$) corresponds to the cipher-text $ct^{(m)}$ that will be registered by the output register at the end of this round. However, this cipher-text $ct^{(m)}$ will be also registered by the embedded RAM (further referenced as S-box). The next step is to track a value that was stored in the S-box register before the $ct^{(m)}$ value was registered. For tracking the value, some part of the key has to be guessed. In fact, the 32-bit MixColumns operation is not involved in the last round, thus the narrowest is the S-box operation and it is performed on 8 bits. For this reason, only 2^8 key hypotheses $k_{i,j}^{(10)}$ are required and the attacker needs to distinguish the correct hypothesis from the 255 incorrect hypotheses only. The 8 bits of the previous value stored in the S-box register are

$$x_{i,j}^{(9)} = SB^{-1}\left(k_{i,j}^{(10)} \oplus ct_{i,j}^{(m)}\right) \tag{5.5}$$

Having both the actual and previous S-box register values, we can construct the Hamming distance power model:

$$H_{k_{i,j}^{(10)}} = HD\left(SB^{-1}\left(k_{i,j}^{(10)} \oplus ct_{i,j}^{(m)}\right), ct_{SR^{-1}(i,j)}^{(m)}\right) \tag{5.6}$$

Other alternative is to construct a Hamming weight power model. Although easier to construct, the 8-bit key hypothesis is required, its precision is much lower

and so much more traces are required to achieve the same success rate as with the Hamming distance power model. The HW power model requires only one state value and so Eq. 5.5 can be used. The HW power model is

$$H_{k_{i,j}^{(10)}} = HW\left(x_{i,j}^{(9)}\right) = HW\left(SB^{-1}\left(k_{i,j}^{(10)} \oplus ct_{i,j}^{(m)}\right)\right) \tag{5.7}$$

We can see that only 2^8 key hypotheses are required for construction of the HD power model.

Next, we show that it is much more difficult to attack the cipher if only input is known.

Known cipher input plain-text attack

In some cases, the attacker does not have a direct access to the cipher output cipher-text and he has to use the cipher input data to recover the secret key. To construct the HD power model, two consecutive register states have to be found. Like in the previous scenario, the S-box input register is suitable for the attack. After starting the enciphering, a plain-text $pt^{(m)}$ is XORed with the $k^{(0)}$ producing the state value $x^{(0)}$. In the next round ($r = 1$), the state value is substituted by the S-box operation, the ShiftRows operation is applied, transformed by the MixColumns operation (applied on 32-bit words) and XORed with the next round key $k^{(1)}$ producing the new state value $x^{(1)}$. This way we can find two consecutive states bytes $x^{(0)}$ and $x^{(1)}$ stored by the S-box registers. However, the attack is not as simple as in the previous case. The computation of at least one byte of the $x^{(1)}$ value necessitates involvement of the 32-bit MixColumns operation and the addition of the round key (for $r = 1$). Thus, at least 32 bits of the round key $k^{(0)}$ have to be guessed to calculate the 32-bit MixColumns operation, and also at least 8 bits of the round key $k^{(1)}$ have to be guessed to obtain a one byte of the state value $x^{(1)}$. Before the HD power model can be constructed, we need to find the equation for the 32-bit output word $w_{*,j}^{(1)}$ of the $j-th$ MixColumns block

$$w_{*,j}^{(1)} = MC\left(\underset{0\leq i\leq 3}{SB}\left(pt_{SR^{-1}(i,j)}^{(m)} \oplus k_{SR^{-1}(i,j)}^{(0)}\right)\right), \tag{5.8}$$

where the asterisk symbol denotes all values for i. The last step is to select one byte $(i-th)$ out of the 32-bit word $w_{*,j}^{(1)}$ and determine a byte of the next state value as $x_{i,j}^{(1)} = w_{i,j}^{(1)} \oplus k_{i,j}^{(1)}$. The computation of the $i-th$ byte of the new state value requires an 8-bit round key hypothesis in round $r = 1$. The resulting HD power model is

$$H_{k_{SR^{-1}(*,j)}^{(0)},k_{i,j}^{(1)}} = HD\left(w_{i,j}^{(1)} \oplus k_{i,j}^{(1)}, pt_{SR^{-1}(i,j)}^{(m)} \oplus k_{SR^{-1}(i,j)}^{(0)}\right), \tag{5.9}$$

and requires guessing four bytes of the round key $k^{(0)}$ in round $r = 0$ and one byte of the round key $k^{(1)}$ in round $r = 1$. Thus the complexity of the HD power model necessitates at least 2^{40} key hypotheses and the attacker have to distinguish the correct hypothesis from the remaining $2^{40} - 1$ incorrect ones. Naturally, much more

power traces are required to distinguish the correct key hypothesis with the same success rate than in the known cipher output case.

Similarly to the previous case, the HW power model can be constructed. Although only the 8-bit key hypothesis is required, the HW power model is not as precise as the HD power model and so much more power traces are required to achieve the same success rate. The HW power model requires only one state value (i.e. $x^{(0)}$). The HW power model for i, j byte of the state value is

$$H_{k_{i,j}^{(0)}} = HW\left(x_{i,j}^{(0)}\right) = HW\left(k_{i,j}^{(0)} \oplus pt_{i,j}^{(m)}\right) \tag{5.10}$$

One way to prevent the attacker from constructing the HD power model is to insert a random pre-charge phase, so that a random value is inserted between two consecutive state values. However, this approach necessitates additional resources and the cipher throughput is significantly lower. Interestingly, hiding the cipher output data from the attacker can be seen as some kind of countermeasure (i.e. when processing a session key using a master key), since much more plain-texts have to be ciphered with the same secret key. Moreover, this countermeasure requires no additional resources and the throughput is maintained, thus it can be considered a zero-cost countermeasure.

AES decipher architecture

The architecture of the AES decipher is shown in Fig. 5.2. Similarly to the AES cipher architecture, the AES decipher features the 128-bit folded datapath capable of deciphering one 128-bit cipher-text in 11 clock cycles. 16 parallel 8-bit inverse substitution tables (InvS-boxes) are implemented in 8 TDPRAM blocks. Registration of the round state is carried out by InvS-box embedded synchronous RAMs. InvShiftRows operation is implemented solely by the routing resources. Four parallel InvMixColumns units are implemented in logic. Input cipher-texts and output plain-texts are registered using DFFs. Round keys are generated by the Key expansion unit, which has also 128-bit folded datapath and computes one round key per one clock cycle.

Let the input data register contains the m-th cipher-text block $ct^{(m)}$. This block is XORed with the round key $k^{(10)}$ (round keys are in reversed order) producing the state $x^{(0)} = pt^{(m)} \oplus k^{(10)}$. Since the round counter is initialized to zero ($r = 0$), the sum propagates to the InvS-box input. The InvSubBytes operation (SB^{-1}) is carried out at the beginning of the next clock cycle ($r = 1$), producing $SB^{-1}(x^{(0)})$. Subsequently, after InvShiftRows (SR^{-1}) a round key $k^{(9)}$ is added producing the $w^{(1)} = SR^{-1}\left(SB^{-1}\left(ct^{(m)} \oplus k^{(10)}\right)\right) \oplus k^{(9)}$. Finally, the InvMixColumns transformation (MC^{-1}) is applied, its output is selected by the multiplexer (since $r = 1$) and the new state value $x^{(1)} = MC^{-1}\left(w^{(1)}\right)$ is calculated. The state $x^{(1)}$ propagates to the InvS-box input, and the InvSubBytes operation is applied again in the subsequent clock cycle, etc.

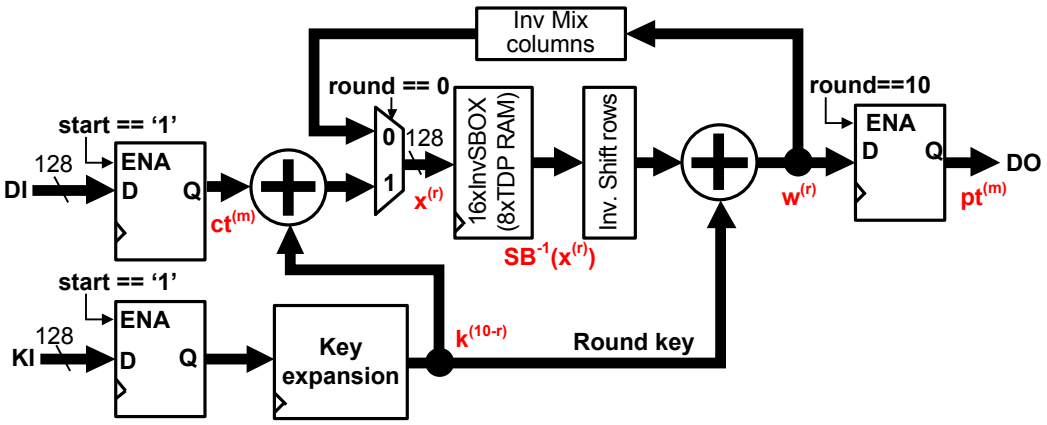

Figure 5.2: Architecture of the AES decipher with the 128-bit folded datapath

In the last round ($r = 10$), the resulting plain-text can be described as

$$pt^{(m)} = w^{(10)} = SR^{-1}\left(SB^{-1}(x^{(9)})\right) \oplus k^{(0)} = SR^{-1}\left(SB^{-1}\left(MC^{-1}\left(w^{(9)}\right)\right)\right) \oplus k^{(0)} \tag{5.11}$$

where $pt^{(m)}$ is registered in the output data register. At the same time the round counter resets to $r = 0$, and the input register outputs the next cipher-text block $ct^{(m+1)}$. Input and output registers are enabled only during the last clock cycle ($r = 10$).

Known decipher output plain-text attack

Let us assume that the attacker has access only to the decipher output (plain-texts). Similarly to the cipher implementation, the best register candidates are the internal registers of the embedded RAMs implementing InvS-boxes (further just InvS-box). Since InvS-box input $x^{(r)}$ is mirrored to the output of its internal registers after the clock rising edge, we have to construct the hypotheses for the InvS-box input $x^{(r)}$. The key expansion will not be considered in this analysis.

In order to construct the HD power model, two consecutive values of the InvS-box register have to be found. The value at the InvS-box input $x^{(10)}$ in the last round ($r = 10$) corresponds to $MC^{-1}\left(pt^{(m)}\right)$. At the end of the $10 - th$ round the $w^{(10)}$ value will be registered by the output data register as the plain-text $pt^{(m)}$ and at the same time the value $x^{(10)}$ will be registered by the InvS-box internal register. The construction of the HD power model requires also computation of one byte of the state $x^{(9)}$ using the equation

$$x^{(9)}_{SR(i,j)} = SB\left(w^{(10)}_{i,j} \oplus k^{(0)}_{i,j}\right) = SB\left(pt^{(m)}_{i,j} \oplus k^{(0)}_{i,j}\right) \tag{5.12}$$

Since the whole 128-bit word of the $x^{(10)}$ can be precomputed using the equation

$$x^{(10)} = MC^{-1}\left(pt^{(m)}\right) \tag{5.13}$$

the HD power model for the decipher output can be constructed as follows

$$H_{k_{i,j}^{(0)}} = HD\left(x_{SR(i,j)}^{(10)}, SB\left(pt_{i,j}^{(m)} \oplus k_{i,j}^{(0)}\right)\right) \tag{5.14}$$

Similarly the construction of the HD model requires 256 key hypotheses. Accordingly the Hamming weight power model can be constructed for the InvS-box input using $8 - bit$ key hypotheses as

$$H_{k_{i,j}^{(0)}} = HW\left(x_{SR(i,j)}^{(9)}\right) = HW\left(SB\left(pt_{i,j}^{(m)} \oplus k_{i,j}^{(0)}\right)\right) \tag{5.15}$$

Next, we show that it is much more difficult to attack the decipher if only the input is known.

Known decipher input cipher-text attack

If we assume that the attacker has access only to the decipher input (cipher-texts) he would probably try to construct the HD power model for the InvS-box input. Unlike the known plain-text case, it is much more complex to calculate the two consecutive state values. The state value during the $0 - th$ round ($r = 0$) requires to guess certain number of key bits because of initial key whitening. After propagating through InvS-boxes and InvShiftRows, another key bits have to be guessed before the InvMixColumns transformation can be applied in the first round ($r = 1$). Consequently, more than 40 key bits have to be guessed and so not only more power traces are required, but complex computations make attack time-consuming or even infeasible.

However, the attack could be still feasible if a sequence of cipher-texts is considered in HD power model and decipher control can be reset. First the plain-text $ct^{(m)}$ is registered by the input data register. Subsequently the key $k^{(10)}$ is XORed with the cipher-text producing the state $x^{(0)}$ in the round $r = 0$. This state value is registered at the next rising edge by the InvS-box input registers. This is the moment when the attacker has to reset the cipher's round counter to $r = 0$ and input data register has to output the next cipher-text $ct^{(m+1)}$. This way the new cipher-text will be XORed with the same key $k^{(10)}$. The produced state value will overwrite the previous value stored in the InvS-box input register. This change can be exploited to construct the HD power model. Naturally, only one byte of the secret key has to be guessed, since the same key is used with the both cipher-texts. The HD power model involving an 8-bit key hypothesis is as follows

$$H_{k_{i,j}^{(10)}} = HD\left(SB^{-1}\left(ct_{i,j}^{(m)} \oplus k_{i,j}^{(10)}\right), SB^{-1}\left(ct_{i,j}^{(m+1)} \oplus k_{i,j}^{(10)}\right)\right) \tag{5.16}$$

The Hamming weight power model does not require resetting the round counter since only one state value is required for one cipher-text. The HW power model is as follows

$$H_{k_{i,j}^{(10)}} = HW\left(ct_{i,j}^{(m)} \oplus k_{i,j}^{(10)}\right) \tag{5.17}$$

Table 5.1: Power analysis attacks on an AES cipher and decipher

	Attack side	**Attack requirements**
SCA1 (ciph.)	Plain-Text input	HD model, 2^{40} key hypotheses at round 1 for each S-Box
SCA2 (ciph.)	Cipher-Text output	HD model, 2^8 key hypotheses at round 10 for each S-Box
SCA3 (deciph.)	Cipher-Text input	HD model, $> 2^{40}$ key hypotheses at round 1 for each S-Box
SCA4 (deciph.)	Cipher-Text input	HD model, 2^8 key hypotheses at round 1 for each S-Box + round counter reset at round 0
SCA5 (deciph.)	Plain-Text output	HD model, 2^8 key hypotheses at round 10 for each S-Box

To sum up, the Hamming distance power model is very difficult to construct, because more than 40 bits of the key have to be guessed. On the other hand, the Hamming weight power model is feasible requiring 256 key hypothesis only. If an option to reset the cipher exists, the HD power model can be substantially simplified to require only 8-bit key hypotheses. The complexity of HD model creation for the AES cipher and decipher are summarized in Tab. 5.1.

The attacks can be prevented using countermeasures. However it is always more expensive to protect the AES decipher using additional countermeasures (i.e. DRP) than the AES cipher, because the implementation of the InvMixColumns operation requires more logic resources than the MixColumns operation [85]. For this reason, the AES cipher with countermeasures is preferred in cost-sensitive applications.

5.2.1.2 Evaluation of HCrypt1 Security Against SCA and FIA

According to separation rules presented in Sec. 3.2, the secret keys are well protected against software attacks if no direct physical path exists between the key memory and the data registers. This essential condition is met very well in HCrypt1, since data buses never cross the cipher-key zone barrier and the key buses never cross data-cipher zone barrier. Thus, the only way for session keys to get to data registers is by passing via the cipher (being enciphered with the master key).

Unfortunately, the security evaluation of HCrypt1 is less positive when examining its robustness against SCA attacks. The master key is used by both cipher and decipher and so both of them should be considered in the security analysis. The security analysis, presented in the previous section, is essential for the evaluation of the HCrypt1 security against physical attacks. We suppose that the attacker not only can passively measure the power consumption of the digital circuit, but also can choose data that enter the circuit. Furthermore, we suppose that the attacker has the ability to manipulate with the control logic (inject faults to the control unit, change the multiplexer select state, etc.). The objective of this work is to find a way to protect the secret keys stored in the crypto-processor from their disclosure despite the fact that the attacker has all mentioned capabilities.

Table 5.2: Possibilities of physical attacks on the HCrypt1 crypto-processor

Operation type	Equation	Potential attack	Target	#
Session key generation	$SK = C_{MK}^{-1}(TRNG)$	FIA on M1 + SCA1	MK	①
Session key transmission	$OUT = C_{MK}(SK)$	FIA on M2 + SCA2	MK	②
Session key reception	$SK = C_{MK}^{-1}(IN)$	SCA1	MK	③
Data enciphering	$OUT = C_{SK}(IN)$	DFA	SK	④
		SCA2	SK	⑤
		DFA + FIA on M3	MK	⑥
		SCA2 + FIA on M3	MK	⑦

Except SCA, we would like also to draw readers' attention to more sophisticated physical attacks exploiting AES cipher output data. Very powerful are the Differential Fault Analysis attacks (DFA) on AES proposed by Piret and Quisquater [139] where the fault is injected between the last and the penultimate AES round. Authors claim that only 2 faulty cipher-texts are necessary to break the AES. We would like to take these attacks also into consideration in the further analyses.

Tab. 5.2 summarizes all physical attacks on HCrypt1 (presented in Sec. 3.3). The examination of the HCrypt1 architecture (see Fig. 3.3) shows that the attacker can gain access to the cipher data input, cipher data output and decipher data input. It is important to note that the adversary could retrieve master key by taking advantage of FIA on multiplexers M1, M2 and M3 (see Fig. 3.3) and observing the first rounds of the decipher or the last round of the cipher (refer to attack ①, ②, ⑦ in Table 5.2). For instance, when M3 was targeted by FIA, the SCA2, which is more powerful than SCA1 as requiring less key hypotheses, can be launched on the master key. The decipher cannot be attacked at the last round (see the SCA5 attack), because its output is stored in the session key memory, which is not available to the attacker. Although the session key can be attacked relatively easily (attacks ④, ⑤), it could be changed dynamically in order to avoid the adversary to accumulate enough power traces for the SCA attack.

To sum up, the HCypt1 was resistant to software attacks, but remained vulnerable to SCA and FIA on master and sessions keys. Especially dangerous are FIA on multiplexer M3 capable of replacing the session key by the master key. To resist SCA attacks, both cipher and decipher had to be protected at the microarchitecture level using some data hiding or data masking techniques. Next, we will present new macroarchitecture that increases the robustness of the crypto-processor and simplifies also its protection at microarchitecture level.

5.2.2 Design of SCA and FIA Resistant HCrypt Version

Starting from HCrypt1 security analysis, the next objective for protecting secret keys against SCA and FIA was to reorganize blocks of the crypto-processor in such

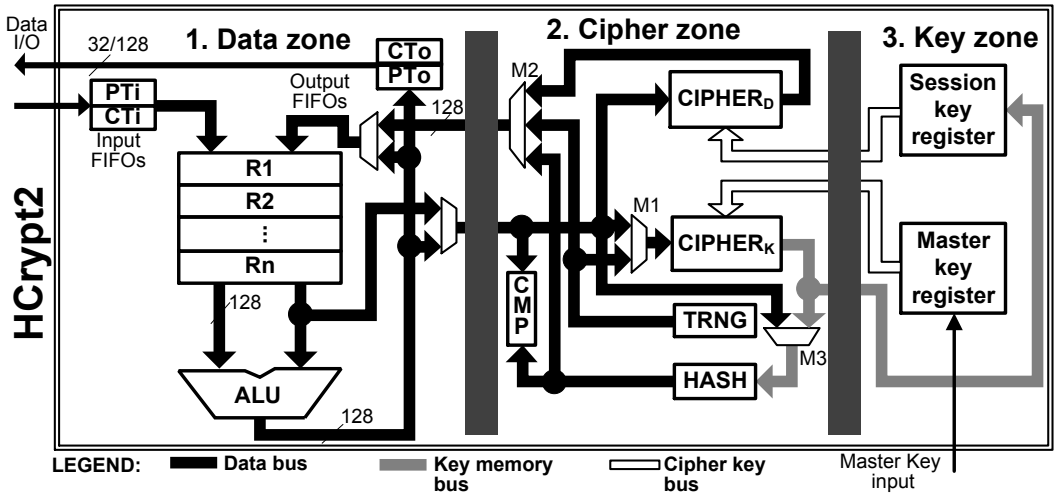

Figure 5.3: HCrypt2 with parallel cipher-cipher architecture

a way, that the cipher/decipher would be easier to protect and the attacker would not have direct access to data required for the attack. The new crypto-processor, replacing HCrypt1 in an MPSoC, had to offer the same features while reaching a higher security level. First, in order to avoid the use of the decipher that is more complex and more expensive to protect against side-channel attacks, we propose to replace the decipher by another cipher and to use the two ciphers for different and independent tasks. Second, we search for such HCrypt architecture, where only one cipher operates with master keys. This way, only one cipher has to be protected by expensive countermeasures. Third, because of this substantial modifications, some new blocks must be added.

The proposed HCrypt2 crypto-processor is depicted in Fig. 5.3. It complies with all separation rules discussed in the previous section. As its predecessor, it is divided to data, cipher and key zones. All internal buses are 128 bits wide. Basically, data buses can never pass to the key zone and cipher key buses to the data zone. Unlike HCrypt1, the key memory buses are used only to transport session keys in clear to session key register or to the Hash unit. No session key in clear can enter the cipher data input anymore only the cipher key input. This unique feature increases security against software attacks even more. The replacement of the decipher by another cipher caused that session keys cannot be authenticated anymore. To allow the authentication of session keys a Hash unit was added.

Next, we will present the hardware architecture of the HCrypt2 crypto-processor, and explain features distinguishing HCrypt2 from its predecessor.

5.2.2.1 Hardware Architecture

The transition from the HCrypt1 to HCrypt2 architecture necessitates various changes in data, cipher and key zones. Next, we present mostly the differences between two crypto-processors. More details on HCrypt1 can be found in Sec. 3.3.

Cipher zone

It can be observed in Fig. 5.3 that the cipher zone of the new crypto-processor is *substantially* changed. It contains two ciphers working independently. The Key Cipher (Cipher_K) is used solely for enciphering of session keys. In order to satisfy separation rules, it cannot be used for session key authentication because its output would need to be connected to the data bus. Instead, we propose to use a hashing function for key authentication. This way, the session key cannot be read in any way from the session key register. Note, that in case of HCrypt1, two 128-bit master key registers were necessary (one for encryption, one for decryption). Unlike HCrypt1, only the Cipher_K operates with the master key (no decryption master key is necessary) and so only one 128-bit master key register is required in HCrypt2.

Data are enciphered only by the Data Cipher (Cipher_D) using stored session keys. The new crypto-processor supports all basic block encryption modes except for the CBC mode, because the decipher is not available. However, data can be authenticated using the CBC-MAC mode, which requires only enciphering.

The last problem to solve is the session key generation and exchange. Session keys that are generated in a TRNG are cryptographically protected, because before being saved in the session key register and used for data enciphering, they are enciphered using the master key. Therefore, protected session keys obtained at the output of the TRNG can be exchanged using unprotected communication channel (in our case the crypto-processor data path).

After receiving the protected session key and its fingerprint generated using a hashing function, the receiving crypto-processor encrypts the session key, computes the key hash value and compares it (in CMP) with the received fingerprint. If fingerprints match, the session key is stored in the session key register. The communication protocol will be presented later in this chapter.

Data zone

The HCrypt2 data zone remains the same as that of HCrypt1 expect for the control unit where several small modifications were necessary. The data zone contains 128-bit data registers, a 128-bit ALU, input/output FIFOs (converting data format from the 32-bit internal SoC bus to a 128-bit crypto-processor data bus) and the control logic.

The control logic responsible for instruction fetching, decoding and execution had to be slightly modified because of changes in the cipher zone. Old instructions using decipher are replaced or updated to support the key cipher instead. Moreover, new instructions are necessary to control the hash unit.

Key zone

As in HCrypt1, the key zone contains session key and master key registers. The master key register is initialized via a dedicated input. The only difference is the absence of the storage for the authentication master key, since hashing algorithm does not required a secret key.

Table 5.3: The modifications of the HCrypt1 instruction set

A) Modified instructions

Instruction	Description
genk kx, rx, ry	1) Generate a protected session key and store it in the data register Rx, 2) Use Cipher_K to generate session key in clear and store it in Kx, 3) Compute the hash from the session key in clear and store it in Ry
getk rx, ry, ky,	1) Use Cipher_K to generate a session key in clear from the protected session key (read from Rx) and store it in Ky 2) Compute the hash of the session key in clear and compare it with the received hash stored in Ry and set the *Authenticated* flag if the two are matching

B) Removed instructions

Instruction	Description
putk kx, ry	Removed because its function is performed by the *genk* instruction

C) New instructions

Instruction	Description
hash rx, ry	Calculate hash of the Rx data and store it to Ry data register
gend rx	Generate random data word using TRNG and store it in data register Rx

5.2.2.2 Implementation of the Crypto-processor on FPGA

Similarly to HCrypt1, all three zones have to be placed into a separated logic partition, and each partition have to be mapped into an isolated physical block. Each physical block must be surrounded by unused logic blocks (i.e. CLB). Only selected wires can cross this isolation fence and interconnect the three zones. This way the physical separation at routing level is achieved. The two are the necessary preconditions for highly secure design.

5.2.2.3 Programming Means

The changes in the cipher zone require changes in the HCrypt1 instruction set shown in Tab. 3.5. First of all, *instructions for key generation and manipulation* (i.e. genk, getk and putk) are updated in the control unit to support the key cipher and the new hash unit. Moreover, the hash unit can be used optionally for data authentication, and so a new instruction was added into the instruction set. The modifications of the HCrypt1 instruction set are summarized in Tab. 5.3.

Since the instruction set was changed, HCrypt software must be modified too. Such modified software can be compiled with FlexASM and the resulting machine code can be used for VHDL simulation or real operation in FPGA hardware.

FlexASM assembler

Although the instruction set was modified, the FlexASM source code does not

have to be rewritten and recompiled, because FlexASM is independent from the processor. Actually, the processor instruction set is defined in the text file that is read by FlexASM, and so only modifications in this file are necessary to support HCrypt2 instruction set.

5.2.2.4 Communication Protocol

Although the robust cryptographic protocol presented for HCrypt1 (see Sec. 3.3.4) must be modified, it must be ensured that the session keys are exchanged and authenticated correctly. Not only session key exchange, but also data has to be protected by encryption and data authenticity has to be guaranteed. The communication protocol is implemented in HCrypt2 software, and all security critical operations are carried out by dedicated instructions. The key exchange part of this protocol is based on the authenticated point-to-point key update protocol described in Sec. 2.5.4.2.

We present the example of the cryptographic protocol for communication between sides Alice (A) and Bob (B) (see Fig. 5.4) in order to illustrate efficiency of the proposed structure. In practice, any other common cryptographic protocol can be implemented. Tasks 1, 2, 3, 8 and 9 are performed solely in the protected area (cipher and key zone) and tasks 5, 6, 7 are executed only in the unprotected area (data zone). Tasks 4 and 10 represent iterative implementation of encryption modes (EM) performed partly by the unprotected area (registering and xor-ing of subsequent data blocks) and partly by the protected area (enciphering E and deciphering E^{-1}). In this protocol, we assume that both devices were initialized by a trusted entity TE using the same enciphering (MK) master key and that the key is saved in master key register (protected area). The key exchange protocol is based on symmetric key cryptography. In the first step, the device starting the communication (A), generates a new session key SK: the protected session key PSK is generated by the TRNG (Task 1 in Fig. 5.4), the protection is removed by being enciphered with the key cipher E_{MK} using the master key MK resulting in the session key in clear SK, which is subsequently stored in clear in the session key memory (Task 2). Note that the session key PSK generated by the TRNG is considered protected and so can be directly transferred to the unprotected area. Finally, a digital fingerprint FP_A is generated by the hash unit ($HASH$) (Task 3).

When both the session key SK and its fingerprint FP_A are generated, Task 4 can be executed in a loop: data blocks ($DATA_i$) are sent from the data zone to the cipher zone, where they are enciphered using SK and sent back to the data zone as $CDATA_i$. ALU combines input and output blocks according to the encryption mode algorithm (EM) and computes $MCDATA_i$. Finally, packet P containing the protected session key PSK, its fingerprint FP_A, and enciphered data blocks $MCDATA_i$ is created (Task 5). The packet is sent to device B (Task 6). The crypto-processor on the side B receives the packet P (Task 7) and extracts the protected session key PSK and its digital fingerprint FP_A. The key is then sent to the cipher zone, where it is enciphered using the master key MK and the resulting

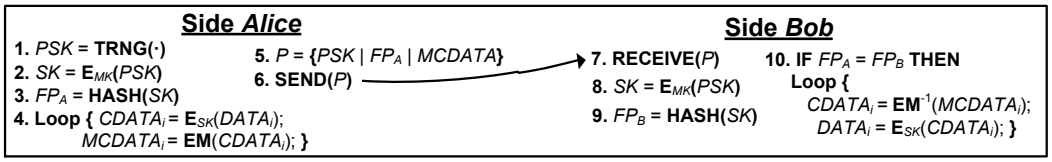

Figure 5.4: HCrypt2 communication protocol between two devices

session key in clear SK is stored in the session key memory (Task 8). A fingerprint FP_B of the session key SK is generated by the hash unit (Task 9) and compared with the received fingerprint FP_A (Task 10). If both FP_A and FP_B are matching, the session key is authenticated and can be used for data enciphering/deciphering (the loop in Task 10).

Note that the same packet structure can be used for both HCryp1 and HCrypt2. This straightforward packet structure is depicted in Fig. 3.6 and more details can be find in Sec. 3.3.4.

5.2.3 Evaluation of the HCrypt2 Security Against SCA and FIA

The HCrypt2 crypto-processor complies with proposed separation rules, so it is as resistant to software attacks as its predecessor. In contrast to HCrypt1, the Cipher_K is used only for the key enciphering and the Cipher_D only for data enciphering, deciphering and authentication. This fact brings three advantages. First, the AES decipher that is expensive to protect against hardware attacks is not needed any more, while the AES cipher, which is used instead, is much easier to protect. Second, both ciphers do not need the same level of protection. The Cipher_K that uses a long life master key as encryption key, should be better protected. The Cipher_D does not need any active protection, since session keys can be frequently regenerated according to the information leakage. Moreover, session keys are exchanged less frequently than data, and so Cipher_K could be much smaller (e.g. 32bit datapath, 8bit datapath). Such small cipher would be even cheaper to protect with additional countermeasures (i.e. masking, DRP, etc.). Third, the decipher and cipher in HCrypt1 required both encryption and decryption master keys (we remind that the decryption key is the last expanded encryption key). Unlike HCrypt1, HCrypt2 requires only one 128-bit master key register.

In the new processor, master key and session key buses are separated. The multiplexer M3 that was security critical in HCrypt1, is not necessary in HCrypt2. The master key is transferred only to the Cipher_K and only by a dedicated bus.

When considering SCA and FIA attacks on AES, it is very important that the Cipher_K output is not available to the attacker. Since the DFA attacks are targeting last cipher rounds, they are infeasible in the current crypto-processor configuration. The same is true for known cipher-text SCA2 attacks using a Hamming distance model that are the most easy to realize on the last cipher round. The only possibility to attack master key is by attacking the first round of Cipher_K (SCA1, attack ❶ and

Table 5.4: Possibilities of physical attacks on the HCrypt2 crypto-processor

Operation type	Equation	Potential attack	Target	#
Sess. key generation	$SK = C_{K_{MK}}(TRNG)$	FIA on M1 + SCA1	MK	❶
Sess. key transmission	$OUT = TRNG$	-	-	-
Sess. key reception	$SK = C_{K_{MK}}(IN)$	SCA1	MK	❷
Data Enciphering	$OUT = C_{D_{SK}}(IN)$	DFA	SK	❸
		SCA2	SK	❹

❷ in Tab. 5.4). However, in AES, the SCA1 attack is significantly harder than SCA2 as the first round includes the MixColumns operation. The SCA and FIA attacks against the Cipher_{D} can be countered if session keys are changed often enough in order to prevent the attacker from gaining sufficient number of power traces. The DFA protection of the Cipher_{D} could also be achieved by resilience properties at protocol level or physical level [140]. Physical attacks on HCrypt2 (based on the attacks defined in Tab. 5.1) are shown in Tab. 5.4.

To sum up, HCrypt2 is resistant to software attacks and FIA against the master key. Moreover SCA attacks against the Cipher_{K} are more difficult to perform than in HCrypt1. The Cipher_{D} protection depends on the key exchange protocol or on the use of resilient operations.

5.2.4 Implementation Results

The HCrypt2 crypto-processor was described as a parameterized VHDL block using Xilinx ISE version 12.4 and mapped to Xilinx Virtex-6 XC6VLX240TFF1156 device. HCrypt2 was implemented and tested in Xilinx ML605 evaluation board. The system utilizes just fine-grain FPGA resources and embedded RAMs/FIFOs. The parameterized width of input/output FIFOs can be set to 32 or 128 bits. The operation of HCrypt2 was first simulated using Xilinx ISim tool and subsequently tested in hardware. The communication between the PC and the board was realized via a small Cypress USB module.

Note that the proposed crypto-processor architecture is independent of the choice of enciphering algorithm, hashing function and TRNG principle. Selection of AES as a testing algorithm is quite natural – it is currently the most frequently used symmetric key enciphering algorithm. Selection of the MD5 algorithm as a hashing function was motivated by two facts: its VHDL sources are freely available and I/O block size (128-bits) was convenient for our application. However, because of concerns about MD5 security, the MD5 algorithm should be replaced by the new SHA-3 algorithm (Keccak) in the final version of the crypto-processor.

Table 5.5: Implementation results of HCrypt1 and HCrypt2 in Xilinx Virtex-6

	HCrypt1				**HCrypt2**				
	Cip	Decip	Other	**Total**	Cip_K	Cip_D	Hash	Other	**Total**
Slices	216	329	877	**1422**	216	216	371	815	**1618**
BRAMs(kb)	180	180	828	**1188**	180	180	0	828	**1188**

Table 5.6: Number of clock cycles required for the packet processing

	Packet overhead				**Data**
	Header decod.	Key decr. (by CIPK)	Key auth. (HASH)	Other	One 128-bit CFB loop
Clk. cycles	34	12	66	30	14

5.2.4.1 Cost Evaluation

Utilization and distribution of resources, together with those of HCrypt1, are illustrated in Tab. 5.5. These resources do not include USB interface since its size is negligible and it was used only during the tests in hardware. HCrypt1 occupies 1422 slices (including AES cipher and decipher, TRNG, processor's data path and 128-bit FIFOs, internal registers and control logic) and 1188 kb of embedded RAM while HCrypt2 utilizes 1618 slices (including two AES ciphers, MD5 hash function, and all other parts like in HCrypt1) and 1188 kb of embedded RAM.

5.2.4.2 Simulation

The simulation was used extensively during the design phase, when many HCrypt2 bugs were found and corrected. It was also used to debug HCrypt2 software. Finally, the simulation was used to estimate the maximum theoretical throughput of HCrypt2, since a cycle count for all critical loops can be accurately simulated.

For the fair comparison both crypto-processors used the same protocol and the same structure of packets. Software was slightly modified, since HCrypt2 has different instruction set and the new cipher zone operates differently. Since external HCrypt1 interface stayed intact, the same test bench was used for the simulation.

First, the crypto-processor was initialized by the master key and program code. Afterwards, packets containing the enciphered session key, its fingerprint (hash value) and 1024 blocks of data (each block 128 bits wide) were sent to the tested processor. The processor analyzed packets, deciphered and authenticated the received session key and deciphered the received data blocks using the CFB cipher mode. Next, it created new packets containing deciphered data blocks and sent them to the test bench for verification. The simulation procedure was exactly the same as in the case of HCrypt1 and further details can be found in Sec. 3.4.2.

Tab. 5.6 shows how many clock cycles are required to carry out different operations that packet processing requires. Data are processed using CFB block cipher mode. The CFB loop constitutes a critical loop.

Table 5.7: Dependence of maximum throughputs on number of 128-bit data blocks in the packet

128b data blks.	1	2	4	6	8	10	12	14	16
Packet data in kb	0.125	0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0
Packet speed in Mb/s	365.5	404.7	467.3	515.1	552.8	583.3	608.5	629.6	647.6
Data speed in Mb/s	73.1	134.9	233.6	309.0	368.5	416.6	456.3	489.7	518.1
Packet overhead in %	80	66.6	50.0	40.0	33.3	28.6	25.0	22.2	20.0

128b data blks.	20	32	50	64	100	128	256	512	1024
Packet data in kb	2.5	4.0	6.25	8.0	12.5	16.0	32.0	64.0	128.0
Packet speed in Mb/s	676.6	731.2	772.8	791.3	817.5	828.4	849.3	860.4	866.1
Data speed in Mb/s	563.8	650.0	715.5	744.4	786.0	803.3	836.2	853.7	862.7
Packet overhead in %	16.7	11.1	7.4	5.9	3.8	3.0	1.5	0.8	0.4

The simulation clock frequency was set to 100 MHz. Using this accurate timing information maximum theoretic packet and data throughputs could be calculated. The maximum throughputs are displayed in Tab. 5.7.

The maximum theoretic packet throughput, when processing 1024 data words in the packet, reaches 866.1 Mb/s if HCrypt operates at the clock frequency of 100 MHz. The pure data throughput (without packet overhead) reaches 862.7 Mb/s when processing 1024 data words.

5.2.4.3 Hardware Tests and Benchmark

The hardware tests are used to definitely prove the functionality of the system and measure the real maximum data throughput. HCrypt2 together with the USB interface were configured into the FPGA which was interconnected with the Cypress USB module. The USB module communicated with the PC. The test was the same as in the case of HCrypt1 and so further details can be found in Sec. 3.4.3.

Two frequency domains were present in the FPGA. HCrypt2 was operating at the 100 MHz internal clock frequency as well as all its internal parts including the AES ciphers. The USB interface was operating at the 48 MHz clock frequency. The two frequency domains were synchronized by HCrypt2 input and output FIFOs.

In order to reach the clock frequency of 100 MHz, extra timing constraints were required in the PlanAhead to shorten critical paths. Unlike the simulation, real maximum data throughput reached 821.7 Mb/s. When considering not only data but the whole packet, the maximum packet throughput was 825.0 Mb/s.

5.2.5 Discussion

Although HCrypt2 was implemented solely in Xilinx Virtex-6 FPGA, it can be implemented in any other family from Xilinx or other vendors. However the VHDL

code portability is limited to FPGA specific resources (i.e. embedded RAMs, FIFOs, PLLs), which have to be regenerated for every FPGA family.

The resource usage report show that most logic resources are dedicated to ALU and processor's data path. In HCrypt1, the decipher occupies almost 50% bigger area than the cipher. Although the decipher was replaced by the smaller cipher in HCrypt2, the MD5 hashing unit occupies more logic resources than the decipher itself. Therefore, HCrypt2 needs 20% more area than HCrypt1. However, the additional hashing unit (not available in HCrypt1) can be used for data integrity checking or other cryptographic protocols. Note that if countermeasures at microarchitecture level (i.e. masking, hiding) were used, the HCrypt2 size increase would be at least two-fold. This is however a very good result since only one cipher has to be protected using these expensive countermeasures. If both the cipher and decipher in HCryp1 included these countermeasures, the crypto-processor size increase would be at least four-fold.

Most of embedded RAM blocks are used for implementation of Input/Output FIFOs, that are used for data bus width conversion (if a 32-bit system bus should be connected to the 128-bit processor bus) and clock domain separation. Other RAM blocks are used for AES S-boxes. Utilization of the RAM memory blocks is not optimal because AES S-box tables occupy only 4 kb out of 18 kb BRAM blocks. However, the unused space can be used to store multiple types of S-boxes if some data masking technique would be used as a countermeasure.

Although the payload data throughput in simulation of approximately 862.7 Mbits/s is smaller than maximum packet throughput, it includes packet overhead and execution of the cipher modes. The obtained maximum real data throughput of 821.7 Mb/s has not reached the maximum theoretic data throughput. The reason could be in the inaccurate simulation model of I/O FIFOs which does not match exactly the real behavior of the embedded FIFO block. In contrast to HCrypt1, protected session key can be exchanged right after being generated by the TRNG thus less clock cycles are required. However, the session key authentication (using the MD5 hash unit) consumes much more clock cycles than CBC-MAC authentication in case of HCrypt1 (66 vs. 13). However, for very long packets (1024 data words of 128-bit size) the throughput decrease compared to HCrypt1 is negligible since the key is exchanged and authenticated only once. The critical path of HCrypt2 is slightly longer that the one of HCrypt1. Is is probably caused by longer critical path of the Hash unit. However, the maximum clock frequency decrease is almost negligible. Its speed could be further increased by instruction set optimization and parallel instruction execution (pipelining).

Currently, only basic NIST block cipher modes that do not require decipher are supported (i.e. OFB, CFB and CTR) [1]. If some new authentication modes (i.e. CMAC) and combined encryption and authentication modes (i.e. CCM, GCM) were required, they could be easily implemented in the proposed crypto-processor thanks to its flexible architecture.

Although the crypto-processor contains two ciphers, only one of them needs to be protected against hardware attacks (the key cipher). However, since the key

cipher output is not available to the attacker, it is intrinsically more robust against these attacks. This approach is a zero-cost countermeasure, because it increases security of master keys without requiring any additional FPGA resources.

Protection of the data cipher will depend on the protocol and the session key lifetime. Thanks to the availability of the key generator, session keys can be frequently regenerated. Since the session keys are enciphered in the key cipher, this cipher must be in this case better protected against side-channel attacks. Again, its intrinsic robustness against such attacks is very useful. Nevertheless, it can be further enhanced by other known countermeasures.

Logically, if more key hypotheses are required it is much harder to distinguish the correct key candidate and so more information leakage is necessary to maintain the same attack success rate. Interestingly, no study has been carried out so far which compares these two aspects. Indeed, this is an open problem and has to be confirmed not only by a DPA attack in hardware, but also by a theoretical explanation including study of impact of distinguishers on the attack success rate.

5.3 Conclusions

We proposed a novel architecture for crypto-processors guaranteeing secure key management based on physical separation of registers and buses aimed at key management from those aimed at data enciphering. The physical separation of the processor blocks was shown to be robust against software attacks, but vulnerable to SCA and FIA attacks.

We showed that by a tricky rearrangement of the crypto-processor blocks (while maintaining the physical isolation of security zones according to the proposed separation rules), the novel HCrypt2 crypto-processor is robust against software attacks, differential power analysis and fault injection attacks.

The low cost of the proposed novel system level countermeasures makes them very attractive in practical applications. Although they are shown to be efficient, they can be completed by some common microarchitecture level countermeasures based on data masking and data hiding. However, we have shown that only one of the two ciphers needs to be protected against attacks and so it is far less expensive to protect HCrypt2 than HCrypt1 using the same data masking or data hiding techniques. Furthermore, the cipher protecting session keys is rarely used (only once per packet). Thus, this cipher can be very small and this further decreases costs for countermeasures at microarchitecture level. Moreover, only one 128-bit master key register is required in HCrypt2 (only for enciphering) when compared to two 128-bit master key registers in case of HCrypt1 (for enciphering and deciphering).

The novel HCrypt2 crypto-processor was simulated using Xilinx ISim and tested in Xilinx Virtex-6 FPGA using ML605 evaluation board. However, it can be mapped to any other FPGA or ASIC. Its VHDL source code is freely available.

HCrypt2 has reached slightly lower maximum data throughput (821.7 Mb/s) than its predecessor (824.7 Mb/s). The reason is in longer computation of the

session key hash value during authentication.

Since the cryptographic part of the crypto-processor can be interfaced with the processor using a simple data bus, the data part of the crypto-processor can be easily replaced by a GPP such as ARM, Leon, NIOS II or MicroBlaze, while maintaining the advantages of the proposed solution.

The work described in this chapter was done in cooperation with Télécom ParisTech. It will be described in a paper and submitted to an international conference soon.

Chapter 6

Partial Reconfiguration of Crypto-processors

Contents

In the previous chapters, we have discussed only the possibility that an FPGA is configured only during the power-up phase. However, if we suppose the FPGA can be reconfigured during its run-time, new attacks can be carried out, not only during the reconfiguration of the FPGA, but also before when the bitstream is stored outside the FPGA. These aspects bring new challenges for the separation rules and the secure key management. We would like to propose modifications of separation rules that can guarantee secure key management even if parts of the system are reconfigured during run-time.

First, this chapter familiarizes the reader with the trends in the FPGA reconfiguration research field with a focus on bitstream security. Second, the novel separation rules considering the FPGA reconfiguration will be presented. These separation rules will be applied on the reconfigurable HCrypt-C crypto-coprocessor. To demonstrate its functionality and security, the reconfigurable HCrypt-C crypto-coprocessor

will be interconnected with the MicroBlaze processor and compared with the three static processor–coprocessor systems presented in Chap. 4.

6.1 FPGA Reconfiguration and Security Aspects

The introduction of FPGAs to market in eighties caused a boom in reconfigurable computing. FPGA vendors (i.e. Xilinx, Altera, Microsemi, etc.), but also numerous third-party vendors flooded the market with many intellectual property (IP) configurations for various purposes. Although previously used mostly in networking, the FPGA circuits emerged in 90-ties as a convenient platforms for implementation of cryptographic algorithms for their flexibility and lower costs in case of small-volume or middle-volume product series. It has become clear that the security of cryptographic algorithms and confidential keys, stored in the FPGA bitstream, is of paramount importance.

6.1.1 FPGA Bitstream Protection

Depending on how the configuration bitstream is stored inside an FPGA, the FPGA devices can be Antifuse-based, Flash-based or SRAM-based. When considering the security of the bitstream, the Antifuse-based FPGAs are the most secure, while the SRAM-based FPGA are considered the least secure [141]. Next, we will briefly introduce each FPGA technology while the most attention will be given to the SRAM-based technology.

6.1.1.1 Antifuse-based FPGAs

The antifuse is a microelectronic component which permanently decreases its resistance if a sufficiently high voltage is applied across its structure. This way the antifuse is considered as programmed and thus creates a conductive connection. The Antifuse-based FPGAs exploit this property to create conductive permanent connections within the programmable logic fabric.

The configuration recovery is extremely difficult, because a lot of failed tries are required to recover the configuration of one cell, while significantly damaging the rest of the circuit. For this reason, about 800,000 Microsemi A54SX16 chips with the same configuration are necessary to extract the whole configuration [142], [52].

Although very secure, the antifuse-based FPGA devices are only one time programmable, and thus any updates of cryptographic algorithms are not possible. However, in security applications configuration updates may be necessary if the security flaws are detected and implementation of additional countermeasures is required. For this reason we will not consider antifuse-based FPGAs further.

6.1.1.2 Flash-based FPGAs

The configuration initialization to flash-based FPGAs represents the programming of non-volatile flash switches. Unlike antifuse-based FPGAs, the Flash-based FP-

GAs are very flexible, because the flash technology enables to reconfigure the FPGA. Flash switches can be configured 10,000 to 100,000 times, because the accumulation of electrons in the floating gate causes gradual rise of transistor threshold voltage [52]. On the other hand, the flash-based FPGAs retain their configuration after being disconnected from the power source, and so they do not have to be reconfigured.

Currently, only the entire FPGA can be reconfigured. If the FPGA is configured in a hostile environment, the configuration bitstream must be encrypted and during the configuration process hard-wired AES unit decrypts the bitstream. This approach requires to initialize a secret AES key into the FPGA in a trusted environment. Although Microsemi FPGAs are configured from outside using the JTAG interface, the new SmartFusion FPGA family permits to configure the FPGA from inside by the hard-wired Cortex M3 processor. This in-application programming approach can be used to remotely configure the FPGA in a secure way [39].

The configuration recovery is very difficult, because the attacker must determine if the electric charge is present on each configured transistor floating gate. However, such an access to transistor floating gate necessitates removing passivation layer without damaging the circuit. Such an operation requires a very expensive equipment used also for the professional ASIC reverse-engineering. Afterwards, the attacker can use microprobing to determine the charge of each floating gate, or can power-up the chip in a vaccum chamber and observe the light emission of each flash transistor [142]. All these techniques require a physical access to the FPGA die, which is not always practical.

Much more dangerous are attacks that do not require depackaging. Recently, a backdoor in the flash-based Microsemi FPGAs, known for their unprecedented security, has been discovered by Skorobogatov et al. [143]. Authors claim that the backdoor can be accessed via the JTAG interface and can enable the readback of the whole FPGA configuration. The attack necessitates to try all the undocumented JTAG commands while exploiting side-channel information leakage. In response, Microsemi claims that for the highest security settings, the backdoor (actually an internal testing facility) can be permanently disabled. Even though it is recommended to physically protect the FPGA from outside to prevent the attacker from accessing the JTAG interface.

6.1.1.3 SRAM-based FPGAs

The most common are the volatile SRAM-based devices. Unlike the previous two technologies, the configuration is stored in static RAM memory cells which require permanent power supply to retain their content. For this reason SRAM-based FPGAs require to be configured each time the power supply is disconnected. During the FPGA initialization the bitstream stored in an external flash memory is transferred to the FPGA. However, if cryptographic implementations and secret keys were part of the bitstream, the security of these confidential data would be questionable.

Although previously claimed that FPGA layout cannot be reverse-engineered from the FPGA bitstream, this security-by-obscurity concept was soon proved vul-

nerable, when NeoCAD company reverse-engineered the bitstream generation flow [144, 145]. The netlist can be also recovered from the bitstream using the free software Debit [146]. The first countermeasure against bitstream reverse-engineering was proposed by Xilinx, where the bitstream was configured into the FPGA device only once and then the whole FPGA was backed-up with the battery [147, 145]. This way the FPGA did not have to be configured again. Although a sufficient security of cryptographic algorithms and confidential keys was achieved, this solution was very inconvenient for the short battery life (high power consumption of an FPGA).

The breakthrough came with the introduction of Xilinx Virtex-II FPGA, where the bitstream stored in an external configuration memory is encrypted with a secret configuration key. The configuration key is stored inside the FPGA in a RAM memory, which is backed up by a small battery. After power-up, the encrypted bitstream is deciphered by the hardwired 3-DES decipher using the configuration key stored in a battery-backed RAM and configured into the FPGA. The similar principle, based on the AES decipher, was later adopted to Virtex-4, -5, -6 and Spartan-6 FPGAs [148].

A non-volatile polyfuse key storage has been implemented by the Altera company in Stratix II. Altera Stratix III, IV and V feature both the non-volatile and volatile (battery-backed) key storages [149]. Recently, Xilinx has embedded a polyfuse key storage (i.e. eFUSE) in the Virtex-6 and Spartan-6 FPGA families. Polyfuses operate inversely when compared to antifuses — a configured polyfuse has high resistance. The polyfuse is much larger than the antifuse and so it is easier to locate the polyfuse and determine its state using a scanning electron microscope. The comparison of the eFUSE and SRAM key storage can be found in [150]. More details on the polyfuse and antifuse technologies can be found in [151], [152].

Despite the effort of FPGA producers, recent works demonstrate that even such security (storage of secret keys inside the FPGA) can be broken. Moradi et al. have shown that side-channel leakage during bitstream deciphering in RAM-based FPGAs (i.e. Xilinx) can be exploited to recover the secret configuration key stored inside the FPGA [153], [154]. The corrections are expected not sooner than in Xilinx Virtex-8 family. Although these attacks are feasible, their realization requires a lot of expertise and finances, and so they will not be considered further on in this chapter.

6.1.2 IP Bitstream Security in Partially Reconfigurable System

An interesting alternative to the full FPGA reconfiguration is the partial reconfiguration. When a partially reconfigurable design is used, the FPGA logic fabric is partitioned into static and reconfigurable partitions. Unlike the static partition, which may be reconfigured only during the full FPGA configuration, the reconfigurable partition can be reconfigured during system operation. This unique feature enables an in-field replacement of system parts. The partial reconfiguration is supported by Xilinx Virtex-4, -5, -6. Spartan-6 supports only difference-based partial reconfiguration, which allows only limited changes of the configured logic [46]. The

partial reconfiguration flow is also supported by Altera Stratix V, Aria V and Cyclone V [47].

One of the most common applications of the Partial Reconfiguration (PR) is the remote update of the partial bitstream. However, PR can also give the attacker more possibilities. The remote bitstream update enables the attacker to intercept the new PR bitstream and to send a forged bitstream instead. For this reason, besides the bitstream encryption, the bitstream authentication is also necessary.

The first idea to take advantage of the partial reconfiguration technology for the secure bitstream decryption was proposed by Bossuet et al. in 2006 [155]. The author proposed to use the partial reconfiguration to load a bitstream containing a decipher unit first. Afterwards the decipher is used to decrypt and configure the application bitstream stored in the external flash. This way, a user can select a ciphering algorithm for the application bitstream protection.

This concept was later extended to support FPGA digital rights management (DRM). DRM restricts the use of FPGA bitstreams and IP cores only in authorized devices. Moreover, DRM enables an IP developer to emit licenses binded to a specific FPGA and thus trace their IP distribution. DRM can thus very effectively prevent overbuilding and cloning. The first attempts came in 2006 when Synplicity released the "Open IP Encryption Flow" allowing to securely exchange and implement IP cores in FPGA development tools [156]. The scheme proposed to distribute IP vendors' private keys inside these development tools. However, software implementations can be easily disassembled and private keys extracted.

First hardware-based solution for IP licensing was proposed by Guneysu et al. in 2007 [157]. The authors propose to implement a publicly available personalization module (PM) in an FPGA prior to IP core purchase. An IP vendor can use the PM to establish a shared symmetric key in an FPGA and to bind the IP core license to the FPGA identification code (ID) and the symmetric key. Another advantage is that the principle can be used for FPGAs not capable of partial reconfiguration. Despite many advantages, this solution requires modification of current FPGAs, because it assumes the possibility to access a secure key storage (i.e. Battery-Backed RAM or eFUSE) from the FPGA logic fabric (impossible in current FPGAs). This principle was extended to multiple cores licensing in a single FPGA proposed by Drimer et al. [158]. Although the principle does not require the partial reconfiguration support in the FPGA, the design partitioning requires some features of the partial reconfiguration design flow.

Recently, Maes et al. [159] proposed a novel licensing scheme for hardware IP cores. The scheme requires a trusted third party, which provides a special security module. The module is used to decipher and install protected IPs into the partially reconfigurable area. The main advantage of this scheme is the fact that is does not require modifications of current FPGA devices.

The SeReCon system enabling the IP protection and the IP secure remote exchange in partially reconfigurable FPGAs was proposed by Kepa et al. [160, 161, 162]. First, the system is loaded into a static partition of the FPGA and it creates a root of trust. During the IP exchange process, SeReCon deciphers the IP, verifies

its digital signature, re-encrypts its bitstream and stores it in an unprotected external repository. When needed, the IP is loaded from the repository, deciphered and authenticated inside SeReCon, and configured into the reconfigurable partition via the ICAP port. SeReCon carefully monitors the access to the ICAP port so that only authorized IP cores can be configured.

In some cases the attacker may want to prevent the FPGA from updating its old IP bitstream (i.e. with security flaws) to a newer version (i.e. without security flaws). This kind of an attack, where the attacker interrupts a bitstream update and replaces a new bitstream with the old one, is called a replay attack. Replay attacks on FPGA bitstream were thoroughly studied and solutions were proposed by Devic et al. [163].

However, none of the above mentioned works considered secure key management and storage. The aim of our work is to increase the flexibility of the HCrypt-C crypto-coprocessor (see Chap. 4) by reconfiguring its cipher zone, while maintaining the unprecedented robustness to software attacks. This modification requires the extension of the separation rules introduced in Sec. 4.2. Similarly to SeReCon we would like to protect the access to the reconfiguration port (ICAP). Our original reconfigurable crypto-coprocessor architecture will enable GPPs to support secure key management and to facilitate remote updates of the ciphering algorithm and TRNG.

6.2 Separation Rules Involving Partial Reconfiguration

The separation rules presented in Sec. 4.2 suggest to separate the key memory from the GPP at protocol, system, architectural and physical levels. To simplify the separation, the system is partitioned into processor, cipher and key zones. Moreover, straightforward interfaces between two adjacent zones facilitate the physical isolation of these zones, using dedicated insulation fences, and simplifies the implementation of trusted bus macros. Furthermore, the fence separating the key zone from the cipher and processor zones can be used during partial reconfiguration of the system when updating the processor or the cryptographic algorithm, while maintaining the key memory contents unchanged. A straightforward interface design also simplifies implementation of bus macros in partial reconfiguration design flow.

Fortunately, the physical isolation design flow is similar to the partial reconfiguration one. Indeed, only a small effort is necessary when implementing partially reconfigurable logic in the device [164].

Upgrading the part of the hardware configuration using the dynamic partial reconfiguration technology is particularly interesting for many cryptographic applications. Next, we analyze the benefits and implications of a full and partial reconfiguration.

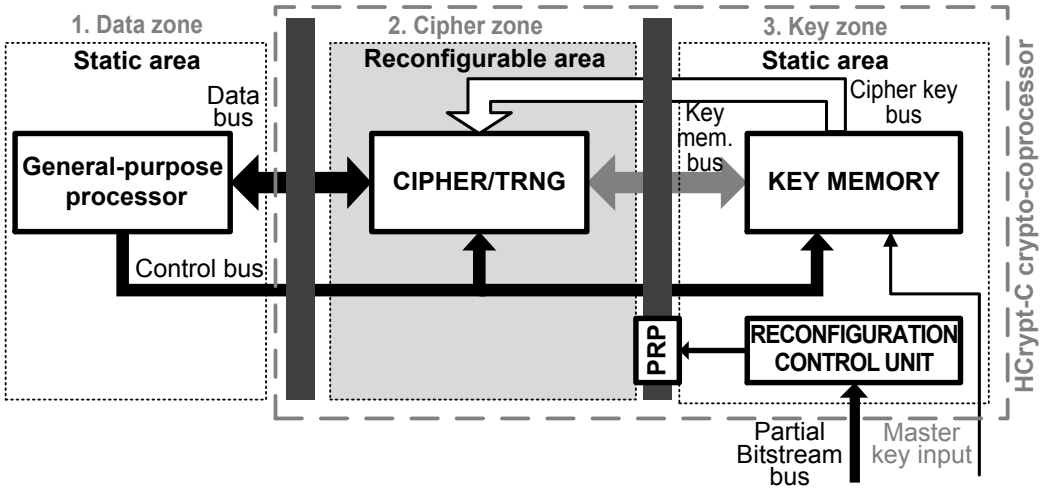

Figure 6.1: Separation rules including partial reconfiguration capability with reconfiguration of the cipher zone only

6.2.1 Total Reconfiguration Versus Partial Reconfiguration of the Device

Cryptographic modules based on symmetric key cryptography must share the same cryptographic key. If the device has been totally reconfigured, the key must be reinitialized. This must be done in a secure environment. Remote key initialization is very dangerous, because the master key cannot be enciphered without any other key and must consequently be transferred in clear. The initialization key cannot be included in the reconfiguration bitstream, because compromising one device would compromise the whole set of devices. Partial device reconfiguration is a better solution for single chip cryptographic applications: the master key can be initialized once in a protected environment and then stored in a static logic partition. During device upgrades, the key is kept the same and only partially reconfigurable blocks and the GPP software are allowed to be changed.

When dividing the system into static and reconfigurable partitions, the approach proposed in Sec. 4.2 for separating buses, key memory, cipher and processor zones is very useful, because communication interfaces between future static and reconfigurable partitions can be easily managed. There are two solutions for partitioning cryptographic system in Fig. 4.1: a) the key zone and processor zones are static and only the cipher zone is reconfigurable; b) the key zone (or at least the master key memory) is kept static and the rest of the device is dynamically reconfigurable. In both solutions, the cipher zone belongs to the reconfigurable partition and can thus reflect the required algorithm changes. The static area containing confidential keys must be handled with special care to avoid unauthorized access to confidential keys after device reconfiguration by fake reconfiguration data.

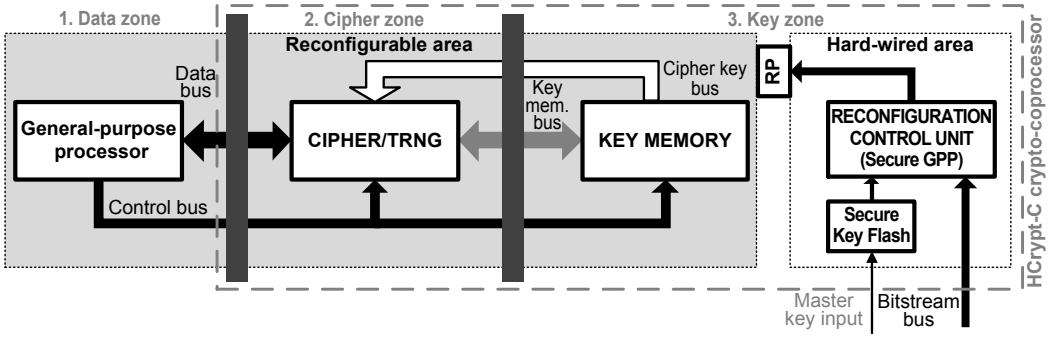

Figure 6.2: Separation rules including partial reconfiguration capability with reconfiguration of the processor, cipher and key zone

6.2.2 Validation of the Principle of HCrypt-C Partial Reconfiguration in SRAM FPGAs

The first solution from the previous section can be implemented in partially reconfigurable Xilinx FPGAs such as Virtex-5 and -6 and recently also in Altera FPGAs (starting from the Stratix V family) as shown in Fig. 6.1. We selected the Xilinx Virtex-6 FPGA family for our tests, because software tools for partial reconfiguration for Altera technologies had not been available yet.

In order to validate the separation concept, we propose three reconfigurable modules: A) one containing the AES cipher and decipher; B) the second one containing the DES cipher (composed of two DES cipher cores in parallel) and decipher (composed of two DES deciphers in parallel); C) the third one representing an empty black-box module.

As required, reconfigurable modules A, B and C have the same interfaces with the surrounding static partition (containing GPP and key memory). The dimensions of the reconfigurable partition must be set to meet the requirements of the biggest module (the one containing the AES cipher and decipher in our case).

The reconfiguration of the reconfigurable logic area is controlled by the reconfiguration control unit (RCU) placed in the static partition (see Fig. 6.1). It is in charge of transferring new logic to the reconfigurable area via a partial reconfiguration port (PRP). The partial bitstream is transferred to the RCU via the partial bitstream bus (PBB). Since the RCU must also ensure (by verifying bitstream integrity and authenticity) that the reconfigurable partition was not modified by an unauthorized person, its implementation is crucial for security. Additional techniques can be used to increase security (e.g. zeroization of all FFs and configuration bits in the reconfigurable area by the PRP before being configured using the new partial bitstream [165]). The RCU can be implemented using another GPP, a dedicated processor or a state machine, but the reconfiguration controller must be strictly separated from the GPP controlling the communication channel. The executable code of the RCU must be write protected to resist software attacks. Since our objective was to demonstrate the separation concept, in this first case, we only implemented a simple

reconfiguration controller (state machine) placed in the static logic area.

6.2.3 Reconfiguration of HCrypt-C Crypto-coprocessor in FPGAs Containing Hardwired GPPs

Many FPGAs are not partially reconfigurable. For devices containing hardwired GPPs featuring a non-volatile memory (e.g. MicroSemi SmartFusion FPGA), we propose another solution that combines the possibility of hardware upgrades with secure key management (see Fig. 6.2). The hardwired GPP can serve as a reconfiguration controller and confidential keys (or at least master keys) can be saved in its non-volatile memory – secure key flash memory (SKF). In this case, the entire programmable logic fabric can serve as a reconfigurable area containing all the system blocks (including the second GPP) except for the reconfiguration controller and the confidential key memory. The bitstream is transferred into RCU via the bitstream bus (BB) and the logic area is configured via the reconfiguration port (RP).

Note that all security zones and especially processor and cipher zones must remain separate as required in Sec. 4.2. It is also of paramount importance that the use of the hardwired GPP is strictly limited to reconfiguration tasks. In particular, it must not be connected to the data bus of the second GPP that is in charge of data processing, i.e. the separation rules must be maintained.

Although this solution is less flexible and slower than the one discussed above, it still offers secure key management and high-level protection of confidential keys. However, the use of the powerful hardwired processor for device reconfiguration will exclude it from other tasks (and especially from communication with the cipher and from key management). Another GPP will have to be used and implemented in the reconfigurable zone, what makes this solution less attractive. For this reason, we did not implement it in hardware. Next, we will give details of the implementation of the reconfigurable HCrypt-C crypto-coprocessor.

6.3 Design of the Reconfigurable HCrypt-C

The HCrypt-C crypto-coprocessor was introduced in Sec. 4.3. The full cryptographic system consists of the GPP and the wrapper (see Fig. 4.2). HCrypt-C is incorporated inside the wrapper together with communication interface and control unit. The main function of the wrapper is to adapt the GPP-specific communication bus to the HCrypt-C interface. To demonstrate its flexibility, HCrypt-C is interconnected with Altera NIOS II using the internal processor bus, Xilinx MicroBlaze using the FSL coprocessor bus and ARM Cortex M1 using AMBA peripheral bus. The implementation of HCrypt-C together with the three GPP implementations are presented in Sec. 4.5. However, the partial reconfiguration technology is only supported by Xilinx FPGAs, and so a reconfigurable version of the HCrypt-C crypto-coprocessor will be interconnected only with the Xilinx MicroBlaze GPP.

In order to support the reconfiguration of the cipher zone, both the data and key zone implementations have to be modified. The reconfiguration is controlled by

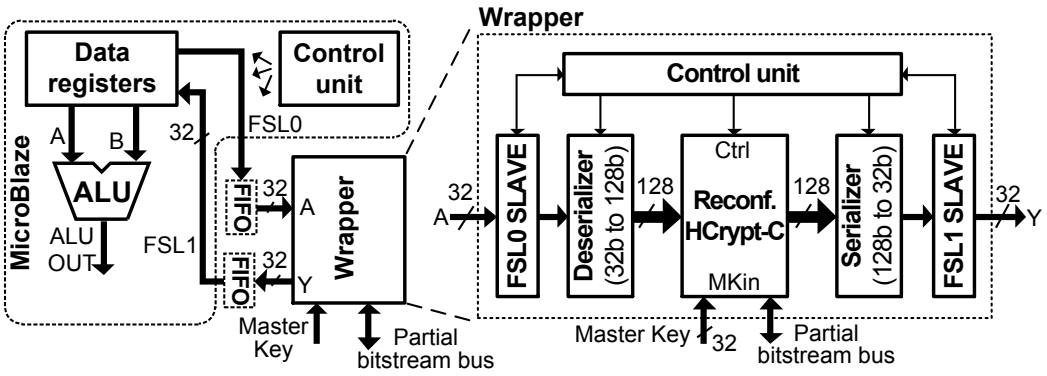

Figure 6.3: MicroBlaze interconnected to Reconfigurable HCrypt-C crypto-coprocessor extension via the FSL bus wrapper

the RCU and performed via the PRP. Both RCU and PRP are located in the static key zone. The Xilinx Virtex-6 FPGA family embeds ICAP which can be used as the PRP. Moreover, ICAP supports bitstream decryption and authentication and so the RCU can be significantly simplified. Several modifications are also required in the wrapper. The RCU reads the configuration through the Partial Bitstream Bus (PBB). The wrapper has to implement a new instruction which instructs the RCU to select the partial bitstream and start the configuration. Moreover, new flag indicating successful reconfiguration must be implemented in the wrapper status register. Microblaze firmware also requires slight modification to support the reconfiguration of the cipher zone.

The MicroBlaze interconnected to the reconfigurable HCrypt-C crypto-coprocessor via the wrapper is shown in Fig. 6.3.

6.3.1 Reconfigurable Cipher Zone Modules

6.3.1.1 AES reconfigurable module

The AES reconfigurable module is almost identical to the original cipher zone presented in Sec. 4.5. The only difference is the addition of the partition pins and bus macros to every signal/bus crossing from the reconfigurable cipher zone to static zones. The AES reconfigurable module is illustrated in Fig. 6.4. The module implements AES cipher and decipher, both featuring 128-bit wide folded datapath. For the sake of simplicity, only a PRNG based on a 128-bit linear feedback shift register is implemented instead of a TRNG. In a real system, it must be replaced by a TRNG (e.g. PLL-based TRNG [36]). The partition pins are implemented as a one-input LUT.

6.3.1.2 DES reconfigurable module

The DES reconfigurable module structure is very similar to the AES one. The AES cipher and decipher are replaced by the DES cipher and decipher units. In order to

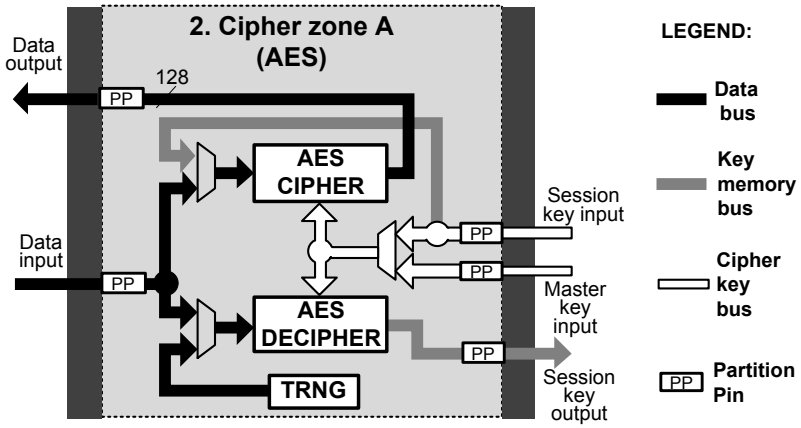

Figure 6.4: Reconfigurable cipher zone containing the AES cipher, decipher and TRNG (AES reconfigurable module)

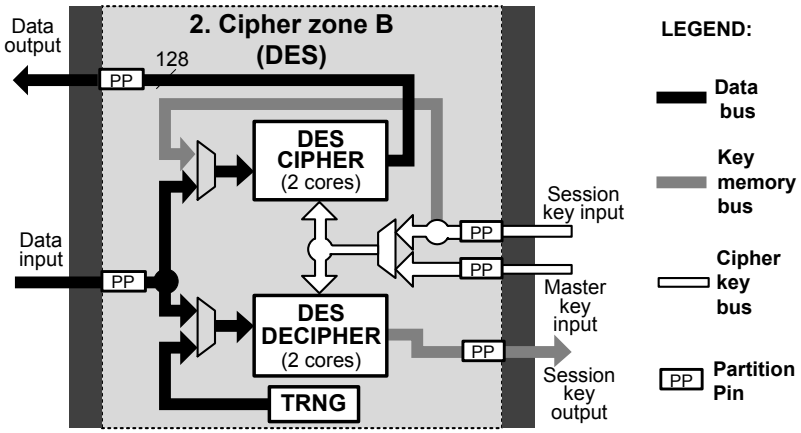

Figure 6.5: Reconfigurable cipher zone containing the DES cipher unit, decipher unit and TRNG (DES reconfigurable module)

avoid bus conversion (note that DES operates on 64-bit data blocks), the DES cipher unit is composed of two 64-bit DES cipher cores working in parallel (further refered as DES cipher). Similarly, the DES decipher unit is composed of two parallel 64-bit DES decipher cores (further refered as DES decipher). This is the necessary step to freely interchange and fairly compare AES and DES units. Consequently, ciphering of one 128-bit block can be accomplished in 16 clock cycles. As a consequence, this architecture allows to execute two independent computations of a same block cipher mode at a time. The detailed architecture of the DES reconfigurable module is given in Fig. 6.5.

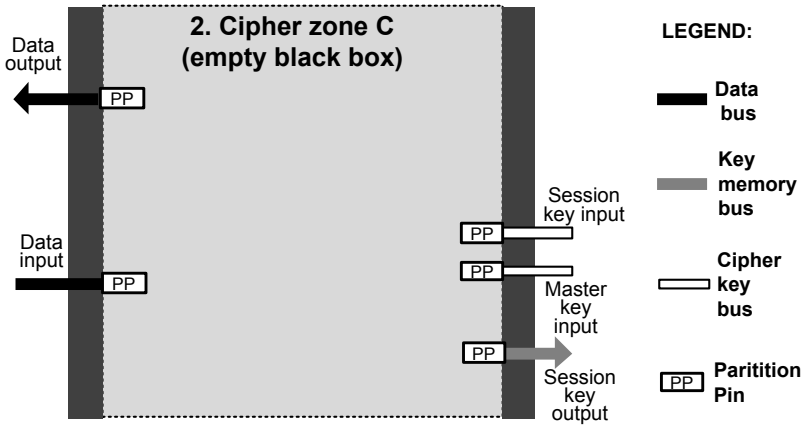

Figure 6.6: Empty reconfigurable cipher zone containing only partition pins (empty black-box reconfigurable module)

6.3.1.3 Empty black-box reconfigurable module

The last reconfigurable module is the empty black-box module. This version of the cipher zone incorporates no FPGA resources except for the partition pins and reconfiguration bus macros. These resources are necessary only for the compatibility with the previous two modules. However, partition pins utilize some FPGA slices which can be observed in the final resource utilization report. Although the empty black-box module is used only for a comparison with the other modules and brings no benefit to this system, its low power consumption can perfectly fit into the power saving strategy of the low-power devices. The empty black-box reconfigurable module is shown in Fig. 6.6.

6.3.2 Reconfiguration Control Unit

The most important element for device reconfiguration is the RCU. Since most of its security tasks can be delegated to hardwired decipher and HMAC via ICAP, the complexity of the RCU can be reduced to a simple state machine.

The architecture of the RCU is illustrated in Fig. 6.7. The RCU is composed of the address counter, 32-bit to 8-bit converter and a control state machine. The MicroBlaze sends the reconfiguration instruction to the wrapper. The wrapper sets the configuration select signal (CSEL) and pulses HIGH the configuration start (CSTART). When the configuration is finished, RCU asserts the configuration done flag (CDONE). After configuration start, address counter is loaded with the bitstream memory address. The the state machine enables/disables counter while monitoring the data valid memory flag (DVLD). The partial bitstream is loaded in 32-bit words and converted to 8-bit words suitable for ICAP. Note that ICAP is also capable of 32-bit word operation but only with the unencrypted bitstream.

The bitstream authentication by the hard-wired HMAC unit prevents the attacker from loading the tampered bitstream into the FPGA. In case that an ad-

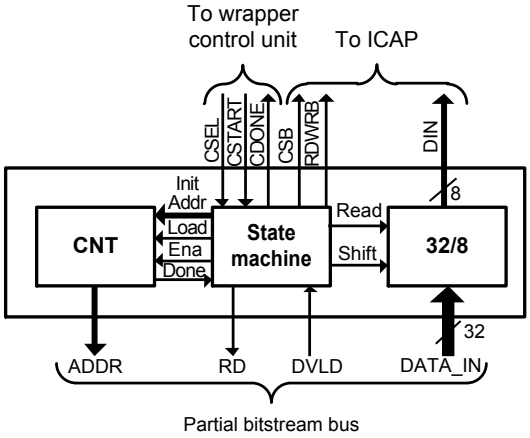

Figure 6.7: Architecture of the Reconfiguration Control Unit

vanced error checking (per small blocks) is required, a comprehensive explanation can be found in [166]. More details on Xilinx partial reconfiguration are provided in the Partial reconfiguration guide [48] and Virtex-6 configuration guide [167].

For the sake of simplicity, the RCU implements an address counter with hard-coded bitstream length. This approach is possible, because the partial bitstream file size is constant for all reconfigurable modules. The RCU stops the reconfiguration process when a counter reaches the end of the bitstream file. Optionally, the end of the partial reconfiguration bitstream can be detected from the partial bitstream itself by searching for the DESYNCH command. However, such an approach requires deeper knowledge of the partial bitstream structure (see [167]) and so it is not considered in this work.

6.4 Implementation Results

The extended MicroBlaze supporting reconfigurable cipher zone was described in VHDL and mapped into Virtex-6 XC6VLX240TFF1156 device (Xilinx ML605 evaluation kit) using ISE (ver. 12.4). Besides the processor and its HCrypt-C crypto-coprocessor extension, a small block containing 16-bit data interface to the external Cypress USB device CY7C68013A was connected to the evaluation board for data transfers from/to the PC. The USB module was also used for emulation of the non-volatile memory, storing the partial bitstreams. The USB module was used only for testing purposes and it does not constitute an inherent part of the system. For this reason, it is not included in further resource utilization reports.

6.4.1 Cost Evaluation

The results of the implementation are given in Tab. 6.1. Columns 8 and 9 in Tab. 6.1 list the resources of a partially reconfigurable system based on MicroBlaze extended by the AES reconfigurable module. The reconfiguration overhead can be observed

Table 6.1: Utilization of Reconfigurable MicroBlaze and compared FPGA resources by tree processors with the HCrypt-C crypto-coprocessor containing the AES cipher

	NIOS II		Cortex M1		MicroBlaze		Reconf. MBlaze	
	ALMs	RAM kb	Tiles	RAM kb	Slices	RAM kb	Slices	RAM kb
System total	2531	243.9	15053	216.0	1954	1206.0	2619	1206.0
→ Processor	1204	187.9	9433	104.0	1350	774.0	1350	774.0
→ HCrypt-C	1327	56.0	5620	112.0	604	432.0	1269	432.0
Ext. overhead	110.2%	29.8%	59.6%	107.7%	44.7%	55.8%	94.0%	55.8%

Table 6.2: Comparison of three versions of the reconfigurable module (RM) in an extended MicroBlaze system

	Static part.	Reconfigurable partition			Total with AES RM
		AES	DES	Empty black box	
Slices	1976	643	442	264	2619
RAM kb	846	360	0	0	1206

by comparing the MicroBlaze system with the static HCrypt-C and the one with the reconfigurable HCrypt-C crypto-coprocessor. Individual results for three different reconfigurable modules in the reconfigurable MicroBlaze system are listed in Tab. 6.2. Note that the size of the system including both the static and reconfigurable partition containing AES in Tab. 6.2 is the same as the sum of resources occupied by the processor and HCrypt-C in columns 8 and 9 in Tab. 6.1. However, HCrypt-C in column 8 of Tab. 6.1 occupies more slices (1269) than the reconfigurable AES module in Tab. 6.2 (643), because the key memory and the wrapper of the second module are included in the static partition.

The reconfigurable MicroBlaze system can be compared with its static extended MicroBlaze version as well as with extended NIOS II and extended Cortex M1 (see Tab. 6.1). The area is expressed in number of occupied ALMs for Altera family, Slices for Xilinx family and Tiles for Microsemi family. For comparison, we recall that one ALM in Altera Stratix II family contains two 4-input LUTs and two FFs. One Slice in Xilinx Virtex-6 family contains four 6-input LUTs and eight FFs. One Tile in Microsemi 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. For clarity, we present the results for the processor and for its HCrypt-C crypto-coprocessor extension separately.

6.4.1.1 Hardware tests and benchmarks

The setup used for hardware tests was explained in Fig. 6.8. The test was very similar to the test presented in Sec. 4.6. Unlike static system tests, reconfigurable system required modification of the USB interface so that is can emulate the non-volatile

memory containing partial bitstream. The partial bitstream was transported from the PC to the RCU via USB wrapper. Moreover, the PC benchmark application was also modified to support partial bitstream files.

In order to compare the throughput with that of the three static extended processors, the clock frequency of all four systems was fixed to 50 MHz. The throughput was evaluated by transferring packets from the PC to the FPGA (and vice versa) via a USB interface. The communication protocol, packet structure and testing procedure are similar to those used for HCrypt testing. The whole testing process started when the benchmark application accessed the *Master key file* and transferred the 128-bit AES and 128-bit DES (two 56-bit keys – one for each DES core, remaining bits were not used) master keys to the FPGA. When master keys were present in HCrypt-C, the PC application read the *Input packet file* and sent data packets to GPP data input.

Each packet contained a cipher zone version, encrypted session key, its digital fingerprint and five 128-bit payload blocks. The *Input packet file* contained 8192 packets for the AES cipher zone, 8192 packets for the DES cipher zone and 8192 packets for the empty black-box cipher zone (for testing only). Packets were analyzed in the GPP, which then sent the session key and its fingerprint to HCrypt-C. Once the key was decrypted and authenticated, the processor sent data blocks to be decrypted using CFB block cipher mode. Subsequently, the processor recreated new packets containing received decrypted data and sent them back to the PC.

When all AES packets were finished and the first DES packet arrived, the MicroBlaze instructed the RCU to reconfigure the cipher zone to the DES module. The RCU requested the DES module from the USB interface, which was promptly loaded from the PC. When the reconfiguration was completed, the MicroBlaze together with HCrypt-C continued the packet processing. The benchmark application read the resulting AES and DES packets, verified them and saved them to the *Output packet file*.

The achieved maximum real data throughput of the reconfigurable MicroBlaze system featuring the AES reconfigurable module was 18.4 Mb/s and the DES reconfigurable module was 17.9 Mb/s. The data throughput of the empty black-box module cannot be evaluated. The comparison of the reconfigurable module data throughput with the other three static processors is given in Tab. 6.3. The maximum system frequency was estimated to 171.3 MHz, which is less than the maximum system frequency of the static system estimated to 232.5 MHz.

The ICAP and USB modules were both operating at the clock frequency of 48 MHz. The maximum theoretical throughput of the 8-bit ICAP was 46,876 kB/s. However, the USB module was capable of delivering the partial bitstream with maximum speed of 37,648 kB/s. The partial bitstream size was 530.25 kB for all three reconfigurable modules. The reconfiguration lasted 14.23 ms in case of AES, 14.34 ms in case of DES and 14.10 ms in case of the empty black-box reconfigurable module. The partial reconfiguration performance results compared with the three static processors are summarized in Tab. 6.4.

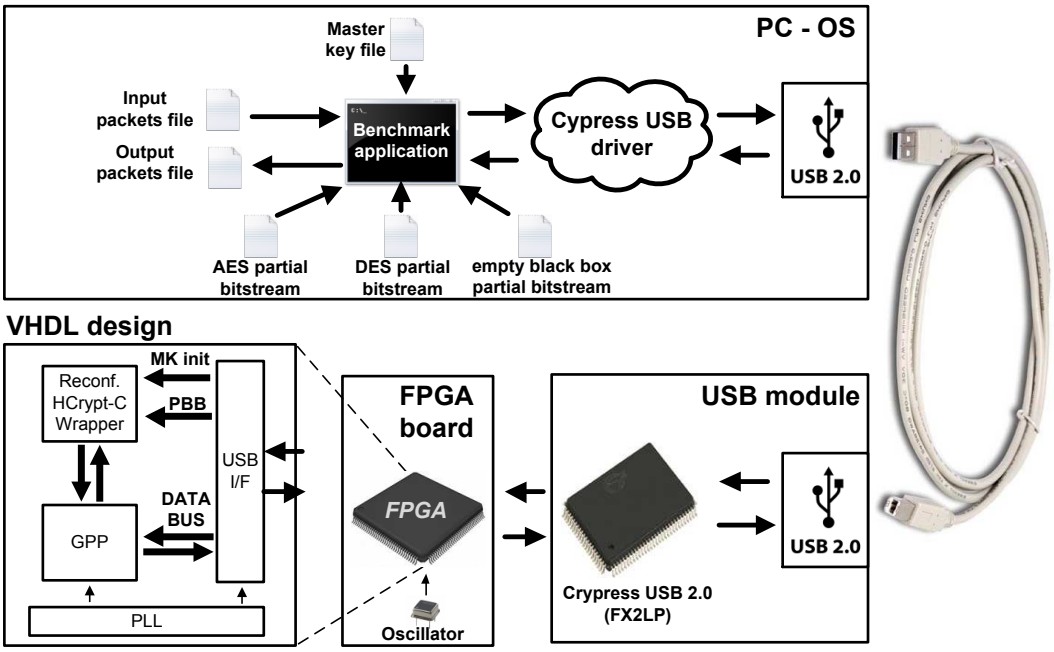

Figure 6.8: Hardware test setup of the Secure General-Purpose Processor with reconfigurable HCrypt-C crypto-coprocessor

Table 6.3: The maximum data throughput comparison of three static system featuring AES cipher zone and reconfigurable MicroBlaze system featuring AES, DES and empty black-box reconfigurable cipher zones

NIOS II	Cortex M1	MicroBlaze	Reconf. MBlaze		
AES (Mb/s)	**AES** (Mb/s)	**AES** (Mb/s)	**AES** (Mb/s)	**DES** (Mb/s)	**B-Box** (Mb/s)
25.1	12.2	18.4	18.4	17.9	—

6.5 Discussion

For partial reconfiguration of Xilinx FPGAs, partition pins increase the occupied area. This effect can be observed if an empty reconfigurable block is implemented. Note that five 128-bit buses and associated control signals must cross the reconfigurable partition border and each slice can implement at most four partition pins. Since not all slices are fully utilized, the number of slices in the empty black box is higher than required. Some extra resources are also necessary for the RCU, that was added into the key zone. Finally, the modified wrapper consumes some extra resources, too. All these effects can be observed when comparing two versions of the extended MicroBlaze system (with or without partial reconfiguration of the cipher zone): the area extension overhead with the AES cipher is 94.0% versus 44.7%. The cost of the isolation of security zones is negligible. Interestingly, both the static and

Table 6.4: ICAP throughput (left) and comparison of three reconfigurable module sizes and configuration time length (right)

ICAP throughput	
theoretical (kB/s)	real (kB/s)
46,876	37,648

AES module		DES module		empty black box	
size (kB)	length (ms)	size (kB)	length (ms)	size (kB)	length (ms)
530.25	14.23	530.25	14.34	530.25	14.10

reconfigurable systems require the same extra amount of the embedded memory (55.8 %).

Note that partial reconfiguration also increases the latency of the system. However, we assume that reconfiguration is only performed occasionally (e.g. if a new attack countermeasure has to be applied). The timing overhead of the reconfiguration is thus negligible.

Since the system timing was only slightly modified, almost the same data throughput is achieved (18.4 Mb/s). The only difference is the requirement to test if a configured cipher zone module matches the one selected in the packet. The DES cipher by its structure requires more rounds to complete the computation. This additional latency results in lower data throughput of 17.9 Mb/s.

Although no speed penalty was observed when comparing static and reconfigurable HCrypt-C featuring the AES module, the insertion of the partition pins and constrained routing (reconfiguration bus macros) lowered maximum clock frequency (171.3 MHz). If both systems were clocked to their maximum frequency, the speed penalty would be observable. The maximum clock frequency can be higher if extra constraints are applied to routing resources and partition pins logic. However, a very long compilation time (more than one day) forced us to loosen these constraints.

The size of each partial bitstream was the same (530.25 kB). The reason is that the bitstream size does not depend on the number of used resources in the reconfigurable area, but rather on the size and location of the reconfigurable area, which must be the same for all reconfigurable modules.

Since both master key bus and data bus are entering the reconfigurable area, it must be sure that direct physical connection between these buses does not exist inside the reconfigurable area. For this reason, only approved reconfigurable modules are allowed to configure FPGA. This issue is solved by the authentication of the partial bitstream that is performed by the hard-wired HMAC unit.

Although the size of the partial bitstream files is the same (530.25 kB), the time interval necessary to configure the partial bitstreams into the FPGA slightly differs. This difference can be explained by various throughputs of the USB interface caused by unpredictable behavior of the PC operating system. However, for the real external bitstream memory, the configuration time would be the same.

It is clear that the architectures presented here could be further analyzed and optimized from the point of view of performance, area and power consumption.

However, it is also clear that the security overhead due to the creation of isolation fences and due to the application of separation rules is negligible.

Although HCrypt2 concept was described before the reconfigurable HCrypt-C in this thesis, HCrypt2 study was conducted after reconfigurable HCrypt-C. For this reason, the proposed solution does not provide protection against physical attacks such as side-channel attacks. We assume that such countermeasures will be included in the cipher module. Indeed, one of the main advantages of the partial reconfigurability of the device is that the cipher module can be updated by a new cipher version implementing countermeasures against the most recent attacks. This approach is not possible in hardwired architectures.

The principle presented in this chapter can be extended to any GPP. One of solutions would be to use an open source GPP and to include HCrypt-C in the processor's data path as was the case with the extended NIOS II processor. This principle can be also applied to a specific-purpose processor such as the one presented in Chap. 3 [80].

6.6 Conclusions

In this chapter, we proposed a novel reconfigurable cryptographic coprocessor that enables secure key management using any GPP. Since the reconfiguration technology violates separation rules presented in Chap. 3, we presented their modification for two special cases.

First, the total reconfiguration of the flash-based FPGA was considered. Such an FPGA must contain a hard-wired reconfiguration control unit and a non-volatile master keys storage. The reconfiguration control unit is responsible for authentication and decryption of the FPGA bitstream and its secure initialization into the FPGA configurable fabric. The closest existing FPGA supporting such configuration is the Microsemi Smart Fusion FPGA. Although this approach enables secure key management for the GPP in FPGA logic, the powerful hard-wired processor (i.e. Cortex M3) is excluded from all system tasks and so this solution is not very attractive. For this reason, this system was not implemented.

Next, the security of confidential keys stored in the partially reconfigurable system were evaluated and a solution was proposed. The system was partitioned into the data, cipher and key zone. Unlike the previous approach, both data and key zones were static and only the cipher zone could be reconfigured during the system run-time. All three security zones were isolated from each other. The security of the partial bitstream was guaranteed by the reconfiguration control unit placed in the key zone. We also showed that stringent security requirements can be met in partially reconfigurable system only if the key zone is kept static.

In order to demonstrate the concept, a partially reconfigurable HCrypt-C was proposed for the MicroBlaze system and was implemented in Xilinx Virtex-6 FPGA. Three different variants of the cipher zone (reconfigurable modules) were tested: AES, DES and the empty black box. In order to compare reconfigurable and static

MicroBlaze systems the AES module was configured first. The reconfigurable MicroBlaze featuring the AES module is bigger than the static one: the area extension overhead of the first type of the first module is 94.0% versus 44.7% of the second one.

The maximum data throughput was evaluated by transferring packets from the PC via USB interface to the MicroBlaze system and results were transferred back to the PC. Both reconfigurable and static MicroBlaze systems containing the AES module reached the data throughput of 18.4 Mb/s. Slightly lower performance was achieved using the DES module (17.9 Mb/s).

The size of every reconfigurable module was 530.25 kB. The reconfiguration time required for every module was approximately 14.20 ms.

Since reconfigurable HCrypt-C is not protected against side-channel attacks, HCrypt2-C (including zero-cost countermeasures) must be designed in the future. The work presented in this chapter was published in [106].

Chapter 7

Summary of Contributions and Conclusions

This thesis presents research work that has been conducted in order to increase security of key management performed in Multi-processor System on Chip. The main objective was to propose a new crypto-processor architecture organized in such a way that the key storage is securely separated from the rest of the system. A new instruction set had to be proposed to facilitate the separation of data and key processing operations. Moreover, crypto-processor had to be optimized for block cipher modes and be able to exchange data and keys in packets. At least one crypto-processor version had to contain countermeasures against side-channel attacks. Since reconfiguration can also be a source of security weakness, its security had to be evaluated and solution proposed. To simplify software development for the new crypto-processor, an assembler tool was necessary. Moreover, frequent modifications of the crypto-processor instruction set required a flexible assembler. The last objective was to evaluate the security level of the proposed crypto-processor.

The research work has reached the proposed goals, and proposed also different small studies and solutions to various other problems.

7.1 Summary of Contributions

The most important proposals and contributions of this research work are as follows:

1. *Set of separation rules* enabling secure key management on cryptographic processors and coprocessors.
2. *HCrypt crypto-processor* that complies with the separation rules. HCrypt is immune to software or fault injection attacks aimed on the disclosure of secret keys.
3. *FlexASM assembler* with a flexible instruction set defined in a text file.
4. *HCrypt-C crypto-coprocessor* guaranteeing secure key management on any general-purpose processor.
5. *Zero-cost countermeasures* increasing security of crypto-processors against side-channel attacks.
6. *Secure key management on totally or partially reconfigurable systems.*

All these contributions necessitated extensive studies of state-of-the-art architectures, security analyses, a lot of implementation work and thorough testing using simulations and hardware.

7.2 Conclusions

The main contribution and proposals of this thesis is a set of separation rules, which instructs designers how to construct cryptographic processors or coprocessors with secure key management. If these rules are respected, the system is immune to software and fault injection attacks targeting a disclosure of secret keys. Separation rules suggest to partition a system to the three following zones, each with different security privileges: data zone, cipher zone, key zone. The three zones must be separated at protocol, system, architectural and physical levels. As a consequence, secret keys stored in the key zone cannot pass directly to the data zone, but must pass through the cipher (representing a physical barrier) in the cipher zone. Thus, any software or fault injection attacks cannot disclose secret keys.

These unique separation rules were demonstrated on the novel HCrypt crypto-processor. HCrypt is a 128-bit cryptographic processor featuring the ALU, which is optimized for block cipher modes and packet management, two AES ciphers, true random number generator, and secure key storage for both session and master keys. HCrypt is divided into data, cipher and key zone each having different security privilege. The most secure is the key zone, where secret keys are stored in clear. HCrypt protects data by using the generated session keys. The session keys must be securely exchanged prior to data exchange using the master keys. Master keys are transferred to HCrypt via a dedicated 32-bit bus. We demonstrated HCrypt operation, security, and programmability and measured its throughput. Tests in Xilinx Virtex-5 and -6 FPGAs indicated data throughput of 824.7 Mb/s, while HCrypt exchanged session keys and processed data in packets using the 128-bit CFB block encryption mode.

To simplify the HCrypt software development, new FlexASM assembler tool was created. Unlike other assemblers, an instruction set can be defined in a text file from which it is loaded prior to the compilation of an assembly code. This feature solved the problem of frequent modifications of HCrypt instruction set during its development. However, FlexASM can be used for other processors as well.

Next, it was shown how crypto-coprocessors supporting secure key management could be constructed. For cost-critical applications requiring an execution of general-purpose instructions, we proposed the HCrypt-C crypto-processor extension to any general-purpose processor. This crypto-coprocessor respects all stringent separation rules, thus enabling secure key management on general-purpose processors. The structure of the HCrypt-C crypto-coprocessor is very similar to the HCrypt crypto-processor. It has the 128-bit wide datapath, embedded AES cipher, decipher and TRNG. The HCrypt-C crypto-coprocessor can be interconnected with a processor using an internal processor bus, a dedicated coprocessor bus or a peripheral bus. All

three bus types were compared, demonstrated on the three processor implementations, and tested in hardware. The Nios II achieved data throughput of 25.1 Mb/s, MicroBlaze 18.4 Mb/s, and Cortex M1 12.2 Mb/s while each system exchanged session keys and processed data in packets using the 128-bit CFB block encryption mode.

In order to increase robustness to side-channel attacks, new zero-cost countermeasures were proposed and demonstrated on novel HCrypt2. Unlike countermeasures at microarchitecture level, zero-cost countermeasures are implemented on macroarchitecture level. We showed that by a tricky rearrangement of the crypto-processor blocks (while maintaining the physical isolation of security zones according to the proposed separation rules), the novel HCrypt2 crypto-processor is robust against software attacks, and more robust against differential power analysis and fault injection attacks. Moreover, we have shown that only one of the two ciphers needs to be protected against attacks, and so it is far less expensive to protect HCrypt2 than HCrypt1 using the same data masking or data hiding techniques. Furthermore, the cipher protecting the session keys is rarely used (only once per packet). Thus, this cipher can be very small, which further decreases the cost for countermeasures at microarchitecture level. In addition to that, only one 128-bit master key register is required in HCrypt2 (only for enciphering) when compared with two 128-bit master key registers in case of HCrypt1 (for enciphering and deciphering).

The last contribution is the extension of separation rules for the partially reconfigurable system. Two different cases are considered: a) FPGA featuring only total reconfiguration of the logic fabric; b) FPGA supporting partial reconfiguration technology. In order to maintain high security, crypto-processor has to include a proposed reconfiguration control unit, secure key storage and an internal configuration interface. The reconfiguration control unit must be hardwired (e.g. embedded secure processor) in case of a totally reconfigurable system, or stored in the static area in case of a partially reconfigurable system. The partially reconfigurable system was implemented using the MicroBlaze processor with the HCrypt-C crypto-coprocessor. The processor and key zones were considered static, and the cipher zone was reconfigurable. For testing purposes, three version of the cipher zone were tested: AES (featuring AES cipher, decipher and TRNG), DES, and empty black box. The partial bitstream size of all three HCrypt-C versions was 530.25 kB.

7.3 Perspectives

The aim of this study was to propose a cryptographic processor that can be included in the SecReSoC Multi-Processor System on Chip.

The completion of the system and its demonstration is another challenge. Security of zero-cost countermeasures has to be proved by real DPA attacks.

The presented static and reconfigurable versions of HCrypt-C do not include zero-cost countermeasures. For this reason, HCrypt2-C involving zero-cost coun-

termeasures must be designed. Such crypto-coprocessor can be better protected against side-channel attacks and fault injection attacks.

Another challenge is the exchange of the long-term master key. Indeed, some initial secret is essential in every cryptographic system. However, master key establishment scheme depends heavily on a target system. In our case a fully symmetric system with manual distribution of master keys was sufficient.

An interesting challenge would be an inclusion of an asymmetric cryptographic unit which can be used for master key distribution and packet data signatures. Nevertheless, it is crucial to maintain a high security level.

List of Publications

International Journal Papers

1. L. Bossuet, M. Grand, L. Gaspar, V. Fischer, G. Gogniat. ***Architectures of flexible symmetric key cryto engines — a survey: from hardware coprocessor to multi-crypto-processor system on chip***, to be published in the journal ACM Computing Surveys, volume 45, number 4, December, 2013.
2. L. Gaspar, V. Fischer, L. Bossuet, R. Fouquet. ***Secure extension of FPGA general purpose processors for symmetric key cryptography with partial reconfiguration capabilities***, Published in journal ACM Transactions on Reconfigurable Technology and Systems, volume 5, issue 3, number 16, October, 2012.

Full Papers on International Refereed Conferences

3. L. Gaspar, V. Fischer, L. Bossuet, M. Drutarovsky. ***Cryptographic Extension for Soft General-Purpose Processors with Secure Key Management***, In FPL 2011: 21st IEEE International Conference on Field Programmable Logic and Applications, pages 500-505, Chania, Greece, September 5th – 7th, 2011.
4. L. Gaspar, V. Fischer, L. Bossuet, R. Fouquet. ***Secure extensions of FPGA soft core processors for symmetric key cryptography***, In ReCoSoC 2011: 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip, pages 1-8, Montpellier, France, June 20th – 22nd, 2011.
5. L. Gaspar, V. Fischer, F. Bernard, P. Cotret, L. Bossuet. ***HCrypt: A Novel Concept of Crypto-processor with Secured Key Management***, In ReConFig 2010: International Conference on Reconfigurable Computing and FPGAs, pages 280-285, Cancun, Mexico, December 13th – 15th, 2010.

Short Papers on International Refereed Conferences

6. P. Cotret, J. Crenne, G. Gogniat, J-P. Diguet, L. Gaspar, G. Duc. ***Distributed security for communications and memories in a multiprocessor architecture***, In RAW 2011: 18th Reconfigurable Architectures Workshop, Published in IPDPS 2012: 26th Annual International Parallel & Distributed Processing Symposium. pages 326-329, Anchorage, Alasca, USA, May 16th – 20th, 2011.
7. L. Gaspar, V. Fischer, F. Bernard, L. Bossuet. ***Cryptographic NIOS II extension with secure key management***, In PhD forum of DATE 2011: Design, Automation & Test in Europe Conference, page 1, Grenoble, France, March 14th – 18th, 2011.
8. L. Gaspar, M. Drutarovsky, V. Fischer, N. Bochard. ***Efficient AES S-boxes implementation in non-volatile FPGAs***, In FPL 2009: 19st IEEE International Conference on Field Programmable Logic and Applications, pages 649-653, Prague, Czech Republic, August 31st – September 2nd, 2009.

Short Paper at National Refeered Conference

9. L. Gaspar, H. Maghrebi, J-L. Danger, V. Fischer. ***HCrypt2 : Un cryptoprocesseur sécurisé par conception***, In SoC-SiP 2012: Le 6éme colloque du GDR System on Chip – System in Package, pages 1-2, Jussieu, Paris, France, June 13th – 15th, 2012.

Bibliography

[1] National Institute of Standards and Technology (NIST). **Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation-Methods and Techniques**. *NIST*, 2001. (Cited on pages xvii, 10, 12, 13, 14, 15, 51, 53 and 108.)

[2] National Institute of Standards and Technology (NIST). **Special Publication 800-38B: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication**. *NIST*, 2005. (Cited on pages xvii, 12, 13 and 16.)

[3] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of applied cryptography*. CRC, 1996. (Cited on pages xvii, 6, 17, 27 and 28.)

[4] G. Stoneburner, C. Hayden, and A. Feringa. **Engineering principles for information technology security (a baseline for achieving security)**. Technical report, DTIC Document, 2001. (Cited on page 6.)

[5] Organisation for Economic Co-operation and Development. **OECD guidelines for the security of information systems and networks: towards a culture of security**. *OECD*, 2002. (Cited on page 6.)

[6] C. Perrin. **What is the CIA Triad**. *Available at: http://www.techrepublic.com/blog/security/the-cia-triad/488*, 2008. (Cited on page 6.)

[7] National Institute of Standards and Technology (NIST). **FIPS-46: Data Encryption Standard (DES)**. *FIPS*, 1977. (Cited on pages 8, 9 and 38.)

[8] National Institute of Standards and Technology (NIST). **FIPS-197: Advanced Encryption Standard (AES)**. *FIPS*, 2001. (Cited on pages 9, 10 and 91.)

[9] National Institute of Standards and Technology (NIST). **FIPS-81: DES modes of operation**. *FIPS*, 1980. (Cited on page 10.)

[10] National Institute of Standards and Technology (NIST). **Special Publication 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality**. *NIST*, 2004. (Cited on page 12.)

[11] National Institute of Standards and Technology (NIST). **Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC**. *NIST*, 2007. (Cited on page 12.)

[12] R. Rivest. **RFC 1320: The MD4 Message-Digest Algorithm**, 1992. MIT and RSA Data Security, Inc. (Cited on page 17.)

[13] R. Rivest. **RFC 1321: The MD5 Message-Digest Algorithm**. 1992. (Cited on pages 17 and 18.)

[14] National Institute of Standards and Technology (NIST). **FIPS 180-1: Secure Hash Standard – SHA-1**. *FIPS*, 1995. (Cited on page 17.)

[15] National Institute of Standards and Technology (NIST). **FIPS 180-2: Secure Hash Standard – SHA-2**. *FIPS*, 2001. (Cited on page 17.)

[16] National Institute of Standards and Technology (NIST). **SHA-3 Selection Announcement**. NIST, 2012. http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf. (Cited on page 17.)

[17] B. Badrignans, J.L. Danger, V. Fischer, G. Gogniat, L. Torres, et al. *Security Trends for FPGAS: From Secured to Secure Reconfigurable Systems*. Springer, 2011. (Cited on pages 20, 21 and 22.)

[18] National Institute of Standards and Technology (NIST). **FIPS 140-1: Security Requirements for Cryptographic Modules**. *FIPS*, 1977. (Cited on page 20.)

[19] National Institute of Standards and Technology (NIST). **FIPS 140-2: Security Requirements for Cryptographic Modules**. *FIPS*, 1994. (Cited on pages 20, 23 and 24.)

[20] National Institute of Standards and Technology (NIST). **Special Publication 800-22: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications**. *NIST*, 2010. (Cited on page 20.)

[21] G. Marsaglia. **DIEHARD: a battery of tests of randomness**. *See http://stat.fsu.edu/geo/diehard.html*, 1996. (Cited on page 20.)

[22] W. Killmann and W. Schindler. **AIS-31: Functionality classes and evaluation methodology for true (physical) random number generators**. *Bundesamt für Sicherheit in der Informationstechnik – BSI*, 2001. (Cited on page 20.)

[23] W. Killmann and W. Schindler. **AIS-31: A Proposal for: Functionality classes and evaluation methodology for random number generators**. *Bundesamt für Sicherheit in der Informationstechnik – BSI*, 2011. (Cited on page 20.)

[24] M. Bucci and R. Luzzi. **Design of testable random bit generators**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'05*, pages 147–156. Springer, 2005. (Cited on pages 20 and 21.)

[25] S.H.M. Kwok and E.Y. Lam. **FPGA-based high-speed true random number generator for cryptographic applications**. In *Proceedings of TENCON'06*, pages 1–4. IEEE, 2006. (Cited on page 20.)

[26] KH Tsoi, KH Leung, and PHW Leong. **Compact FPGA-based true and pseudo random number generators**. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines – FCCM'03*, pages 51–61. IEEE, 2003. (Cited on page 20.)

[27] B. Sunar, W.J. Martin, and D.R. Stinson. **A provably secure true random number generator with built-in tolerance to active attacks**. *IEEE Transactions on Computers*, **56**(1):109–119, 2007. (Cited on page 21.)

[28] S. Yoo, B. Sunar, D. Karakoyunlu, and B. Birand. **A robust and practical random number generator**, 2007. Citeseer. (Cited on page 21.)

[29] K. Wold and C.H. Tan. **Analysis and enhancement of random number generator in FPGA based on oscillator rings**. *International Journal of Reconfigurable Computing*, **2009**:4, 2009. (Cited on page 21.)

[30] D. Schellekens, B. Preneel, and I. Verbauwhede. **FPGA vendor agnostic true random number generator**. In *Proceedings of the conference on Field Programmable Logic and Applications – FPL'06*, pages 1–6. IEEE, 2006. (Cited on page 21.)

[31] M. Bucci, L. Giancane, R. Luzzi, M. Varanonuovo, and A. Trifiletti. **A novel concept for stateless random bit generators in cryptographic applications**. In *Proceedings of the International Symposium on Circuits and Systems – ISCAS'06*, pages 4–pp. IEEE, 2006. (Cited on page 21.)

[32] P. Kohlbrenner and K. Gaj. **An embedded true random number generator for FPGAs**. In *Proceedings of ACM/SIGDA International Symposium on Field programmable gate arrays – FPGA'04*, pages 71–78. ACM, 2004. (Cited on page 21.)

[33] T. Tkacik. **A hardware random number generator**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'02*, pages 875–876. Springer, 2002. (Cited on page 21.)

[34] M. Varchola and M. Drutarovsky. **New high entropy element for FPGA based true random number generators**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'10*, pages 351–365. Springer, 2010. (Cited on page 21.)

[35] M. Goresky and A. Klapper. *Algebraic Shift Register Sequences*. Cambridge University Press, 2012. (Cited on page 21.)

[36] V. Fischer, M. Drutarovský, M. Šimka, and F. Celle. **Simple PLL-based True Random Number Generator for Embedded Digital Systems**. In *Design and Diagnostics of Electronic Circuits and Systems Workshop – DDECS'04*, pages 129–136, 2004. (Cited on pages 21, 22, 49 and 120.)

[37] V. Fischer, M. Drutarovský, M. Šimka, and N. Bochard. **High performance true random number generator in Altera Stratix FPLDs**. In *Proceedings of the conference on Field Programmable Logic and Application – FPL'04*, pages 555–564. Springer, 2004. (Cited on page 22.)

[38] Microsemi. **Fusion Family of Mixed Signal FPGAs datasheet**, 2012. http://www.actel.com/documents/Fusion_DS.pdf. (Cited on page 32.)

[39] Microsemi. **SmartFusion Customizable System-on-Chip (cSoC) datasheet**, 2012. http://www.actel.com/documents/SmartFusion_DS.pdf. (Cited on pages 32 and 113.)

[40] Altera. **Stratix II Device Handbook**, 2011. http://www.altera.com /literature/hb/stx2/stratix2_handbook.pdf. (Cited on page 32.)

[41] Xilinx. **Virtex-5 FPGA User Guides**, 2012. http://www.xilinx.com /support/documentation/virtex-5_user_guides.htm. (Cited on page 32.)

[42] Xilinx. **Virtex-6 FPGA User Guides**, 2012. http://www.xilinx.com /support/documentation/virtex-6_user_guides.htm. (Cited on page 32.)

[43] J. M. McConnell. **TEMPEST/2-95**. *NSTISSAM*, 1995. http://cryptome .org/tempest-2-95.htm. (Cited on pages 32 and 47.)

[44] Xilinx. *XAPP1105: Single Chip Crypto Lab Using PR/ISO Flow with the Virtex-5 Family for ISE Design Suite 12.1*, 2011. http://www.xilinx.com/. (Cited on pages 33 and 47.)

[45] M. McLean and J. Moore. **FPGA-based single chip cryptographic solution**. *Military Embedded Systems*, 2007. (Cited on pages 33 and 47.)

[46] Xilinx. **Difference-Based Partial Reconfiguration – XAPP290**, 2007. http://www.xilinx.com/support/documentation/application_notes/xapp290 .pdf. (Cited on pages 33 and 114.)

[47] Altera. **Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs**, 2010. http://www.altera.com /literature/wp/wp-01137-stxv-dynamic-partial-reconfig.pdf. (Cited on pages 33 and 115.)

[48] Xilinx Inc. **Partial Reconfiguration User Guide – UG702**, 2010. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/ ug702.pdf. (Cited on pages 34, 39 and 123.)

[49] S. Bajikar. **Trusted platform module (tpm) based security on notebook pcs-white paper**. *Mobile Platforms Group, Intel Corporation*, 2002. (Cited on page 38.)

[50] S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga. **Security evaluation of scenarios based on the TCG's TPM specification**. In *Proceedings of the European Symposium on Research in Computer Computer Security – ESORICS'07*, pages 438–453. Springer, 2007. (Cited on page 38.)

[51] L. Chen and M. Ryan. **Offline dictionary attack on TCG TPM weak authorisation data, and solution**. In *Proceedings of the Future of Trust in Computing conference*, pages 193–196. Springer, 2009. (Cited on page 38.)

[52] T. Wollinger and C. Paar. **How secure are FPGAs in cryptographic applications?** In *Proceedings of Field-Programmable Logic and Applications conference – FPL'03*, pages 91–100. Springer, September 2003. (Cited on pages 39, 112 and 113.)

[53] T. Wollinger, J. Guajardo, and C. Paar. **Security on FPGAs: State-of-the-art implementations and attacks**. *ACM Transactions on Embedded Computing Systems (TECS)*, **3**(3):534–574, 2004. (Cited on page 39.)

[54] P. Davies. **Flexible Security**, 2003. Cryptography & Interoperability, White Paper, Thales e-Security. (Cited on page 39.)

[55] Altera. **FPGA Run-Time Reconfiguration: Two Approaches**, 2010. http://www.altera.com/literature/wp/wp-01055-fpga-run-time-reconfiguration.pdf. (Cited on page 39.)

[56] P. Kocher, J. Jaffe, and B. Jun. **Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems**. In *Proceedings of Advances in Cryptology – CRYPTO'96*, **1109**, pages 104–113. Springer, 1996. (Cited on pages 39 and 86.)

[57] D. Page. **Theoretical use of cache memory as a cryptanalytic side-channel**. *Department of Computer Science, University of Bristol, Tech. Rep. CSTR-02-003*, 2002. (Cited on page 39.)

[58] C. Percival. **Cache missing for fun and profit**. In *Proceedings of the BSDCan conference*. Citeseer, 2005. (Cited on page 39.)

[59] D.J. Bernstein. **Cache-timing attacks on AES**, 2005. Citeseer. (Cited on pages 39 and 40.)

[60] E. Bangerter, D. Gullasch, and S. Krenn. **Cache games–Bringing access-based cache attacks on AES to practice**. In *Proceedings of Constructive Side-Channel Analysis and Secure Design workshop – COSADE'11*, pages 215–221. CASED, 2011. (Cited on pages 39 and 40.)

[61] D. Osvik, A. Shamir, and E. Tromer. **Cache attacks and countermeasures: The case of AES**. In *Proceedings of Cryptographers' Track at the RSA conference – CT-RSA'06*, pages 1–20. Springer, 2006. (Cited on page 40.)

[62] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. **A side–channel analysis resistant description of the AES S–box**. In *Proceedings of the workshop on Fast Software Encryption*, pages 199–228. Springer, 2005. (Cited on pages 40 and 90.)

[63] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, and G. Gogniat. **Architectures of flexible symmetric key crypto engines – a survey: from hardware coprocessor to multi-crypto-processor system on chip**. *To be published in ACM Computer Surveys*, **45**(4), December 2013. (Cited on pages 40, 42, 43, 47 and 63.)

[64] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass. **System design methodologies for a wireless security processing platform**. In *Proceedings of Design Automation Conference – DAC'02*, **39**, pages 777–782. ACM, August 2002. (Cited on pages 41 and 42.)

[65] J. Burke, J. McDonald, and T. Austin. **Architectural support for fast symmetric-key cryptography**. In *Proceedings of the conference on Architectural support for programming languages and operating systems – ASPLOS'00*, **34**, pages 178–189. ACM, 2000. (Cited on pages 41 and 42.)

[66] S. Tillich, J. Grossschädl, and A. Szekely. **An instruction set extension for fast and memory-efficient AES implementation**. In *Proceedings of the conference on Communications and Multimedia Security – CMS'05*, pages 11–21. Springer, 2005. (Cited on pages 41 and 42.)

[67] S. Tillich and J. Grossschädl. **Instruction set extensions for efficient AES implementation on 32-bit processors**. In *Proceedings of Cryptographic Hardware and Embedded Systems workshop – CHES'06*, pages 270–284. Springer, 2006. (Cited on page 41.)

[68] P. Hämäläinen, M. Hännikäinen, and T. Hämäläinen. **Review of hardware architectures for advanced encryption standard implementations considering wireless sensor networks**. In *Proceedings of the workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation – SAMOS'07*, pages 443–453. Springer, 2007. (Cited on page 41.)

[69] S. Tillich and C. Herbst. **Boosting AES performance on a tiny processor core**. In *Proceedings of Cryptographers' Track at the RSA conference – CT-RSA'08*, pages 170–186. Springer, 2008. (Cited on page 41.)

[70] Xilinx. *CryptoBlaze: 8-Bit Security Microcontroller*, September 2003. http://www.xilinx.com/support/documentation/application_notes/xapp374.pdf. (Cited on pages 41 and 42.)

[71] S. Gueron. **Intel Advanced Encryption Standard (AES) Instructions Set**, 2010. White Paper, Intel Mobility Group, Israel Development Center. (Cited on pages 41 and 42.)

[72] L. Barthe, L-V. Cargnini, P. Benoit, and L. Torres. **Optimizing an Open-Source Processor for FPGAs: A Case Study**. In *Proceedings of the conference on Field Programmable Logic and Application – FPL'11*, pages 551–556. IEEE, 2011. (Cited on page 41.)

[73] A. Martin, T.R. Newman, and D. Murotake. **Development approaches for an international tactital radio cryptographic API**. In *Proceedings of the Software Design Radio Technical Conference – SDR'08*, pages 1–6, 2008. (Cited on page 43.)

[74] L. Wu, C. Weaver, and T. Austin. **CryptoManiac: a fast flexible architecture for secure communication**. In *Proceedings of the Symposium on Computer Architecture – ISCA'01*, pages 110–119. IEEE, 2001. (Cited on pages 43 and 44.)

[75] C. Weaver, R. Krishna, L. Wu, and T. Austin. **Application specific architectures: a recipe for fast, flexible and power efficient designs**. In *Proceedings of the conference on Compilers, architecture, and synthesis for embedded systems – CASES'01*, pages 181–185. ACM, 2001. (Cited on page 43.)

[76] D. Theodoropoulos, I. Papaefstathiou, and D. Pnevmatikatos. **Cproc: An efficient Cryptographic Coprocessor**. In *Proceedings of the conference on Very Large Scale Integration – VLSI-SoC'08*. Citeseer, 2008. (Cited on pages 43 and 44.)

[77] D. Theodoropoulos, A. Siskos, and D. Pnevmatikatos. **CCproc: A custom VLIW cryptography co-processor for symmetric-key ciphers**. In *Proceedings of the workshop on Reconfigurable Computing: Architectures, Tools and Applications – ARC'09*, pages 318–323. Springer, 2009. (Cited on page 43.)

[78] R. Buchty, N. Heintze, and D. Oliva. **Cryptonite–A programmable crypto processor architecture for high-bandwidth applications**. In *Proceedings of the conference on Organic and Pervasive Computing – ARCS'04*, pages 184–198. Springer, 2004. (Cited on pages 43 and 44.)

[79] M. Grand, L. Bossuet, G. Gogniat, B.L. Gal, J.P. Delahaye, and D. Dallet. **A Reconfigurable Multi-core Cryptoprocessor for Multichannel Communication Systems**. In *Proceedings of the Parallel and Distributed Processing Workshops – IPDPSW'11*, pages 204–211. IEEE, 2011. (Cited on pages 43 and 44.)

[80] L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret. **HCrypt: A Novel Concept of Crypto-processor with Secured Key Management**. In *Proceedings of the conference on Reconfigurable Computing and FPGAs – ReConFig'10*, pages 280–285. IEEE Computer Society, 2010. (Cited on pages 43, 44, 47, 63, 90 and 128.)

[81] L. Gaspar, V. Fischer, F. Bernard, and L. Bossuet. **Cryptographic NIOS II extension with secure key management**. In *PhD Forum proceedings of Data, Automation and Test in Europe conference – DATE'11*. ACM, 2011. (Cited on pages 44 and 84.)

[82] L. Gaspar, V. Fischer, L. Bossuet, and M. Drutarovsky. **Cryptographic extension for soft general-purpose processors with secure key management**. In *Proceedings of the conference on Field Programmable Logic and Applications – FPL'11*, pages 500–505. IEEE, 2011. (Cited on pages 44 and 84.)

[83] L. Gaspar, V. Fischer, L. Bossuet, and R. Fouquet. **Secure extensions of FPGA soft core processors for symmetric key cryptography**. In *Proceedings of the workshop on Reconfigurable Communication-centric Systems-on-Chip – ReCoSoC'11*, pages 1–8, 2011. (Cited on pages 44 and 84.)

[84] Altera. *AN 567: Quartus II Design Separation Flow*, 2012. http://www.altera.com/. (Cited on page 50.)

[85] V. Fischer, M. Drutarovsky, P. Chodowiec, and F. Gramain. **InvMixColumn decomposition and multilevel resource sharing in AES implementations**. *IEEE Transactions on Very Large Scale Integration Systems – VLSI'05*, **13**(8):989–992, 2005. (Cited on pages 58 and 98.)

[86] I. Verbauwhede, F. Hoornaert, J. Vandewalle, and H. De Man. **ASIC Cryptographical Processor Based on DES**. In *Proceedings of the conference Euro ASIC'91*, pages 292–295. IEEE, 1991. (Cited on pages 66, 67 and 68.)

[87] H. Kuo and I. Verbauwhede. **Architectural optimization for a 1.82 Gbits/sec VLSI implementation of the AES Rijndael algorithm**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'01*, pages 51–64. Springer, 2001. (Cited on pages 66, 67 and 68.)

[88] I. Verbauwhede, P. Schaumont, and H. Kuo. **Design and performance testing of a 2.29-GB/s Rijndael processor**. *IEEE Journal of Solid-State Circuits*, **38**(3):569–572, 2003. (Cited on pages 66, 67 and 68.)

[89] A. Hodjat and I. Verbauwhede. **High-throughput programmable cryptocoprocessor**. *IEEE Micro magazine*, **24**(3):34–45, 2004. (Cited on pages 66, 67 and 68.)

[90] A. Hodjat and I. Verbauwhede. **Interfacing a high speed crypto accelerator to an embedded CPU**. In *Proceedings of the Asilomar conference on Signals, Systems and Computers – ACSSC'04*, **1**, pages 488–492. IEEE, 2004. (Cited on pages 66, 67 and 68.)

[91] A. Hodjat and I. Verbauwhede. **Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors**. *IEEE Transactions on Computers*, pages 366–372, 2006. (Cited on page 66.)

[92] E. Mosanya, C. Teuscher, H. Restrepo, P. Galley, and E. Sanchez. **Cryptobooster: A reconfigurable and modular cryptographic coprocessor**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'99*, pages 726–726. Springer, 1999. (Cited on pages 66, 67 and 68.)

[93] G. Kuzmanov, G. Gaydadjiev, and S. Vassiliadis. **The molen processor prototype**. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines – FCCM'04*, pages 296–299. IEEE, 2004. (Cited on page 66.)

[94] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. Sousa. **Reconfigurable Cryptographic Processor**. In *Workshop on Circuits, Systems and Signal Processing – CSSP'06*. Citeseer, 2006. (Cited on pages 66, 67 and 68.)

[95] M. Pericas, R. Chaves, G. Gaydadjiev, S. Vassiliadis, and M. Valero. **Vectorized AES core for high-throughput secure environments**. In *Proceedings of the conference on High Performance Computing for Computational Science – VECPAR'08*, pages 83–94. Springer, 2008. (Cited on pages 66, 67 and 68.)

[96] M.Y. Wang, C.P. Su, C.L. Horng, C.W. Wu, and C.T. Huang. **Single- and multi-core configurable AES architectures for flexible security**. *IEEE Transactions on Very Large Scale Integration Systems – VLSI'10*, **18**(4):541–552, 2010. (Cited on page 66.)

[97] C.P. Su, C.L. Horng, C.T. Huang, and C.W. Wu. **A configurable AES processor for enhanced security**. In *Proceedings of the Asia and South Pacific Design Automation Conference – ASP-DAC'05*, pages 361–366. ACM, 2005. (Cited on pages 66, 67 and 68.)

[98] D.D. Hwang, K. Tiri, A. Hodjat, B.C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede. **AES-Based Security Coprocessor IC in 0.18-μm CMOS with Resistance to Differential Power Analysis Side-Channel Attacks**. *IEEE Journal of Solid-State Circuits*, **41**(4):781–792, 2006. (Cited on pages 66, 67 and 68.)

[99] G. Gogniat, T. Wolf, W. Burleson, J.P. Diguet, L. Bossuet, and R. Vaslin. **Reconfigurable hardware for high-security/high-performance embedded systems: the SAFES perspective**. *IEEE Transactions on Very Large Scale Integration Systems – VLSI'08*, **16**(2):144–155, 2008. (Cited on pages 66, 67 and 68.)

[100] B. MuthuKumar and S. Jeevananthan. **Performance Enhanced Co-Processor for Elliptic Curve Cryptography over GF (p)**. *European Journal of Scientific Research*, **68**(4):544–555, 2012. (Cited on page 66.)

[101] M. Morales-Sandoval, C. Feregrino-Uribe, R. Cumplido, and I. Algredo-Badillo. **A reconfigurable GF (2^M) elliptic curve cryptographic coprocessor**. In *Proceedings of the Southern Conference on Programmable Logic – SPL'11*, pages 209–214. IEEE, 2011. (Cited on page 66.)

[102] F. Crowe, A. Daly, T. Kerins, and W. Marnane. **Single-chip FPGA implementation of a cryptographic co-processor**. In *Proceedings of the conference on Field-Programmable Technology – FPT'04*, pages 279–285. IEEE, 2004. (Cited on page 66.)

[103] Y. Eslami, A. Sheikholeslami, P.G. Gulak, S. Masui, and K. Mukaida. **An area-efficient universal cryptography processor for smart cards**. *IEEE Transactions on Very Large Scale Integration Systems – VLSI'06*, **14**(1):43–56, 2006. (Cited on page 66.)

[104] P. Cotret, J. Crenne, G. Goniat, J-P. Diguet, L. Gaspar, and G. Duc. **Distributed security for communications and memories in a multiprocessor architecture**. In *Proceedings of the International Parallel & Distributed Processing Symposium – IPDPS'11*. IEEE Computer Society, 2011. (Cited on page 70.)

[105] ARM. *AMBA Specification, rev. 2.0*, 1999. http://www.arm.com. (Cited on page 77.)

[106] L. Gaspar, V. Fischer, L. Bossuet, and R. Fouquet. **Secure extension of FPGA general purpose processors for symmetric key cryptography with partial reconfiguration capabilities**. *ACM Transactions on Reconfigurable Technology and Systems – TRETS'12*, **5**, September 2012. (Cited on pages 84 and 129.)

[107] P. Kocher, J. Jaffe, and B. Jun. **Introduction to differential power analysis and related attacks**, 1999. http://www.cryptography.com/dpa/technical. (Cited on pages 86 and 88.)

[108] K. Gandolfi, C. Mourtel, and F. Olivier. **Electromagnetic analysis: Concrete results**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'01*, pages 251–261. Springer, 2001. (Cited on page 86.)

[109] J.J. Quisquater and D. Samyde. **ElectroMagnetic analysis (EMA): Measures and counter-measures for smart cards**. In *Proceedings of the conference on Smart Card Programming and Security – E-smart'01*, pages 200–210. Springer, 2001. (Cited on page 86.)

[110] T. Messerges, E.A. Dabbish, and R.H. Sloan. **Investigations of power analysis attacks on smartcards**. In *Proceedings of the USENIX Workshop on Smartcard Technology*, pages 17–17. USENIX Association, 1999. (Cited on page 86.)

[111] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. **Examining smart-card security under the threat of power analysis attacks**. *IEEE Transactions on Computers*, **51**(5):541–552, 2002. (Cited on page 86.)

[112] S. Mangard. **A simple power–analysis (SPA) attack on implementations of the AES key expansion**. In *Proceedings of the conference on Information Security and Cryptology – ICISC'02*, pages 343–358. Springer, 2003. (Cited on page 86.)

[113] S.B. Örs, F. Gurkaynak, E. Oswald, and B. Preneel. **Power-Analysis Attack on an ASIC AES implementation**. In *Proceedings of the conference on Information Technology: Coding and Computing – ITCC'04*, **2**, pages 546–552. IEEE, 2004. (Cited on page 86.)

[114] F.X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J.J. Quisquater. **Power analysis of FPGAs: How practical is the attack?** In *Proceedings of the conference on Field Programmable Logic and Application – FPL'03*, pages 701–710. Springer, 2003. (Cited on page 86.)

[115] F.X. Standaert, S. Örs, J.J. Quisquater, and B. Preneel. **Power analysis attacks against FPGA implementations of the DES**. In *Proceedings of the conference on Field Programmable Logic and Application – FPL'04*, pages 84–94. Springer, 2004. (Cited on page 86.)

[116] F.X. Standaert, S.B. Örs, and B. Preneel. **Power Analysis of an FPGA**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'04*, pages 30–44. Springer, 2004. (Cited on page 86.)

[117] T. Messerges. **Using second–order power analysis to attack DPA resistant software**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'00*, pages 27–78. Springer, 2000. (Cited on pages 88 and 89.)

[118] S. Chari, J. Rao, and P. Rohatgi. **Template attacks**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'02*, pages 51–62. Springer, 2002. (Cited on page 88.)

[119] E. Brier, C. Clavier, and F. Olivier. **Correlation power analysis with a leakage model**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'04*, pages 135–152. Springer, 2004. (Cited on page 88.)

[120] T.H. Le, J. Clédière, C. Canovas, B. Robisson, C. Servière, and J.L. Lacoume. **A proposition for correlation power analysis enhancement**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'06*, pages 174–186. Springer, 2006. (Cited on page 88.)

[121] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. **Mutual information analysis**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'08*, pages 426–442. Springer, 2008. (Cited on page 88.)

[122] S. Mangard. **Hardware countermeasures against DPA–a statistical analysis of their effectiveness**. In *Proceedings of Cryptographers' Track at the RSA conference – CT-RSA'04*, pages 1998–1998. Springer, 2004. (Cited on page 88.)

[123] T. Güneysu and A. Moradi. **Generic side–channel countermeasures for reconfigurable devices**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'11*, pages 33–48. Springer, 2011. (Cited on pages 88 and 89.)

[124] P.C. Kocher. **Leak-resistant cryptographic indexed key update**. *United States Patent 6,539,092 filed on July 2nd, 1999 at San Francisco, CA, USA*, **6**, March 25th 2003. (Cited on page 88.)

[125] P. Kocher. **Design and validation strategies for obtaining assurance in countermeasures to power analysis and related attacks**. In *NIST Physical Security Testing Workshop – Honolulu*, 2005. (Cited on page 88.)

[126] T. Güneysu. **Using Data Contention in Dual–ported Memories for Security Applications**. *Journal of Signal Processing Systems*, pages 1–15, 2010. (Cited on page 89.)

[127] C. Beckhoff, D. Koch, and J. Torresen. **Short–circuits on fpgas caused by partial runtime reconfiguration**. In *Proceedings of the conference on Field Programmable Logic and Applications – FPL'10*, pages 596–601. IEEE, 2010. (Cited on page 89.)

[128] C. Clavier, J.S. Coron, and N. Dabbous. **Differential power analysis in the presence of hardware countermeasures**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'00*, pages 13–48. Springer, 2000. (Cited on page 89.)

[129] J.S. Coron and I. Kizhvatov. **Analysis and improvement of the random delay countermeasure of CHES 2009**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'10*, pages 95–109. Springer, 2010. (Cited on page 89.)

[130] T. Popp and S. Mangard. **Masked dual-rail pre-charge logic: DPA-resistance without routing constraints**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'05*, pages 172–186. Springer, 2005. (Cited on pages 89 and 90.)

[131] M.L. Akkar and C. Giraud. **An implementation of DES and AES, secure against some attacks**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'01*, pages 309–318. Springer, 2001. (Cited on page 90.)

[132] K. Tiri and I. Verbauwhede. **Place and route for secure standard cell design**. *Smart Card Research and Advanced Applications VI*, pages 143–158, 2004. (Cited on page 90.)

[133] K. Tiri, D. Hwang, A. Hodjat, B.C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede. **Prototype IC with WDDL and differential routing–DPA resistance assessment**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'05*, pages 354–365. Springer, 2005. (Cited on page 90.)

[134] M. Bucci, L. Giancane, R. Luzzi, and A. Trifiletti. **Three–phase dual-rail pre–charge logic**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'06*, pages 232–241. Springer, 2006. (Cited on page 90.)

[135] K. Tiri and I. Verbauwhede. **Securing encryption algorithms against DPA at the logic level: Next generation smart card technology**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'03*, pages 125–136. Springer, 2003. (Cited on page 90.)

[136] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev. **Design and analysis of dual–rail circuits for security applications**. *IEEE Transactions on Computers*, **54**(4):449–460, 2005. (Cited on page 90.)

[137] M. Nassar, S. Bhasin, J.L. Danger, G. Duc, and S. Guilley. **BCDL: a high speed balanced DPL for FPGA with global precharge and no early evaluation**. In *Proceedings of the conference on Design, Automation & Test in Europe – DATE'10*, pages 849–854. IEEE, 2010. (Cited on page 90.)

[138] T. Popp and S. Mangard. **Implementation aspects of the DPA–resistant logic style MDPL**. In *Proceedings of the International Symposium on Circuits and Systems – ISCAS'06*, pages 2913–2916. IEEE, 2006. (Cited on page 90.)

[139] G. Piret and J.J. Quisquater. **A differential fault attack technique against SPN structures, with application to the AES and KHAZAD**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'03*, pages 77–88. Springer, 2003. (Cited on page 99.)

[140] S. Guilley, L. Sauvage, J.L. Danger, and N. Selmane. **Fault injection resilience**. In *Proceedings of the workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'10*, pages 77–88. IEEE, 2010. (Cited on page 105.)

[141] T. Barraza. **How to Protect Intellectual Property in FPGAs Devices – Part 1**, 2005. http://eetimes.com/design/programmable-logic/4014780/How-to-Protect-Intellectual-Property-in-FPGAs-Devices-Part-1. (Cited on page 112.)

[142] B. Dipert. **Cunning circuits confound crooks**. *Electronics Design Network – EDN*, **45**(21):103–112, 2000. (Cited on pages 112 and 113.)

[143] S. Skorobogatov and Ch. Woods. **Breakthrough silicon scanning discovers backdoor in military chip**. In *Proceedings of the workshop on Cryptographic Hardware and Embedded Systems – CHES'12*, pages 23–40. Springer, 2012. (Cited on page 113.)

[144] A. Lesea. **jbits & reverse engineering**, 2005. http://groups.google.com/group/comp.arch.fpga/msg/821968d7dcb50277. (Cited on page 114.)

[145] S. Trimberger. **Trusted design in FPGAs**. In *Proceedings of the Design Automation Conference – DAC'07*, pages 5–8. IEEE/ACM, 2007. (Cited on page 114.)

[146] J-B Note and É. Rannaud. **From the bitstream to the netlist**. In *Proceedings of the ACM/SIGDA symposium on Field programmable gate arrays – FPGA'08*, pages 264–264. ACM, 2008. (Cited on page 114.)

[147] S. Trimberger. *Field-programmable gate array technology*. Springer, 1994. (Cited on page 114.)

[148] Xilinx. **Xilinx: Design Security Solutions**, 2012. http://www.xilinx.com/products/technology/design-security/. (Cited on page 114.)

[149] Altera. **Altera: Design Security**, 2012. http://www.altera.com/devices/fpga/stratix-fpgas/about/security/stx-design-security.html. (Cited on page 114.)

[150] A. Lesea. **IP security in FPGAs**. *Xilinx*, 2007. http://direct.xilinx.com/bvdocs/whitepapers/wp261.pdf. (Cited on page 114.)

[151] L. Hong. **Comparison of Embedded Non-Volatile Memory Technologies and Their Applications, white paper**. Kilopass, 2009. http://www.kilopass.com/wp-content/uploads/2010/04/comparison_of_embedded_nvm.pdf. (Cited on page 114.)

[152] W.R. Tonti. **eFuse Design and Reliability**. In *Integrated Reliability Workshop Final Report*, 2008. Available online: http://paris.utdallas.edu/ssiri08/Tonti_SSIRI_eFuse_V2.pdf. (Cited on page 114.)

[153] A. Moradi, A. Barenghi, T. Kasper, and C. Paar. **On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex–II FPGAs**. In *Proceedings of the ACM conference on Computer and communications security – CCS'11*, pages 111–124. ACM, 2011. (Cited on page 114.)

[154] A. Moradi, M. Kasper, and C. Paar. **Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures**. In *Proceedings of Cryptographers' Track at the RSA conference – CT-RSA'12*, pages 1–18. Springer, 2012. (Cited on page 114.)

[155] L. Bossuet, G. Gogniat, and W. Burleson. **Dynamically configurable security for SRAM FPGA bitstreams**. *International Journal of Embedded Systems*, **2**(1):73–85, 2006. (Cited on page 115.)

[156] A. Dauman. **An Open IP Encryption Flow Permits Industry-Wide Interoperability**, 2006. http://www.asicfpga.com/site_upgrade/asicfpga/pds/ip_pds_files/ip_encryption_wp.pdf. (Cited on page 115.)

[157] T. Guneysu, B. Moller, and C. Paar. **Dynamic intellectual property protection for reconfigurable devices**. In *Procedings of the Field-Programmable Technology Conference – ICFPT'07*, pages 169–176. IEEE, 2007. (Cited on page 115.)

[158] S. Drimer, T. Güneysu, M.G. Kuhn, and C. Paar. **Protecting multiple cores in a single FPGA design**. *http://www.saardrimer.com/sd410/papers/protect_many_cores.pdf*, 2008. (Cited on page 115.)

[159] R. Maes, D. Schellekens, and I. Verbauwhede. **A Pay-per-Use Licensing Scheme for Hardware IP Cores in Recent SRAM-Based FPGAs**. *IEEE Transactions on Information Forensics and Security*, **7**(1):98–108, 2012. (Cited on page 115.)

[160] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz. **Serecon: A secure dynamic partial reconfiguration controller**. In *Proceeding of the IEEE Computer Society Annual Symposium on VLSI – ISVLSI'08*, pages 292–297. IEEE, 2008. (Cited on page 115.)

[161] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz. **SeReCon: a secure reconfiguration controller for self-reconfigurable systems**. *International Journal of Critical Computer-Based Systems*, **1**(1):86–103, 2010. (Cited on page 115.)

[162] K Kepa. *Secure Intellectual Property Management in Reconfigurable Computing Systems*. PhD thesis, College of Engineering and Informatics, National University of Ireland, Galway, 2010. (Cited on page 115.)

[163] F. Devic, L. Torres, J. Crenne, B. Badrignans, and P. Benoît. **SecURe DPR: Secure Update Preventing Replay Attacks for Dynamic Partial Reconfiguration**. In *Proceedings of the conference on Field Programmable Logic and Application – FPL'12*, pages 57–62. IEEE, 2012. (Cited on page 116.)

[164] Xilinx Inc. **Developing Secure Designs Using the Virtex-5 Family–XAPP1134**, 2011. (Cited on page 116.)

[165] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine. **Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems**. In *Proceedings of the IEEE Symposium on Security and Privacy – SP'07*, pages 281–295. IEEE, 2007. (Cited on page 118.)

[166] Xilinx Inc. **PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration – XAPP887**, 2012. http://www.xilinx.com/support/documentation/application_notes/xapp887_PRC_EPRC.pdf. (Cited on page 123.)

[167] Xilinx Inc. **Virtex-6 FPGA Configuration User Guide – UG360**, 2011. http://www.xilinx.com/support/documentation/user_guides/ug360.pdf. (Cited on page 123.)

Appendix A: Introduction

A.1 Sécurité Flexible et Implantations Matérielles

La sécurisation des données est aujourd'hui de plus en plus demandée, en particulier avec l'essor des communications sans fil grand public. Les technologies telles que WIFI, Bluetooth, UMTS... sont largement utilisées. De plus, l'utilisation de dispositifs portables monte en flèche. Dans ce secteur très lucratif une solution reconfigurable faible coût combinant un crypto-processeur embarqué (accélérateur matériel) et un générateur de nombres aléatoires est particulièrement attractive. La capacité à garantir une sécurité au niveau de l'implantation physique (concept du cloisonnement entre zones protégée et non-protégée) ainsi que les techniques de contremesures ont permis de passer de l'application standard des produits électroniques à une utilisation dans des domaines plus sensibles comme : l'avionique, l'automobile et les applications militaires qui requièrent des besoins en sécurité plus élevés.

Les systèmes cryptographiques matériels doivent fréquemment implanter des algorithmes parallèles rapides (chiffreurs symétriques, certains modes de chiffrement, etc.) conjointemenet à des algorithmes très séquentiels (autres modes de chiffrement, protocoles cryptographiques, etc.). L'implantation d'algorithmes et protocoles cryptographiques dans le matériel nécessite l'utilisation de nombreuses machines d'états complexes qui rendent la logique très instable et vulnérable. De plus, les mises à jour de logique câblée peuvent devenir compliquées, longues et coûteuses. D'un autre côté, la sécurité du système lui-même et la protection des données confidentielles sont souvent sous-estimées.

La solution la plus courante consiste à utiliser un processeur généraliste combiné à un ou plusieurs coprocesseurs cryptographiques. Cette solution permet d'implanter les algorithmes séquentiels (qui évoluent très fréquemment en raison des attaques et/ou de l'évolution des standards) dans le processeur, et les tâches pouvant s'exécuter en parallèle dans le coprocesseur implanté dans le même composant logique. Cette solution pose cependant quelques questions quant à la sécurité du système : premièrement, le processeur généraliste manipule les clés comme n'importe quelle donnée ordinaire et toute modification (intentionnelle ou non) du contenu de la mémoire programme peut permettre la lecture des clés en clair à l'extérieur du système ; deuxièmement, l'utilisation d'un processeur standard ne permet pas d'isoler efficacement les zones de communication rouges (non protégées) et noires (protégées) à l'intérieur du composant.

L'utilisation d'un processeur généraliste pour des applications cryptographiques conduit à une perte de vitesse pour deux raisons. La première est due à la largeur de bus : la taille du bus et des données (généralement 32 bits) limite considérablement la performance du système. La seconde est due à la complexité du jeu instructions : les

processeurs visant des applications cryptographiques (secondés par un coprocesseur cryptographique) n'ont pas besoin du jeu d'instructions complexe qui est nécessaire aux processeurs généralistes et l'optimisation (réduction) de ce jeu d'instruction peut augmenter la vitesse du système. La conception d'un processeur dédié avec un jeu d'instructions dédié est particulièrement intéressante dans le cas du secteur émergeant des applications systèmes embarqués multi processeurs (Multi-Processor System-on-Chip - MPSoC).

A.2 Objectifs de la Thèse

Afin de remplir les conditions mentionnées (souvent contradictoires) pour le système cryptographique, l'Agence Nationale de la Recherche (ANR) a sélectionné le projet intitulé *Systèmes sur puces reconfigurables pour la sécurisation de données* (SecReSoC) proposé par l'équipe *Systèmes Embarqués Sécurisés* du laboratoire Hubert Curien. Le but du projet SecReSoC était d'étudier les aspects sécurité d'un MPSoC et de réaliser un démonstrateur du système MPSoC final implanté dans un composant FPGA (Field Programmable Gate Array). Le travail qui est présenté dans ce document a été réalisé dans le cadre du projet ANR SecReSoC – ANR-09-SEGI-013. Les objectifs envisagés sont les suivants :

1. Réaliser une analyse des architectures de processeurs embarqués de type software (softcores) existantes afin d'étudier dans quelle mesure l'architecture du crypto-processeur peut s'appuyer sur la transformation d'une architecture existante.

2. Proposer une nouvelle architecture du processeur permettant de séparer les registres de données et les registres de clés confidentielles.

3. Optimiser l'architecture de l'Unité Arithmétique et Logique (UAL) pour les opérations utilisées dans les protocoles cryptographiques et les modes de chiffrement.

4. Proposer un jeu d'instructions permettant de traiter séparément des données et des clés.

5. Intégrer le noyau de chiffrement et les outils de contremesures permettant de lutter contre les différents types d'attaques dans l'architecture du crypto-processeur.

6. Proposer un outil de programmation (compilateur) paramétrable, permettant de faire évoluer le langage assembleur du processeur avec son jeu d'instructions.

7. Intégrer les éléments de reconfiguration et programmation sécurisés dans la structure logique du crypto-processeur.

8. Évaluer le niveau de sécurité du crypto-processeur proposé.

A.3 Contribution

Ce travail ne se limite pas aux objectifs susmentionnés mais propose en plus de nouveaux principes, de nouvelles règles de conception, des analyses de sécurité et de nouvelles architectures dédiées. Ceci permet la gestion sécurisée des clés de chiffrement dans des processeurs cryptographiques ou même dans des processeurs généralistes (associés à un coprocesseur cryptographique sécurisé). Les processeurs cryptographiques développés (nommés par la suite simplement crypto-processeurs) et les coprocesseurs cryptographiques (nommés par la suite crypto-coprocesseurs) peuvent être utilisés directement dans la pratique. Ce travail est complété par une étude des contremesures aux attaques par canaux cachés puis par la protection du système lors des reconfigurations totales ou partielles. Ce travail ne se contente pas de solutions partielles mais propose un système complet avec un processeur cryptographique HCrypt entièrement fonctionnel et testé ainsi qu'un coprocesseur cryptographique HCrypt-C disponibles dans de nombreuses variantes dédiées à des objectifs spécifiques.

Chaque problématique abordée dans ce travail est précédée d'une étude approfondie de l'état de l'art du domaine qui est résumée en début de chaque chapitre de cette thèse.

A.4 Structure de la Thèse

Le chapitre 2 présente les connaissances de base que doit avoir le lecteur pour une lecture compréhensible des prochaines parties de ce document. Ce chapitre présente d'abord les techniques cryptographiques de base pour assurer une communication sécurisée. Puis il décrit les principaux algorithmes de cryptographie symétrique (DES, AES, modes de chiffrement, etc.). Ceci est suivi par une description des fonctions de hachage ainsi que des générateurs de nombres aléatoires. Puis par une déscription de la gestion des clés, des différents acteurs de la communication et des principales menaces de sécurité. Les processus importants d'échange des clés sont brièvement introduits et comparés. La deuxième partie du chapitre 2 se focalise sur les FPGAs ainsi que sur les processus de conception avancés correspondants.

Le chapitre 3 contient un résumé de l'état de l'art des processeurs cryptographiques. Il est d'abord question des problèmes de sécurité dans les implantations d'algorithmes cryptographiques, suivi d'une vue d'ensemble des architectures cryptographiques matérielles. Ensuite, sont présentées les nouvelles règles de séparation pour la gestion des clés de sécurité. Ces règles sont implantées dans le crypto-processeur HCrypt initial. Par la suite, son architecture matérielle et logicielle ainsi que le protocole de communication sont décrits, suivis par les simulations et les tests matériels. Les propriétés du HCrypt et les résultats d'implantation sont présentés.

Le chapitre 4 décrit les coprocesseurs cryptographiques avec gestion sécurisée des clés. Après présentation de l'état de l'art des crypto-coprocesseurs, de légères modifications des règles de séparation pour crypto-coprocesseurs sont proposées. Ces règles de séparation sont appliquées sur le crypto-coprocesseur HCrypt-C. Sont en-

suite présentées les topologies d'interface les plus courantes pour les interconnections entre un coprocesseur et un processeur standard. Ensuite, trois processeurs généralistes différents sont associés au HCrypt-C, avec présentation des architectures et des résultats.

Les menaces d'attaques par canaux cachés (SCA) sont étudiées dans le chapitre 5. Après avoir introduit l'état de l'art des attaques matérielles, une analyse de sécurité approfondie du chiffreur AES et du HCrypt est présentée. L'analyse de sécurité du HCrypt fait ressortir une vulnérabilité aux SCA, c'est pourquoi le crypto-processeur HCrypt2 est envisagé. HCrypt2 met en oeuvre de nouvelles contremesures (contremesures zéro-coût). Les analyses de sécurité du HCrypt2 confirment l'amélioration de la robustesse aux SCA. L'architecture et les résultats du HCrypt2 sont présentés et évalués.

Dans le chapitre 6 une technique avancée spécifique aux FPGA est évaluée : la reconfiguration partielle. Ce chapitre présente tout d'abord les avantages et inconvénients de la reconfiguration partielle, puis le travail accompli dans ce domaine. Ensuite, des nouvelles règles de séparation en accord avec la reconfiguration partielle sont définies. Les implications en terme de sécurité des reconfigurations totales et partielles sont discutées. Enfin, un crypto-processeur HCrypt-C partiellement reconfigurable est décrit et testé. Cette reconfiguration partielle concerne la zone du chiffreur décrite selon trois variantes.

Enfin, le chapitre 7 résume toutes les contributions importantes, les propositions et les conclusions. Il présente également les nouveaux défis qui pourraient être sujets d'études futures.

Appendix B: Aperçu des Contributions et Conclusions

Cette thèse présente le travail de recherche qui a été conduit dans le but d'accroître la sécurité de la gestion des clés réalisée dans les systèmes sur puce multi-processeurs. L'objectif principal était de proposer une nouvelle architecture crypto-processeur organisée de façon à ce que le stockage des clés soit fermement séparé du reste du système. Un nouveau jeu d'instructions a été proposé pour faciliter la séparation des données et des opérations de traitement des clés. De plus, le crypto-processeur doit être optimisé pour les modes de chiffrement par blocs et doit permettre d'échanger les données et les clés par paquets. Une des versions au moins du crypto-processeur doit intégrer des contremesures aux attaques par canaux cachés. La reconfiguration pouvant elle aussi être source de faiblesse, sa sécurité doit être évaluée et des solutions proposées. Pour simplifier les développements logiciels pour le nouveau crypto-processeur, un outil assembleur était nécessaire. De plus, les fréquentes modifications du jeu d'instructions du crypto-processeur nécessitent un assembleur flexible. Le dernier objectif était d'évaluer le niveau de sécurité du crypto-processeur proposé.

Ce travail de recherche atteint les objectifs mentionnés, et propose en complément différentes solutions aux problèmes variés rencontrés au cours de ces trois années de thèse.

B.1 Aperçu des Contributions

Les contributions les plus importantes de ce travail de recherche sont les suivantes :

1. *Proposition d'un ensemble de règles de séparation* qui permet une gestion sécurisée des clés dans les processeurs et coprocesseurs cryptographiques

2. *Développement du crypto-processeur HCrypt* qui suit ces règles de séparation. De plus, HCrypt résiste aux attaques logicielles ou par injection de fautes ciblant une révélation des clés secrètes

3. *Développement de l'assembleur FlexASM* avec un jeu d'instructions défini dans un fichier texte

4. *Proposition d'un crypto-coprocesseur* qui permet une gestion sécurisée des clés pour chaque processeur généraliste

5. *Proposition de nouvelles contremesures à bas coût* qui augmentent la sécurité du crypto-processeur contre les attaques par canaux cachés

6. *Proposition d'une gestion sécurisée des clés dans un système entièrement ou partiellement reconfigurable*

Toutes ces contributions exigeaient une étude appronfondie des architectures existantes, une analyse de la sécurité et nécessitaient un grand nombre d'implantations, de simulations et de tests matériels.

B.2 Conclusions

Les contributions et propositions principales présentées dans cette thèse concernent la définition d'un ensemble de règles de séparation qui aident les développeurs à la création de processeurs et coprocesseurs cryptographiques avec gestion sécurisée des clés. Si ces règles sont respectées, le système deviendra robuste aux attaques logicielles ou par injections de fautes visant à une révélation des clés sécurisées. Les règles de séparation suggèrent une division du système en trois zones de niveaux de securite differents : la zone de données, la zone du chiffreur et la zone des clés. Ces trois zones doivent être séparées au niveau du protocole, du système, de l'architecture et au niveau physique. En conséquence, les clés sécurisées qui sont enregistrées dans la zone des clés ne peuvent pas être directement transmises dans la zone de données. En revanche, elles doivent passer par un chiffreur (représentant une barrière physique) dans la zone du chiffreur. Ainsi, toutes les attaques logicielles ou par injection de fautes ne peuvent pas révéler les clés sécurisées.

These unique separation rules were demonstrated on the novel HCrypt cryptoprocessor. HCrypt is a 128-bit cryptographic processor featuring the ALU, which is optimized for block cipher modes and packet management, two AES ciphers, true random number generator, and secure key storage for both session and master keys. HCrypt is divided into data, cipher and key zone each having different security privilege. The most secure is the key zone, where secret keys are stored in clear. HCrypt protects data by using the generated session keys. The session keys must be securely exchanged prior to data exchange using the master keys. Master keys are transferred to HCrypt via a dedicated 32-bit bus. We demonstrated HCrypt operation, security, and programmability and measured its throughput. Tests in Xilinx Virtex-5 and -6 FPGAs indicated data throughput of 824.7 Mb/s, while HCrypt exchanged session keys and processed data in packets using the 128-bit CFB block encryption mode.

Ces règles uniques de séparation ont été appliquées à un nouveau crypto-processeur intitulé HCrypt. HCrypt est un processeur cryptographique de 128 bits contenant l'UAL qui a été optimisée pour les modes de chiffrement et la gestion de paquets. De plus, HCrypt comporte deux chiffeurs AES, un générateur de nombres véritablement aléatoires et des mémoires sécurisées pour les clés de sessions et clés maîtres. HCrypt est divisé en trois zones (données, chiffreur et clés) chacune avec un niveau de sécurité différent. La zone la plus sécurisée est la zone des clés. Dans cette zone les clés sécurisées peuvent être enregistrées en clair. HCrypt protège les données en utilisant les clés de session éphémères. Les clés de session doivent être échangées

de manière sécurisée avant l'échange de données utilisant les clés maîtres. Les clés maîtres sont transmises par un bus de 32 bits dédié. Un demonstrateur du HCrypt a été réalisé, avec évaluation de sa sécurité et sa programmabilité et mesure du débit. Les tests, effectués dans les FPGAs Xilinx Virtex-5 et -6, atteignent un débit de données de 824.7 Mb/s (comprenant l'échange de clés et le chiffrement des données avec le mode CFB 128 bits).

Un nouvel outil assembleur FlexASM a été créé pour simplifier le développement logiciel du HCrypt. Contrairement aux autres assembleurs, le jeu d'instructions peut être défini dans un fichier texte chargé avant la compilation du code assemblé. Cette caractéristique résout le problème des modifications fréquentes du jeu d'instructions du HCrypt pendant son développement. FlexASM peut cependant être utilisé pour d'autres processeurs.

Nous montrons ensuite la façon de construire les crypto-processeurs supportant la gestion des clés. Pour les applications à coût critique nécessitant l'exécution d'instructions généralistes, nous proposons l'extension du crypto-coprocesseur HCrypt-C à n'importe quel processeur généraliste. Ce crypto-coprocesseur respecte toutes les règles de séparation strictes, permettant ainsi aux processeurs généralistes de gérer les clés. La structure du crypto-coprocesseur HCrypt-C est très similaire à celle du crypto-processeur HCrypt. Il a un bus de données de 128 bits, un chiffreur/déchiffreur AES embarqué et un TRNG. Le crypto-coprocesseur HCrypt-C peut être interconnecté avec un processeur utilisant un bus processeur interne, un bus coprocesseur dédié ou un bus périphérique. Ces trois sortes de bus sont comparées, évaluées sur les trois implantations de processeurs et testées dans le matériel. Le processeur Nios II atteint un débit de données de 25.1 Mb/s, le MicroBlaze 18.4 Mb/s et le Cortex M1 12.2 Mb/s, tous échangeant les clés de session et traitant les données par paquets en utilisant le mode de chiffrement par blocs CFB.

Dans le but d'augmenter la robustesse aux attaques par canaux cachés, de nouvelles contremesures zéro-coût sont proposées et testées dans un nouveau HCrpt2. Contrairement aux contremesures niveau microarchitecture, les contremesures zéro coût sont implantées à un niveau macroarchitecture. Nous montrons que par un réarrangement astucieux des blocs du cryptoprocesseur (tout en maintenant l'isolation physique des zones de sécurité selon les règles de séparation proposées), le nouveau crypto-processeur HCrypt2 est robuste aux attaques logicielles, et plus robuste contre les DPA et les attaques par injection de fautes. De plus, nous avons montré qu'un seul des deux chiffreurs a besoin d'être protégé contre les attaques, ce qui rend encore moins couteuse la protection du HCrypt2 que celle du HCrypt1 en utilisant les mêmes techniques de masquage des données. De plus, le chiffreur protégeant les clés de session est rarement utilisé (une seule fois par paquet). Ce chiffreur n'a donc pas besoin d'être rapide et peut donc être très petit, ce qui diminue encore plus le coût de cette contremesure au niveau microarchitecture. En plus de ceci, seul un registre clé de 128 bits est nécessaire dans HCrypt2 (pour le chiffrement) par rapport aux deux registres nécessaires au HCrypt1 (un pour le chiffrement et un pour le déchiffrement).

La dernière contribution est l'extension des règles de séparation aux systèmes

partiellement reconfigurables. Deux cas sont étudiés : a) les FPGA permettant uniquement la reconfiguration totale de la structure logique ; b) les FPGA permettant la reconfiguration partielle. Pour maintenir un haut niveau de sécurité, le crypto-processeur doit contenir une unité de contrôle de la reconfiguration, un stockage des clés sécurisé et une interface de configuration interne. L'unité de contrôle de reconfiguration doit être câblée (processeur sécurisé embarqué) dans le cas d'un système totalement reconfigurable, ou stockée dans la zone statique dans le cas des systèmes partiellement reconfigurables. Le système partiellement reconfigurable a été implanté à l'aide d'un processeur MicroBlaze associé au crypto-processeur HCrypt-C. Les zones processeur et clé sont considérées statiques, alors que le chiffreur est reconfigurable. Pour le test, trois versions de chiffreur ont été testées : AES (chiffreur, déchiffreur et TRNG), DES et une boite noire vide. La taille du bitstream partiel de chacune des 3 versions du HCrypt-C est de 530.25 kB.

B.3 Perspectives

Dans le cadre de ce travail, nous avons également voulu proposer un processeur cryptographique qui peut être intégré dans le Multiprocesseur System on Chip SecReSoC. La finalisation du système avec sa démonstration est un autre défi. La sécurité relative aux contremesures à faible coût doit être éprouvée par des attaques DPA réelles.

Les deux versions de HCrypt-C (statique et reconfigurable) présentées ici n'incluent pas les contre-mesures faible coût. Pour cette raison, HCrypt2-C comportant ces contre-mesures doit être conçu. Un tel crypto-coprocesseur offrira une meilleur protection contre les attaques par canaux auxiliaires et les attaques par injection de fautes.

Un autre défi concerne l'échange des clés maîtres à long terme. En effet, un secret initial est fondamental pour chaque système cryptographique. Toutefois, le processus d'établissement des clés maîtres dépend strictement du système cible. Dans le cas de notre étude, un système entièrement symétrique avec une distribution manuelle des clés maîtres a été suffisant.

Finalement, un défi intéressant pourrait se manifester dans l'intégration d'une unité cryptographique asymétrique qui peut être utilisée pour la distribution manuelle des clés maîtres et pour les signatures des données dans les paquets. Néanmoins, il est crucial de maintenir un niveau de sécurité élevé.

Appendix C: Résumé de Thèse

C.1 Chapitre 1 : Introduction

La sécurisation des données est aujourd'hui de plus en plus demandée, en particulier avec l'essor des communications sans fil grand public. La capacité à garantir une sécurité au niveau de l'implantation physique (le concept du cloisonnement entre zone protégée et non-protégée) ainsi que les techniques des contremesures ont permis de passer de l'application standard des produits électroniques à une utilisation dans des domaines plus sensibles comme : l'avionique, l'automobile et les applications militaires qui requièrent des besoins en sécurité plus élevé.

Les systèmes cryptographiques matériels doivent fréquemment implanter des algorithmes parallèles rapides (c.-à.-d. chiffreurs symétriques, quelques modes de chiffrement, etc.) avec des algorithmes très séquentiels (c.-à.-d. modes de chiffrements, protocoles cryptographiques). Dans la plupart des cas, la logique séquentielle complexe conduit à utiliser des machines d'état complexes ou exige la présence d'un processeur ou coprocesseur cryptographique. Par contre, la sécurité du système lui-même et la protection des données confidentielles sont souvent sous-estimées. Premièrement, le processeur généraliste manipule les clés comme les données standards. Par conséquent, une modification des contenus de la mémoire du programme (intentionnelle ou non-intentionnelle) peut rendre possible une lecture des clés en clair en dehors du système. Deuxièmement, l'utilisation des processeurs généralistes ne permet pas d'isoler efficacement les zones de communication rouges (non-protégées) des zones noires (protégées) à l'intérieur de l'appareil. Troisièmement, en ce qui concerne l'utilisation du processeur généraliste en cryptographie, on considère les problèmes suivants : ce sont la limitation de la vitesse (chemin de données de 32 bits, Unité arithmétique et logique non-optimisée pour la cryptographie) et la complexité du jeu d'instructions. Ainsi, la conception des processeurs dédiés avec un jeu d'instructions dédié est particulièrement intéressant dans le cas des champs d'application émergents basés sur le Multiprocesseur System on Chip (MPSoC).

C.1.1 Objectifs de la thèse

Afin de remplir les conditions mentionnées (souvent contradictoires) pour le système cryptographique, l'Agence Nationale de la Recherche (ANR) a sélectionné le projet intitulé «Systèmes sur puces reconfigurables pour la sécurisation de données» (SecReSoC). Le travail qui est présenté dans ce document a été réalisé dans le cadre du projet ANR SecReSoC – ANR-09-SEGI-013. Les objectifs envisagés sont les suivants :

1. Réaliser une analyse des architectures de processeurs embarqués de type software (softcores) existantes afin d'étudier dans quelle mesure l'architecture du crypto-processeur peut s'appuyer sur la transformation d'une architecture existante.

2. Proposer une nouvelle architecture du processeur permettant de séparer les registres de données et les registres de clés confidentielles.

3. Optimiser l'architecture de l'Unité Arithmétique et Logique (UAL) pour les opérations utilisées dans les protocoles cryptographiques et les modes de chiffrement.

4. Proposer un jeu d'instructions permettant de traiter séparément des données et des clés.

5. Intégrer le noyau de chiffrement et les outils de contremesures permettant de lutter contre les différents types d'attaques dans l'architecture du crypto-processeur.

6. Proposer un outil de programmation (compilateur) paramétrable, permettant de faire évoluer le langage assembleur du processeur avec son jeu d'instructions.

7. Intégrer les éléments de reconfiguration et programmation sécurisés dans la structure logique du crypto-processeur.

8. Évaluer le niveau de sécurité du crypto-processeur proposé.

C.1.2 Contribution

Ce travail de recherche est motivé par des besoins pratiques. Le processeur et coprocesseur cryptographique proposé fait partie d'un système réel utilisé dans le secteur bancaire. Ce travail n'envisage pas seulement les objectifs susmentionnés mais propose en plus des nouveaux principes, les règles de conception, les analyses et les architectures permettant la gestion sécurisée des clés de chiffrement sur des processeurs cryptographiques ou même sur des processeurs généralistes (élargi par un coprocesseur cryptographique sécurisé). Une partie du travail concerne aussi les contremesures contre les attaques par canaux cachés et se réfère aussi à la protection du système totalement ou partiellement reconfigurable. Le système proposé a été testé et il est entièrement fonctionnel.

C.2 Chapitre 2 : Approche Théorique et Technologique

Ce chapitre présente les connaissances de base que doit avoir le lecteur pour une lecture compréhensible des prochaines parties de ce document. D'abord, les techniques cryptographiques de base pour assurer une communication sécurisée sont décrites. Ensuite, les principaux algorithmes de cryptographie symétrique (c.-à.-d.

DES, AES, modes de chiffrement, etc.) sont décrits. Ceci est suivi par une description des fonctions de hachage ainsi que des générateurs de nombres aléatoires. Par la suite, la gestion des clés, les différents acteurs de la communication et les principales menaces de sécurité seront présentés. Ainsi, les processus importants d'établissement des clés sont brièvement introduits et comparés. La deuxième partie du chapitre 2 se focalisera sur les FPGAs ainsi que sur les processus de conception avancés correspondants.

C.3 Chapitre 3 : Crypto-processeur avec une Gestion Sécurisée des Clés

Ce chapitre est consacré à démontrer pourquoi les implantations logicielles de cryptosystèmes doivent être protégées au niveau matériel afin d'être sécurisée contre les attaques logicielles. Puis, les architectures matérielles cryptographiques actuelles seront présenté avec une description de leurs avantages et inconvénients. Les nouvelles règles de séparation pouvant permettre à un processeur cryptographique de supporter une gestion sécurisée des clés sera également proposée. Ces nouvelles règles de séparation sont mises en evidence dans le nouveau crypto-processeur HCrypt.

HCrypt est un processeur cryptographique de 128 bits possédant une UAL qui est optimisée pour des modes de chiffrement, la gestion de paquets, les deux chiffrements AES, les générateurs de nombres véritablement aléatoires et le stockage sécurisé des clés de session et des clés maîtres. HCrypt est divisé en zones : de données, de chiffrement et de clés. Chaque zone est caractérisée par un niveau de sécurité qui lui est propre. La zone la plus sécurisée constitue la zone dans laquelle les clés secrètes sont enregistrées en clair. HCrypt protège les données utilisant les clés de session générées à partir de la clé maître. Les clés de session doivent être échangées de manière sécurisée avant l'échange de données utilisant les clés maîtres. Les clés maîtres sont transférées au HCrypt à travers un bus dédié de 32 bits. L'opérabilité, la sécurité et la programmabilité du HCrypt sont démontrées et le débit a été mesuré. Les tests effectués dans les FPGAs Xilinx Virtex-5 et -6 indiquent que le débit s'élève à 824,7 Mb/s pendant que HCrypt échange les clés de session et chiffre des données par paquets en utilisant le mode de chiffrement CFB de 128 bits.

Afin de simplifier le développement du logiciel-HCrypt, un nouvel outil assembleur FlexASM a été créé. Contrairement aux autres outils assembleurs, le jeu d'instructions est défini dans un fichier texte à partir duquel le jeu d'instructions sera chargé avant la compilation d'un code assembleur. Ces choix permettent de résoudre le problème concernant les modifications fréquentes du jeu d'instructions du HCrypt pendant son développement. En outre, le FlexASM peut également être utilisé dans d'autres processeurs.

C.4 Chapitre 4 : Crypto-coprocesseur avec une Gestion Sécurisée des Clés

Les règles de séparation, présentées pour le crypto-processeur HCrypt, peuvent également être adaptées pour le crypto-coprocesseur. Nous envisageons d'expliquer la façon dont un crypto-coprocesseur supportant une gestion sécurisée des clés peut être créé. Pour les applications dont les coûts jouent un rôle primordial et qui exige aussi l'exécution d'instructions généralistes, nous proposons d'ajouter le crypto-coprocesseur HCrypt-C au processeur généraliste. le crypto-coprocesseur mentionné, respecte toutes les règles de séparation ; ceci permet ainsi une gestion sécurisée des clés pour les processeurs généralistes.

L'architecture du crypto-coprocesseur HCrypt-C se montre très similaire à celle du HCrypt. HCrypt-C est divisé en zone de données, zone du chiffreur et zone des clés. Il est composé d'un chemin de données de 128 bits, d'un chiffreur AES embarqué, d'un déchiffreur et d'un TRNG. Il est possible d'interconnecter le crypto-coprocessor HCrypt-C avec un processeur utilisant un bus situé à l'intérieur du processeur, un bus dédié à un coprocesseur ou un bus périphérique. Les trois types de bus ont été comparés, utilisés dans les trois implantations du processeur et ont été testés dans le matériel. Les débits de données pour les versions sécurisées des processeurs s'élévent à 25,1 Mb/s pour le NIOS II, 18,4 Mb/s pour le MicroBlaze, et 12,2 Mb/s pour le Cortex M1. Les débits sont mesurés pendant que chaque système échange les clés de sessions et traite les données par paquets en utilisant le mode de chiffrement CFB de 128 bits.

C.5 Chapitre 5 : Protection des Crypto-processeurs contre les SCA au niveau Macroarchitecture

Dans la plupart des cas, les algorithmes cryptographiques contemporains connaissent une bonne protection contre la cryptanalyse linéaire, différentielle ou algébrique. En effet, les problèmes apparaissent lors d'une implantation de ces algorithmes dans les appareils électroniques. Principalement, les processus physiques qui dépendent des calculs des algorithmes cryptographiques à l'intérieur de l'appareil peuvent partiellement être observés à l'extérieur de l'appareil et peuvent être considérés comme une fuite d'information exploitable. En conséquence, de nouvelles méthodes doivent constamment être élaborées afin de minimiser les conséquences de ces phénomènes physiques et protéger ainsi l'information sensible enregistrée et traitée dans les appareils cryptographiques.

Dans ce chapitre, nous expliquons le principle des attaques par canaux cachés. De plus, nous présentons les études qui ont été menées sur la protection des circuits cryptographiques contre les attaques par canaux cachés. Ensuite, une analyse détaillée de la sécurité du chiffreur AES sera donnée. Cette analyse est fondamentale pour l'analyse de sécurité de HCrypt. Contrairement au modèle de menace qui a été mentionné dans les chapitres précédents, l'analyse de sécurité du HCrypt ne repose

pas seulement sur les attaques logicielles, mais aussi sur les attaques par canaux cachés et par injection de fautes. Cette analyse débouche sur la proposition de nouvelles contremesures à faible coût pour le crypto-processeur et le crypto-coprocesseur.

Ces nouvelles contremesures sont proposées dans la nouvelle version de HCrypt : HCrypt2. Contrairement aux contremesures trouvées au niveau microarchitecture, ces nouvelles contremesures sont implantées au niveau macroarchitecture. Nous prouvons ainsi, grâce à un regroupement astucieux des blocs crypto-processeur (tandis que l'on assure l'isolation physique des zones de sécurité en fonction des règles de séparation proposées), que cette nouvelle version du crypto-processeur HCrypt2 se montre robuste contre les attaques logicielles, la DPA et les attaques par injection de fautes. En outre, nous avons mis en évidence le fait qu'il faut seulement protéger un des deux chiffreurs contre les attaques. Ainsi, les coûts pour une protection de HCrypt2 sont considérablement plus bas que ceux pour protéger HCrypt1 en utilisant les mêmes techniques de masquage des données ou de dissimulation des données. De plus, le chiffreur protégeant les clés de session est rarement utilisé (seulement une fois par paquet). Par conséquent, il est possible d'avoir un chiffreur plus petit. Enfin, uniquement un seul registre des clés maîtres de 128 bits dans HCrypt2 est exigé (seulement pour chiffrer) quand cela est comparé avec deux registres des clés maîtres de 128 bits dans le cas de HCrypt1 (pour chiffrer et déchiffrer). Pour ces raisons, que les contremesures (au niveau microarchitecture) sont encore moins coûteuses.

C.6 Chapitre 6 : Reconfiguration Partielle des Crypto-processeurs

Dans les chapitres précédents, nous avons considéré uniquement la possibilité qu'un FPGA est configuré seulement pendant le démarrage. Cependant, si nous envisageons le cas d'une reconfiguration du FPGA pendant son exécution, de nouvelles attaques peuvent se réaliser non seulement pendant la phase de reconfiguration du FPGA, mais aussi avant le démarrage quand le bitstream est stocké à l'extérieur du FPGA. Ces aspects posent alors de nouveaux défis par rapport aux règles de séparation ainsi qu'à la gestion sécurisée des clés.

Dans un premier temps, ce chapitre veut familiariser le lecteur avec les tendances actuelles que l'on trouve dans le domaine de la recherche sur la reconfiguration du FPGA, en se focalisant sur la sécurité du bitstream. Ensuite, l'extension des règles de séparation pour le système partiellement reconfigurable sera décrit. Nous considérons les deux cas différents suivants : a) le FPGA ne permettant qu'une reconfiguration totale de la logique et b) le FPGA supportant une technologie de reconfiguration partielle. Afin de maintenir une sécurité élevée, le crypto-processeur doit inclure une unité de contrôle pour la reconfiguration, un stockage des clés sécurisées ainsi qu'une interface interne pour la configuration. L'unité de contrôle pour la reconfiguration doit être câblée (p. ex. le processeur sécurisé embarqué) dans le cas d'un système entièrement reconfigurable ou implanté dans le secteur statique s'il s'agit

d'un système partiellement reconfigurable. Le système partiellement reconfigurable a été implanté en utilisant le processeur MicroBlaze avec le crypto-coprocesseur HCrypt-C. Les zones du processeur et des clés se trouvent dans le secteur statique et la zone du chiffreur se situe dans le secteur reconfigurable. Pour être capable de réaliser les tests, les trois versions de la zone du chiffreur ont été testées : AES (contenant le chiffreur AES, le déchiffreur et le TRNG), le DES, et la boîte noire vide. La taille du bitstream partiel pour les trois HCrypt-C s'élève à 530,25 kB.

C.7 Chapitre 7 : Résumé de la Contribution et des Conclusions

Ce travail de recherche n'atteint pas seulement les objectifs mentionnés, mais intègre en plus différentes études annexes et solutions dûes aux autres problèmes variés rencontrés au cours de ces trois années de thèse.

C.7.1 Aperçu des Contributions

Les contributions les plus importantes de ce travail de recherche sont les suivantes :

1. *Proposition d'un ensemble de règles de séparation* qui permet une gestion sécurisée des clés dans les processeurs et coprocesseurs cryptographiques
2. *Développement du crypto-processeur HCrypt* qui suit les règles de séparation. De plus, HCrypt résiste aux attaques logicielles ou par injection de fautes ciblant une révélation des clés secrètes
3. *Développement de l'assembleur FlexASM* avec le jeu d'instructions défini dans un fichier de texte
4. *Proposition d'un crypto-coprocesseur* qui permet une gestion sécurisée des clés pour chaque processeur généraliste
5. *Proposition de nouvelles contremesures à bas coût* qui augmentent la sécurité du crypto-processeur contre les attaques par canaux cachés
6. *Proposition d'une gestion sécurisée des clés dans un système entièrement ou partiellement reconfigurable*

Toutes ces contributions exigeaient des études étendues au niveau des architectures existantes, de l'analyse de sécurité et nécessitaient un grand nombre d'implantations, de simulations et de tests matériels.

C.7.2 Perspectives

Dans le cadre de ce travail, nous avons également voulu proposer un processeur cryptographique qui peut être intégré dans le Multiprocesseur System on Chip SecReSoC. La finalisation du système avec sa démonstration est un autre défi. La

sécurité relative aux contremesures à faible coût doit être éprouvée par des attaques DPA réelles.

Les deux versions de HCrypt-C (statique et reconfigurable) présentées ici n'incluent pas les contre-mesures faible coût. Pour cette raison, HCrypt2-C comportant ces contre-mesures doit être conçu. Un tel crypto-coprocesseur offrira une meilleur protection contre les attaques par canaux auxiliaires et les attaques par injection de fautes.

Un autre défi concerne l'échange des clés maîtres à long terme. En effet, un secret initial est fondamental pour chaque système cryptographique. Toutefois, le processus d'établissement des clés maîtres dépend strictement du système cible. Dans le cas de notre étude, un système entièrement symétrique avec une distribution manuelle des clés maîtres a été suffisant.

Finalement, un défi intéressant pourrait se manifester dans l'intégration d'une unité cryptographique asymétrique qui peut être utilisée pour la distribution manuelle des clés maîtres et pour les signatures des données dans les paquets. Néanmoins, il est crucial de maintenir un niveau de sécurité élevé.