HIGH PERFORMANCE VLSI ARCHITECTURES IMPLEMENTING STRONG CRYPTOGRAHIC PRIMITIVES

THÈSE N° 2561 (2002)

PRÉSENTÉE À LA FACULTÉ SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

DANS LE DOMAINE D'ÉLECTRICITÉ

PAR

Feth Allah CHERIGUI

ingénieur électricien diplômé EPF

et de nationalité algérienne

acceptée sur proposition du jury:

Prof. D. Mlynek, directeur de thèse
Prof. Y. Leblebici, rapporteur
Prof. Ch. Piguet, rapporteur
Dr A. Vachoux, rapporteur
M. P. Mercier, rapporteur

Lausanne, EPFL
2002

ACKNOWLEDGEMENTS

This work has been accomplished during the years 1999 to 2001 at the Integrated System Laboratory (ISL) of the Swiss Federal Institute of Technology (EPFL) at Lausanne, Switzerland.

Firstly I wish to express my gratitude to my advisor Professor Mlynek (EPFL) for giving me the opportunity of pursuing a PhD research in his laboratory. I am thankful to Prof. Yusuf Leblebici (WPI/EPFL), Prof. Christian Piguet (CSEM) and Dr. Alain Vachoux (Antrim Design Systems), and Mr. Philippe Mercier (TranSwitch Europe) who do me the honour of accepting to take part in the thesis jury.

I express my thanks to people of ISL who were working with me in the RPK Project: Stephan Bories, Antonio Romeo and Geovani Romolotti. I am grateful to Dr. Alain Vachoux for giving me the necessary packages and sub-micron technology libraries.

Special thanks go to Dr. Paul Debefve for his availability and willingness to solve the numerous CAD-problems that arose during the years, and his assistance for providing the necessary literature. Sincere acknowledgements go to my colleagues of the ISL: Yves Pizzuto, Sergio Lopez, Alexandre Schmid, Leonel Comminilli and Seong Garin for their friendship. I would like to address a further acknowledgements, specially to people who contribute to this work: Alexandre Fotinos, Luc Piguet, Silvain Aguirre and Olivier Croset, many thanks for their valuable work.

Last but certainly not least, I wish to express my deepest gratitude to my parents who enabled me to pursue all my studies and to Monika who did support me for all these years of study. This thesis would have certainly not been accomplished without her help and her love.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ..... **II**
TABLE OF CONTENTS ..... **IV**
LIST OF FIGURES ..... **XII**
LIST OF TABLES ..... **XVI**
ACRONYMS ..... **XVIII**
ABSTRACT ..... **XX**
RÉSUMÉ ..... **XXII**
CHAPTER 1 INTRODUCTION **24**
1.1 MOTIVATION 24
1.2 CURRENT AND FUTURE TRENDS FOR CRYPTOGRAPHY 25
1.2.1 Designing Cryptographic Primitives 25
1.2.1.1 Block Ciphers *25*
1.2.1.2 Stream Ciphers *25*
1.2.1.3 Synchronous Stream Ciphers *25*
1.2.1.4 Public-Key Cryptosystems *26*
1.2.2 Crypto standardization 26
1.3 SECURITY AND CRYPTANALYSIS 27
1.4 SYSTEM DESIGN AND IMPLEMENTATION 27
1.4.1 Performance versus security 28
1.4.2 Performance Evaluation and Optimization 29
1.4.3 ASIC versus FPGA Design 29
1.5 LOW POWER HARDWARE DESIGN 29
1.5.1 Low-power design for SOC 30
1.6 HIGH PERFORMANCE VLSI ARCHITECTURES FOR CRYPTOGRAPHY 30
1.7 SCOPE OF THIS WORK 31
1.8 COMPLEXITY 31
1.9 STRUCTURE OF THIS THESIS 32
CHAPTER 2 DESIGNING FOR LOW-POWER/LOW-ENERGY **34**
2.1 MOTIVATION FOR LP DESIGN 34
2.2 SOURCE OF POWER IN CMOS CIRCUIT 34
2.3 CMOS LOW POWER DESIGN TECHNIQUES 36
2.3.1 Power Supply and Technology 36
2.3.1.1 Voltage scaling *36*
2.3.1.2 Technology mapping *36*

2.3.2 Reducing the activity factor 36
2.3.2.1 Glitch avoidance 37
2.3.2.2 Clock and Signal Gating 37
2.3.2.3 Power-down modes of global clock gating 38
2.3.2.4 Algorithm Optimization (reviewing) 38
2.3.3 Reducing the Capacitive Load 38
2.3.3.1 Partitioning into Blocks 38
2.3.3.2 Locality of reference 38
2.3.3.3 Clock and control signals distribution 39
2.3.3.4 Buffering the Heavily Loaded lines 39
2.3.4 Reducing the Short Circuit Current 39
2.3.4.1 Resistive networks 39
2.3.4.2 Switching current 39
2.3.4.3 Glitch propagation 40
2.4 POWER ESTIMATION AND OPTIMIZATION 40
2.4.1 Power Analysis and optimization Tools 40
2.4.2 LP Automated design Methodology 41
2.4.2.1 Preparing for RTL Design 41
2.4.2.2 Pre-synthesis design impact 41
2.5 LOW POWER DESIGN FLOW 42
2.5.1 Simulation Based Power Estimation and Optimization 42
2.5.1.1 Selecting the best Vector Set 42
2.5.1.2 ASIC Libraries 43
2.5.1.3 VSS-FAIF interface 43
2.5.2 Front-end or Logical Design 44
2.5.2.1 Behavioural/RT Power Estimation/Reduction 44
2.5.2.2 Gate level Power Calculation and Optimization 44
2.5.3 Back-end or Physical design 45
2.5.3.1 Timing and parasitics back-annotation 46
2.5.3.2 Priority in Back annotation 46
CHAPTER 3 STATE OF THE ART IN DESIGNING CRYPTOGRAPHIC PRIMITIVES 48
3.1 CRYPTOGRAPHIC PRIMITIVES 48
3.2 INFORMATION SECURITY OBJECTIVES 49
3.2.1 Data confidentiality or privacy 49
3.2.1.1 Encryption/Decryption process 49

3.2.1.2 Cryptosystem *49*
3.2.2 Data Integrity and Authentication 49
3.2.3 Digital Signature and Verification 50
3.2.4 Entity Authentication and Key Establishment Protocols 51
3.3 SECURITY CONSTRAINTS 52
3.3.1 Cryptanalysis (Attack Strategies) 52
3.3.1.1 Attacks Classification *52*
3.3.2 Asymmetric vs Symmetric primitive security 54
3.3.2.1 Brute-Force or Exhaustive Key Search *54*
3.3.2.2 Codebook Attack *54*
3.4 BUILDING CRYPTOGRAPHIC PRIMITIVES WITH THE CFSM MODEL 55
3.4.1 Block ciphers (Substitution permutation ciphers) 55
3.4.1.1 Block Cipher Properties *55*
3.4.1.1.1 Data Diffusion *55*
3.4.1.1.2 Avalanche effect: *56*
3.4.1.1.3 Completeness effect: *56*
3.4.1.2 Block ciphers construction *56*
3.4.1.3 Mode of operations *56*
3.4.2 Stream ciphers 57
3.4.2.1 Properties *58*
3.4.2.1.1 Stream Cipher Diffusion *58*
3.4.2.2 Stream Cipher Construction *58*
3.4.2.2.1 Confusion Sequence *58*
3.4.2.2.2 Combiner *59*
3.4.2.3 Security of Stream Cipher *59*
3.4.2.4 LFSRs based stream ciphers *60*
3.4.3 Public Key Ciphers 62
3.4.3.1 Certification *62*
3.4.3.2 Public Key Cryptosystems Properties *63*
3.4.3.2.1 Exponential versus Polynomial time Algorithms *63*
3.4.3.2.2 DLP versus IFP *63*
3.4.3.3 DLP in GF(p) and GF(2^m) *64*
3.4.3.4 DLP based Cryptosystems Construction *64*
3.4.4 Cryptographic Hash Functions (CHF) 65
3.4.4.1 Security Properties *66*

3.4.5 Keyed Hash Functions (KHF) 67
3.4.5.1 Requirements for Keyed Hash Functions 67
3.4.5.1.1 Security requirements 67
3.4.5.1.2 Design Heuristics 67
3.4.5.2 Construction of Keyed Hash Functions 68
3.4.5.2.1 The Secret Prefix Method 68
3.4.5.2.2 The Secret Suffix Method 68
3.4.5.2.3 The Envelope Method 68
3.4.5.2.4 New Construction 68
CHAPTER 4 FINITE FIELD ARITHMETIC FOR CRYPTOGRAPHY 70
4.1 FINITE FIELD ARCHITECTURES FOR CRYPTOGRAPHY 70
4.2 FINITE FIELD FUNDAMENTALS 71
4.2.1 Finite fields 71
4.2.2 Galois Fields 71
4.2.3 The extension field $GF(2^m)$ 72
4.2.4 The polynomial basis and primitive elements 72
4.2.5 The Dual Basis 73
4.2.6 Normal basis 74
4.2.7 Exponentiation Over $GF(2^m)$: S&M Algorithm 74
4.3 PRIMITIVE POLYNOMIALS AND FIELD SIZE FOR FINITE FIELD BASED CRYPTOSYSTEMS 75
4.4 ARCHITECTURES FOR MULTIPLICATION OVER LARGE $GF(2^m)$ 75
4.4.1 Bit-serial PB Multipliers 76
4.4.2 Architecture Level Transformation 78
4.4.2.1 Digit-Serial MSR Multiplier 78
4.4.2.2 Linear Digit-Serial Systolic Array Multiplier 80
4.4.3 Implementation Issues and Comparison 84
4.5 EXPONENTIATION OVER LARGE $GF(2^m)$ 87
4.5.1 Low Power Architectures for Large $GF(2^m)$ Exponentiation 87
4.5.1.1 Reducing number of operations 87
4.5.1.2 Speeding up the computation 88
4.5.1.3 Bases choice 88
4.5.2 LSA Based Architecture for GF(2M) Exponentiation 89
4.5.2.1 Linear Systolic Array Exponentiator 89
4.5.2.2 Digit-Serial Linear Systolic Array Exponentiator 91

4.5.3 Implementation results and Comparison 92
4.6 PHYSICAL DESIGN: LSA EXPONENTIATOR (PLACEMENT AND ROUTING) 96
CHAPTER 5 SELF-SYNCHRONIZING STREAM CIPHERS (SSSC) 98
5.1 SSSC DESIGN PARAMETERS 98
5.1.1 Cipher Feedback Approach 99
5.1.2 Fault Tolerance 100
5.2 FRAME BASED SSSC (SSMG PRINCIPLE) 100
5.2.1 Mixture Generator MG 101
5.2.1.1 Bit and Byte Granularity 101
5.2.1.2 MG Stuttering 102
5.2.1.3 MG Initialization 103
5.2.2 Mangler 103
5.2.3 Keyed Hash Function (KHF) 103
5.2.3.1 Implementation of the Hash Function 104
5.2.3.2 Diffusion and Confusion 105
5.2.3.3 Collision free 105
5.3 SSMG: OPERATIONS OF THE CIPHER 106
5.4 SSMG: CRYPTOGRAPHIC SECURITY 106
5.4.1 Repeated Occurrences 107
5.4.2 Period of the SSCS and Hash Result Size 107
CHAPTER 6 SECURING MPEG-2 STREAM IN DVB 108
6.1 MPEG-2 STREAM SOFTWARE ENCRYPTION 108
6.1.1 Cryptographic Primitives 109
6.1.2 Structure of the MPEG-2 Codec 109
6.1.3 MPEG-2 Stream Encryption 110
6.1.3.1 MPEG-2 selective encryption scheme 110
6.1.3.2 Quality of selective encryption 111
6.1.4 Encryption/Decryption Process 111
6.1.4.1 Constraints 111
6.1.4.2 Design choices 111
6.1.4.3 Interface functions 112
6.1.4.4 SSMG Software Implementation 113
6.1.5 Implementation and Analysis 113
6.1.5.1 Performance profile of the MPEG encoder 114
6.1.5.2 Performance of the ciphers during encryption 114

6.1.5.3 Performance profile of the SSMG ... *117*
6.1.5.4 Decryption and Real-time Playback ... *117*
6.1.6 Main results ... 118
6.1.6.1 Fastest ciphers ... *118*
6.1.6.2 Measuring one sequence ... *118*
6.1.6.3 Encryption overhead ... *118*
6.1.6.4 Interfacing the ciphers ... *118*
6.1.6.5 Weaknesses of the encryption scheme ... *118*
6.2 PART2: MPEG-2 STREAM HARDWARE ENCRYPTION ... 119
6.2.1 Program Stream and Transport Stream ... 119
6.2.2 MPEG-2 Transmission using DVB standards ... 119
6.2.2.1 Transmission of the Mpeg-TS ... *119*
6.2.2.2 DVB Bearer Networks ... *120*
6.2.2.3 FEC and Channel Coding ... *120*
6.2.3 DVB Encryption System ... 121
6.2.3.1 Baseband MPEG-2 Interfaces ... *122*
6.2.3.1.1 Asynchronous Serial Interface (ASI) ... *123*
6.2.3.1.2 Synchronous Serial Interface (SSI) ... *123*
6.2.3.1.3 Synchronous Parallel Interface (SPI) ... *124*
6.2.3.2 Cryptosystem Structure ... *124*
6.2.3.2.1 PK-SSMG Interface ... *124*
6.2.3.2.2 MPEG-TS Parsing (frame decoding) ... *126*
6.2.3.2.3 Mixture Generator ... *126*
6.2.3.2.4 Exponentiator ... *126*
6.2.3.2.5 Input/Output Fifos ... *127*
6.2.3.2.6 Bus Multiplexing ... *128*
6.2.3.2.7 Mangler ... *129*
6.2.3.3 PK-SSMG Controller ... *129*
6.2.3.3.1 The Burst Mode ... *130*
6.2.3.3.2 Exponentiation Phase ... *130*
6.2.3.3.3 MG Initialization and Combining Phases ... *131*
6.2.4 Performance Measurement and Results ... 131
6.2.4.1 Physical Design ... *133*
CHAPTER 7 EFFICIENT IMPLEMENTATION OF RIJNDAEL BLOCK CIPHER ... 134
7.1 CHOICE OF AN ARCHITECTURE ... 134

7.1.1 Mode of operation 134
7.1.2 Hardware Parameters 134
7.1.3 Cipher Architectures for Feedback modes 135
7.1.3.1 Basic Iterative Looping 135
7.1.3.2 Partial and Full Loop Unrolling 135
7.1.4 Cipher Architectures for Non-Feedback Mode 136
7.1.4.1 Partial/Full Pipelining 136
7.1.4.2 Partial/Full Pipelining with Sub-Pipelining 137
7.2 STRUCTURE AND PARAMETERS OF RIJNDAEL BLOCK CIPHER 137
7.3 RIJNDAEL ITERATIVE LOOP UNROLLING 139
7.3.1 Interface 139
7.3.2 Key schedule 140
7.3.3 Data Flow 141
7.3.3.1 Shift Row Function 142
7.3.3.2 MIX Column Function 142
7.4 RIJNDAEL SBOXES 143
7.4.1 Algorithm of Morii-Kasahara 144
7.4.2 $GF(2^8)$ as an Extension of $GF(2^4)$ 144
7.4.3 Transition Between Representations of $GF(2^8)$ 145
7.4.4 An inverter over $GF(2^8)$ 146
7.4.5 Rijndael SBOX Configuration 147
7.5 IMPLEMENTATION RESULTS 147
7.5.1 ASIC Implementation 147
7.5.2 FPGA IMPLEMENTATION 150
CHAPTER 8 SUMMARY AND CONCLUSIONS 152
8.1 MAIN CONTRIBUTIONS 152
8.1.1 Arithmetic in Large $GF(2^m)$ 152
8.1.2 Self-Synchronizing Mixture Generator 153
8.1.3 Securing MPEG-2 Bitstream 153
8.1.4 Efficient Implementation of Rijndael Block Cipher 154
8.2 FURTHER WORK 154
REFERENCES 156
APPENDIX A CRYPTOGRAPHY (BASICS AND LFSRS THEORY .....) 162
A.1 DEFINITIONS 162
A.1.1 Maximal Length 162
A.1.2 Linear Complexity 162

A.2 LFSR THEORY 162
A.2.1 Golomb's Principles 163
A.2.2 Some Facts and Definitions From Algebra 164
A.3 DISCRETE LOGARITHM PROBLEM (DLP) 166
A.3.1 Diffie-Hellman Key Exchange 167
A.3.2 El-Gamal Cryptosystem 167
A.3.3 Shank's Algorithm for Solving the Discrete Logarithm Problem 167
A.3.4 Pohlig-Hellman Algorithm 168
A.3.5 Chinese Remainder Theorem 168
APPENDIX B BIT PARALLEL ARITHMETICS IN $GF(2^4)$ 170
B.1 REPRESENTATIONS OF $GF(2^4)$ 170
B.2 STANDARD MULTIPLICATION IN $GF(2^4)$ 170
B.3 STANDARD MULTIPLICATION BY CONSTANT W^{14} IN $GF(2^4)$ 171
B.4 STANDARD SQUARING IN $GF(2^4)$ 171
B.5 DIRECT INVERSION IN $GF(2^4)$ 171
APPENDIX C PERFORMANCE EVALUATION OF $GF(2^{607})$ MULTIPLICATION AND EXPONENTIATION 172
C.1 $GF(2^{607})$ MULTIPLICATION 172
C.2 $GF(2^{607})$ EXPONENTIATION 175
APPENDIX D SELECTIVE ENCRYPTION BITS FOR A SEQUENCE 178

LIST OF FIGURES

Low-power oriented design flow using sub-micron processes. 43
Detailed LP deep submicron design flow 45
Cryptographic primitives 48
Digital signature for data integrity validation 50
Diffie-Hellman Key establishment protocols (key authentication and key agreement) . . 51
Cryptographic finite state machine: implementing of a Block Cipher 55
Block Cipher operating mode (encryption). 57
Streaming using Block Cipher in CFB mode. 57
Cryptographic finite state machine: implementation of a PRG 58
Examples of LFSR based Stream Cipher 61
Public and Private Operation Scheme of DLP based Cryptosystem. 65
Cryptographic finite state machine: implementation of a CHF 66
MSR m-bit multiplier architecture 76
Linear systolic array multiplier 77
LSA basic processing cell and its algorithm 77
Digit-serial MSR multiplier for field-generating polynomial $p(x)=1+x^k+x^m$ 79
Digit-serial, linear systolic array multiplier 81
LSA digit-serial basic processing cell and its algorithm 81
LSA CELL-K multiplier representation 81
Circuit diagram of the F function 82
Folding bit-serial computations 83
Circuit diagram of the G function for one bit output 84
Total delay comparison as function of the digit-size for one $GF(2^{607})$ multiplication . . . 85
Area in gates of the MSR, LSA and clock gating LSA digit-serial $GF(2^{607})$ multipliers as function of the digit size 85
Energy-Delay product comparison between MSR and LSA digit-serial $GF(2^{607})$ multipliers 86
Energy-Delay-AreaproductcomparisonbetweenMSRandLSAdigit-serialGF(2^{607})multipliers 87
$GF(2^m)$ LSA based exponentiator for odd field size m 89
Overlapping the squaring and multiplication operations 90
The exponentiator operation modes 90
Digit-Serial LSA multiplier based exponentiator 92
Total Delay as function of the digit size for one LSA digit-serial $GF(2^{607})$ exponentiation 93
Area in gates of the LSA digit-serial $GF(2^{607})$ exponentiator as function of the digit size...93
Different types of power dissipation components as function of the digit size of the LSA digit-serial $GF(2^{607})$ exponentiator 94
Power profile for LSA digit-serial (D=4) exponentiator (1.8v) 94

Energy-Delay product as function of digit-size of the LSA digit-serial $GF(2^{607})$ exponentiator 95
Energy-Delay-Area product as function of the digit size of the LSA digit-serial $GF(2^{607})$ exponentiator 95
Digit-serial (D=4) LSA Exponentiator Layout (0.18m,6ML) 96
Cryptographic state machine: implementation of an SSSC 99
SSMG stream cipher encryption 100
SSMG stream cipher decryption 101
Byte Granularity: trinomial LFSR 102
Bit Granularity: Speeding up the Bottom LFSR 102
Mixture Generator Stuttering scheme 103
KHF structure 104
Cellhash structure 105
SSMG operation: inter-frame dependencies during encryption and decryption 106
Structure of the MPEG-2 codec 110
MPEG encryption/decryption structure. 112
Interface for ciphers. 113
Performance of different ciphers during encryption vs MPEG encoding. 115
SSMG encryption overhead reported to DES 115
Bits Rates for the Different Cipher 116
SSMG encryption bit-rate reported to DES 116
Impact of Cellhash on SSMG performance 117
MPEG-2 from Presentation unit to Transport stream 119
DVB Encryption System using SPI 122
MPEG-2 TS packet formats for SPI 123
PK-SSMG datapath 125
TS packet header 126
Bus multiplexing during combining phase 128
Bus multiplexing: data bypass and encryption 129
PK-SSMG finite state machine (controller) 129
Burst mode 130
Disabling data request from the input FIFO 131
Area and power profiles of the cryptosystem 132
Energy consumption per bit of different implementations 132
PK-SSMG Layout 133
Alternative architectures of the encryption/decryption unit of a block cipher suitable for FB mode: a) basic iterative looping, b) partial loop unrolling, c) full loop unrolling 136
Alternative architectures of the encryption/decryption unit of a block cipher suitable for NFB mode: a) basic iterative looping, b) partial outer-round pipelining, c) full outer-round pipelining, d) inner-round pipelining, e) partial mixed inner- and outer-round pipelining, f) full mixed inner- and outer-round pipelining. 137
Rijndael round function structure: a) general structure, b) optimized structure. 138
Throughput & area vs round stages. 139

External interface. 139
On-chip subkeys calculation. 140
Key-schedule Block bit-level Structural Architecture 140
Output stage of the Key-schedule for final operation 141
Bit-level Structural architecture of the round function 141
SHIFT1 and SHIFT3 components 142
MIX and Elementary MIX component structure 142
Rijndael SBOX 143
Memory requirement for Rijndael SBOXes vs round stages 144
SBOX: a) structure of the elementary SBOX function, b) Rijndael SBOX component 147
Gate count of both SBOX-LUT and SBOX-IN architectures: a) circuit operating at 0.18v, b) circuit operating at 0.9v 148
ThroughputofbothSBOX_LUTandSBOX-OFarchitecturesiniterativeloopingconfiguration 148
Dynamic Power Consumption 149
Energy-Delay Product during encryption 150

LIST OF TABLES

Characteristic features of implementations of cryptographic transformations in ASICs, FPGAs, and Software 28
Attack strategies on cryptographic primitives 53
Description of LFSR based Stream Cipher 61
PRPG's Period and Linear Complexity 61
Most useful irreducible trinomials x^m+x^k+1, for each large Mersenne prime m, $512 \le m \le 4423$ 75
X_1 and X_2 values for digit size D = 8,16,32 80
Key parameters for different architectures of serial $GF(2^m)$ multipliers using different bases 88
Area overhead: digit-serial (D=4) LSA Exponentiator core implementation 96
Configuration Values of the Encryption Interface 109
Selective Encryption Schemes 111
Encrypted Bits Per Mode 114
Time Consumption Profile of the MPEG-2 Codec 114
Performance profile of the SSMG with key length 607 117
User interface control bits: data input destination 125
User interface control bits: data output 125
MG configuration parameters 126
SSMG Latency: byte granularity 127
SSMG Latency: byte granularity 127
Input FIFO: data flags 128
Bus Sharing: source and destination 128
SSMG Latency with bus sharing: bit/byte granularity 128
Area and power consumption of each instance in the PK-SSMG cryptosystem 131
PK-SSMG Core VLSI implementation (0.18um,6ML) 133
number of round r as a function of key and block length. 138
FPGA Performance evaluation of Rijndael LU-1 150

ACRONYMS

AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
CFSM	Cryptographic FSM
CHF	Cryptographic Hash Function
CMOS	Complementary MOS
DES	Data Encryption Standard
DL-P	Discrete Logarithm-Problem
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GF	Galois Field
HDL	Hardware Description Language
HDTV	High Definition Television
IC	Integrated Circuit
IDEA	International Data Encryption Algorithm
KHF	Keyed Hash Function
LFSR	Linear Feedback Shift Register
LP	Low Power
LUT	Look Up Table
MG	Mixture Generator
PDEF	Physical Design Exchange Format
PRBG	Pseudo-Random Bit Generator
QEF	Quasi Error Free
RNG	Random Number Generator
RTL	Register Transfer Level
SA	Switching Activity
SAIF	Switching Activity Interchange Format
SDF	Standard Delay Format
SL	Synchronization Loss
SSSC	Self-Synchronous Stream Cipher

SSMG	Self-Synchronous Mixture Generator
VHDL	VHSIC Description Language
VSS	VHDL System Simulator

ABSTRACT

The main topic of this thesis is related to the state of the art in designing cryptographic primitives from a hardware point of view. A special emphasis is dedicated to low-power/low-energy CMOS design. A set of solutions is proposed including an LFSR based stream cipher with self-synchronizing capabilities, a new memory-less Rijndael block cipher architecture and a public key scheme in the class of discrete logarithm. The former is based on arithmetic in large finite field, mainly Galois extension field $GF(2^m)$. These solutions are droved using low-energy techniques, in order to decrease both the switching activity and the total delay. The fundamental motivation supporting this work, is to demonstrate that practical solutions can be obtained for implementing such complex primitives in large scaled circuits, that are at once, high performance architectures (low-power, high-speed) and cryptographicaly strong, using the well known trade-off between area-speed or area-power. Security constraint has been duly considered, mainly by increasing the key-size.

In this work, we explore the general aspects of designing the above mentioned cryptographic functions. We give an extensive survey of some cryptographic primitives from the hardware point of view and expose their security properties. The thesis favours stream cipher and public-key schemes, as currently the most promising advance to capture the notion of key generation and establishment and data bulk encryption. One contribution is the convenient notation for expressing cryptographic self-synchronizing stream ciphers SSSC schemes and our SSMG proposal, a scheme based on packet fingerprint identification, that relies on keyed cryptographic hash function to achieve the security requirements. We maintain an important distinction between hardware implementation and algorithm's security, because the security of cryptographic primitives cannot be based on mathematically strong functions only but requires an extensive cryptanalysis at different levels including the application. This causes a concern for a formalization of the security of an implemented cryptographic primitive. Nevertheless, while some schemes are well known to be secure such as DL based public key schemes and enough cryptanalyzed such as the new standard Rijndael, some security aspects of the SSMG are discussed.

A part of this work studies the specific aspects related to hardware implementation of Rijndael block cipher, the new standard designed to be a substitute for DES. An efficient architecture is developed targeting FPGA implementation, by simply avoiding memory blocks dedicated to the implementation of S-boxes and replacing them by on-chip forward computation using composite Galois field. This technique helps to reduce considerably the amount of hardware required at the cost of little increase of the switching activity.

The main conclusion is that, while security constraint of cryptographic primitives increases the hardware complexity and reduces the performances, practical solutions exist for reducing such complexities while keeping or increasing the level of security. Nevertheless, major open questions remain both for a firm theoretical foundation and the proper cryptanalysis of certain solutions.

RÉSUMÉ

Ce travail est dédié à l'état de l'art dans la conception de circuits intégrés implémentant des primitives de cryptographie. L'accent est mis la conception d'architectures CMOS à basse consommation et basse énergie. Un ensemble de solutions est proposé incluant les 2 procédée de cryptage: à voir le cryptage en continu avec des stream ciphers, ou générateurs de suites de bits à base de registres et le cryptage en block basé sur implémentation sans mémoire de l'algorithm Rijndael. D'autres architectures sont proposées, qui implémentent des arithmétiques utilisées par une classe d'algorithmes à clé publique (asymétriques) basés sur la difficulté de résoudre le logarithme numérique dans un ensemble fini, principalement dans le champ polynomial de Galois $GF(2^m)$. Ces solutions sont établies en utilisant des techniques qui réduisent à la fois l'activité des circuits et le délai total de calcul. La motivation fondamentale est de démontrer que des solutions pratiques existent, qui sont à la fois performantes et cryptographiquement sûres, tout en se basant sur le compromis surface-vitesse de traitement ou surface-consommation. La contrainte de sécurité est directement liée à la clé qui permet de changer le niveau de sécurité.

Dans ce travail nous explorons les aspects généraux liés à la conception des primitives susmentionnées. Nous exposons quelques fonctions cryptographiques ainsi que leurs propriétés de sécurité de point de vue de leur implémentation matérielle. On favorise les primitives à clé publique pour la génération et l'établissement de la clé secret ainsi que celle du cryptage en continue qui accélère nettement les débits et autorise le cryptage en masse des données. Une autre contribution consiste en la formulation de l'expression de stream cipher auto-resynchronizé SSAS et la proposition de notre solution SSMG. Ce dernier est un stream cipher basé sur le principe d'identification des paquets avec une fonction de hachage cryptographique à clé afin d'augmenter le degré de sécurité. Dans ce travail on maintient une distinction entre l'implémentation matérielle et la sécurité des algorithmes car la sécurité des primitives de cryptographie ne peut pas être basée sur des fonctions mathématiques sûres seulement, mais une cryptanalyze approfondie doit être fournie. Ceci pose le problème de formuler théoriquement la sécurité de chaque primitive implémentée. Néanmoins, alors que quelques primitives exposées ici ont des propriétés bien connues, comme celles des algorithmes à clé publique basés sur l'exponentiation dans un champ fini, et celle de Rijndael block cipher qui est bien cryptanalyzé, quelques aspects de sécurité du SSMG sont discutés.

Une partie de ce travail est dédiée à l'implémentation de l'algorithme Rijndael, le nouveau standard désigné parla communauté internationale comme remplaçant pour DES. Une architecture optimisée en surface a été développée pour une implémentation FPGA. Dans cette architecture les S-boxes ont été remplacés par des calculs à la volée en utilisant de l'arithmétique dans le champ composite de galois. Cette technique permet de réduire considérablement la surface du circuit mais augmente sensiblement son activité.

Bien que la contrainte de sécurité de ces primitives augmente la complexité matérielle de leurs circuits correspondant et réduits les performances, des solutions pratiques existent pour réduire cette complexité et maintenir le même niveau de sécurité. Néanmoins, la base théorique et la cryptanalyze de certaines solutions restent à developpées.

CHAPTER 1

1 INTRODUCTION

1.1 MOTIVATION

The advent of the Information age and the emergence of the Internet and world wide networking technology made Cryptography as an essential part of today's information systems for both the public and commercial sectors, in order to protect sensitive, but unclassified information. Cryptography a set of mathematical tools helps provide fairness, privacy, accuracy and confidentiality services. These tools described by algorithms must be well designed and implemented as software or hardware layers for a specific application.

For performance as well as for physical security reasons, it is often less convenient but highly desirable to realize cryptographic algorithms in hardware. The availability of fast/low-power hardware cryptographic cores tends to follow wide range of application starting from applications with potential data rates in the gigabit range, such as offered by high bandwidth networks till portable applications with embedded processor blocks performing computations with power efficiency.

Meanwhile, the increasing security demands by increasing the keys size, put many constraints on the design of a secure, but suitable high throughput cryptosystem implementing public-key and data confidentiality schemes. These applications require hardware that operates at link speed, implementing one-way functions such as $GF(2^m)$ exponentiator, large stream or block cipher. Also, for applications requiring bit-synchronization some mechanisms such as stream cipher resynchronization if doesn't increase the band-width it adds latency.

These cryptographic circuits have to be regular and scalable in order to makes them simple to extend to even larger widths in future implementations requiring higher degree of security. In the other hand, arithmetic operators in long public-key cryptosystem and or stream/block cipher exhibit in general a great activity and dissipate consequent shares of the power supply. Unlike most research involving large number of cryptographic primitives, this work targets a low-power/low-energy crypto-components through the application of various well defined power reduction techniques to different types of architecture and comparing their power consumption among other factors, rather than comparing complexity measures such as gate count or area. Gate count is used as a starting point to choose potential architectures.

Currently, the design of low-power high-performance architectures is the major challenge in most areas of VLSI design. Quantifying and exploring the trade-off between power consumption and speed should stimulate the research and development of cryptographic primitives according to the security requirements, which was forced too long in the past to proceed without actual fast and low-power hardware.

1.2 CURRENT AND FUTURE TRENDS FOR CRYPTOGRAPHY

1.2.1 DESIGNING CRYPTOGRAPHIC PRIMITIVES

Mainly, three are four common cryptographic primitives: block ciphers, stream cipher, on-way hash functions and public-key schemes.

1.2.1.1 BLOCK CIPHERS

The popularity of block ciphers is closely related to the popularity of DES [1]. It has influenced conventional cryptography in a major way. Today DES has exceeded his life time; it is no longer secure in presence of huge computational methods and resources available today. Researchers actually redirect their activity toward the design of more convenient block cipher either for software as well as hardware implementation with a high degree of security.

A large number of recent block cipher proposals are based on special operations to thwart some cryptanalysis attacks. These proposals include the same components for global structure, the number of rounds, non-linearity, diffusion and key schedule. They are designed based on non-linear substitutions such as S-Blocks, transposition such as bit permutation to ensure the diffusion. These techniques are alternated with mixing functions in order to diffuse and confuse the redundancy by the mixing transformation. Feistel structure of DES is an example of transformation that ensures both substitution and transposition. Then, the strength of the block cipher can be obtained by repeating this simple transformation.

1.2.1.2 STREAM CIPHERS

Anyone looking through the cryptographic literature will be struck by a great difference in the treatment of block ciphers and stream ciphers. Practically all work in the cryptanalysis of block ciphers is focused on DES and nearly all the proposed block ciphers are based in some way on the perceived design goals of DES. There is no algorithm occupying an equivalent position in the field of stream ciphers. There are a huge variety of alternative stream cipher designs and cryptanalysis tends to be couched in very general terms.

Many stream ciphers have been proposed in the vast literature, which use very basic building blocks. The mathematical analysis of these components has been very advanced for some considerable time, and intensive design and cryptanalysis over the years has resulted in the formulation of a set of ground rules for the design of stream ciphers. It is well known that highly developed analytic techniques facilitate both design and cryptanalysis. Today, more developers look to stream ciphers to provide the encryption speed they need. In this work a particular attention is considered for a class of synchronous stream cipher of *clock control* LFSR type. This scheme is particularly well suited strong for hardware implementation. Moreover, they are fast enough to include more some mechanism such as *self-synchronization.*

1.2.1.3 SYNCHRONOUS STREAM CIPHERS

Binary self-synchronous stream ciphers (SSSCs) while are useful for widespread applications, they have received little attention in the open cryptographic literature. While some design and security criteria are known for synchronous stream ciphers and block ciphers, only little is known about the design of SSSCs. These are synchronous stream ciphers with self-synchronization capabilities using a synchronization parameters, i.g., using ciphertext to generate the *keystream* sequences. In this work, we present a dedicated SSSC design based on clock controlled LFSR stream cipher and fingerprint of subsequent ciphertext blocks as synchronization parameters. This scheme

presents the feature of high degree of security and conserves the size of the original ciphertext/ plaintext. Thus, it doesn't increase the bandwidth of the communication channel between the transmitter and receiver at the cost of increasing latency but suit a range of application such as securing a DVB MPEG-2 stream.

1.2.1.4 PUBLIC-KEY CRYPTOSYSTEMS

The public key label covers a large number of cryptographic algorithms having different capabilities. It is interesting to note that most of these algorithms are incapable of data encryption. However, their principal applications are for digital signatures, secure exchange of secret keys, and authentication. In fact, the existing public key algorithms, while satisfactory for securing key generation (key distribution decrease the possibility of secret keys falling into the wrong hands), are not fast enough and not suitable, even when implemented in hardware, to encrypt high volumes of data traffic and hardly perform at speeds suitable for today's communications systems. Rather, they are well suited for use in digital signature generation and key establishment. The generated message key can be used by a conventional secret key cipher, which actually enciphers the data. Thus, hybrid systems could be constructed giving rise to public-key based either stream or block encryption. The security of the underlaying algorithms of these schemes is based on the difficulty of solving a mathematical problem. The discrete logarithm based scheme is one of these problem (DLP) that is at least as hard as the well defined Integer factorization problem (IFP) on which RSA [50] is based (named for its inventors Rivest, Shamir and Adleman). We'll discuss it here as the representative of the entire class mainly the arithmetic in large Galois field GF(2^m).

1.2.2 CRYPTO STANDARDIZATION

Over the last 5 years, a huge effort has been undertaken to provide a Standard Specifications for Public Key Cryptography under a process called IEEE P1363 project, bringing together many of the important developments of the past 20 years into a single reference document for the first time. The project is scheduled to be published later this year (2001) and has as its goal the issuance of Standard Specifications For Public-Key Cryptography. This is accomplished via a series of standards documents. The IEEE P1363 working group is writing the actual standards documents and determines what kinds of algorithms should be included in them. This process is open to the public, and occasional editorial teleconferences.

The IEEE 1363 standard defines a full range of common public-key techniques such as key agreement, public-key encryption and digital signatures for the principal cryptographic families. The standard supports a wide range of application and security requirements and offers detailed descriptions of the main algorithms employed in today's most popular public-key cryptographic technologies, including the RSA public-key cryptosystem, Diffie-Hellman key agreement and elliptic curve cryptography. It is designed to accommodate improvements in technology over time. The standard was given final approval by the IEEE-SA Standards Board at its meeting in Singapore in January 2002.

In the other hand, DES which was developed in 1977 with an anticipated lifetime of 5 years, has been the most commonly used encryption algorithm in the industry for the last twenty years and became the de-facto standard world-wide. With the advent of technology and improved computational resources to handle the huge amounts of data involved in some attacks, the 56-bits DES became obsolete in today high security-demanding applications. It has exceeded its lifetime by a large margin and researchers demonstrated that it can be broken using brute force attack (see Section 3.3.2.1) in 2.5 days, on average, with a $10'000 initial investment into custom hardware [22]. Triple-DES [23] emerged as a temporary solution in applications such as banking, but its performance

mainly speed was not sufficient for other tasks like streaming multimedia contents. The ongoing request of security and performance brings the National Institute of Standards and Technology (NIST) to define a set of new encryption standards. In 1997, NIST announced an international in progress standard effort to define a new bulk data encryption standard for masses with extremely high security and high performance. This process is called the Advanced Encryption Standard (AES) conducted by a list of specifications, which must be met by an AES candidate. Unlike DES, which was designed specifically for hardware implementation, on of the design criteria for AES candidate algorithms is that they can be efficiently implemented in both hardware and software. In 2000, NIST released a draft standard called RIJNDAEL algorithm, that have been have chosen as a replacement for DES. It likely to be the international cipher of choice for the next 30 years.

1.3 SECURITY AND CRYPTANALYSIS

Cryptographic design may seem as easy as selecting a cipher from a book of ciphers. But ciphers, per se, are only part of a secure encryption system. It is common for a cipher system to require cryptographic design beyond simply selecting a cipher, and such design is much trickier than it looks.

In general, beside cipher cryptanalysis which concentrate on braking the block using mathematical background the strength of keyed cryptographic schemes is related to the difficulty of discovering the key, which in turn depends on the length of the key. This is a fact when cryptanalysis fail to attack the cryptographic primitive. In such case the only one measure of security is the amount of effort undertaken under brute-force attack to break the system. Encryption strength is often described in terms of the size of the keys used to perform the encryption, where longer keys provide stronger encryption. Nevertheless, when considering cryptanalysis, different ciphers may require different key lengths to achieve the same level of encryption strength. This is the case when comparing the IFP versus DLP based cryptosystem (see Section 3.4.3.2).

1.4 SYSTEM DESIGN AND IMPLEMENTATION

The security of most modern public-key ciphers is related to some long-studied mathematical problem that is difficult to solve (i.g., the integer factoring problem IFP or finding discrete logarithms DLP). Technology advances quickly and day by day it becomes more likely that this security rely on key sizes beyond the proven security based on the hard or difficult mathematical problem [2]. Also the best (practice) known attack against secure symmetric key systems is a brute-force search in the key space, which is purely exponential. Nothing else seems to work. Why not simply oversize the encryption algorithm to solve this inconvenient but inevitable security problem? The main reason is speed beyond the problem of key management and distribution for symmetric key systems. It is usually claimed that long keys slow down the algorithm too much. That's particularly true in software implementation because execution time increases at least by the key size at power of two in general purpose processor [3]. A hardware implementation can be of consideration in order to overcome this limitation. However, the effect is an increase in the cost function defined by power and area constraints. Furthermore, practical cryptography is rarely broken through the mathematic; other parts of systems are much easier to break (software, computer and network security) especially, the people-key management, human/computer interface security, access control.

	Acids (semi-custom and full-custom)	FPGA	Software (general purpose processor)
Performance Characteristics			
Parallel processing of data	yes	yes	limited
Pipelining	yes	yes	limited
Word size	variable	variable	fixed
Speed	very fast	fast	moderately fast
Power optimization	variable	Limited	Limited
Functionality and Security			
Algorithm agility	no	yes	yes
Temper resistance	strong	limited	weak
Access control to keys	strong	moderate	weak
Development process			
Description language	VHDL, Verilog HDL	VHDL, Verilog HDL	C, C++, Java, Assembly language
Design cycle	long	moderately long	short
Design tools	very expensive	moderately expensive	inexpensive
Testing	expensive	moderately expensive	inexpensive
Maintenance and upgrades	expensive	inexpensive	inexpensive

Table 1-1: Characteristic features of implementations of cryptographic transformations in ASICs, FPGAs, and Software

An increases number of applications uses software implementation of cryptographic algorithms in order to provide an acceptable security at low cost. Currently, most ciphers are implemented in software; that is, by a program of instructions executed by a general-purpose computer. Normally, software is cheaper, but hardware can run faster. Of course, there are levels to hardware, from chips (which thus require significant interface software) to external boxes with communications lines running in and out. But there are several possible problems:

- Software, especially in a multi-user system, is almost completely insecure. Anyone with access to the machine could insert modified software which would then be repeatedly used under the false assumption that effective security was still in place. This may not be an issue for home users, and real solution here may depend upon a secure operating system.
- Hardware represents a capital expense, and is extremely inflexible. So if problems begin to be suspected in a hardware cipher, the expense of replacement argues against an update. Indeed, a society-wide system might well take years to update anyway.

One logical possibility is the development of ciphering processors (little ciphering computers) in secure packaging. Limited control over the processor might allow a public-key authenticated software update, while otherwise looking like hardware. But probably most users will not care until some hidden software system is exposed on some computers.

1.4.1 PERFORMANCE VERSUS SECURITY

Achieving high performance at the cost of security?!. It is clear that someone with a good knowledge of present day cryptanalysis can design a secure but slow algorithm with a very limited effort. At present, there exist a trade-off between the security and the performance requirements of cryptographic primitives. In fact, most of these ciphers are secure after sufficiently many rounds but are too slow. The performance of these primitives depends on the implementation details, even

for a given environment, i.e. processor and memory in software, and power and area constraints in hardware.

1.4.2 PERFORMANCE EVALUATION AND OPTIMIZATION

Optimizing the performance of software and hardware implementation of the cryptographic primitives is quite different as shown in Table 1-1. Fast hardware relies on parallelism and pipelining, while for software the access to memory is a key to high performance. Other aspect which influence software performance are word size, byte ordering *endianness*, which is processor dependent. Running an algorithm on a processor that has a smaller word size than that of the algorithm will normally result in reduced performance (it requires more instructions to perform the same operations). Software optimization aims to minimize access to slow memory, and performing calculations on chip as much as possible using registers and cache memory. Also, the conducting recent evolutions in general purpose processors towards more inherent parallelism, contributed to increase the performance both on the hardware level using superscalar architectures (multiple execution units) and software level (SIMD instructions).

Hardware implementation helps to achieve the required speed, since working on low-level on dedicated architecture. Nevertheless, an effort must be handled to achieve the requested performances mainly power consumption for ASIC implementation.

1.4.3 ASIC VERSUS FPGA DESIGN

FPGA costs and time-to-market are looking awfully attractive. The design cycle for ASICs is getting longer due to the deep sub-micron effects. On the other hand, FPGAs have grown to become multi-million gate devices with system level features that can implement complete systems on a chip. Today's largest FPGAs approaches the few-million-gate size of a typical ASIC design, and continue to sprout embedded cores, such as CPUs, memories, and interfaces. But still ASIC technology provide a continuum of possibilities among these are power performance and high density about 20 times improvement over standard FPGAs.

In spite of programmability and the short design cycle features within FPGAs, there are performance issues including power and speed and density issues. Obviously, ASICs will still be faster with smaller area than FPGAs and consuming low power when using the same technologies. Quantifying and exploring area and power gap between ASIC and FPGA is all wrong. Performance issues can be addressed with new logic styles while density issues can be addressed by introducing a small degree of mask programmability [4]. In the same way (concurrently), as process geometries shrink and new design styles and methodologies are developed, ASIC is pushed toward opportunities with more and more logic, speed and less power consumption. It is still a fact so far as existing ASIC design tools and design methodologies are not overpowered for future sub-100 *nm* fabrication technologies.

1.5 LOW POWER HARDWARE DESIGN

Power dissipation has emerged as an important design parameter in VLSI circuit design. The huge impact of power on ASIC design is new and prompting significant changes in design methodology. Power is moving up behind timing in significance, and pushing area to third place.

The general increase in power dissipation in complex ICs presents several classes of problems. One is simply the power budget of the overall system. Many kinds of equipment have power ceil-

ings constrained by relatively fixed factors such as battery life for portable applications. Another class of problems is associated with temperature and its impact on reliability.

Traditional reliability issues generally place a ceiling on acceptable junction temperatures, and these ceilings don't scale with Moore's Law. As a result, higher-power chips generally demand more elaborate packaging and cooling infrastructures, which can add rapidly to the manufacturing cost, size and weight of the end equipment. Newer reliability concerns such as electromigration affect metal pitch and routability. Finally, chip temperatures have a small but significant impact on electrical performance. In a high-performance chip, internal power dissipation may determine the difference between meeting or missing timing constraints. Thus, reducing power consumption is equally important for non-portable systems.

1.5.1 LOW-POWER DESIGN FOR SOC

A simple view of SOCs is that they integrate functionality that previously would have taken several different ICs. Among other factors at work are increases in clock frequencies, higher device densities at 0.18 μm and smaller geometries, increases in architectural sophistication and features, and the migration of functionality from power-thrifty hardware (especially analog) into power-hungry software. As a result of these influences, chip designs now routinely dissipate several watts or more on a single chip and the figures are increasing.

1.6 HIGH PERFORMANCE VLSI ARCHITECTURES FOR CRYPTOGRAPHY

Latency is the major consequence of increasing security level in cryptographic primitives. Low latency can be achieved by buffering data as little as possible and high performance (high speed, low power) design may be achieved by using custom design in order to save silicon area (compact design) and increase the speed using dynamic logic on critical paths [5]. Custom design allows using of different dedicated design styles to improve the power consumption [6]. This task can be achieved using different power reduction techniques, applied at different levels, mainly logical level[9]. Further, there are two ways to improve an algorithm's performance from the process point of view. One can choose a dense but slow technology such as silicon CMOS and increase performance by parallelizing the algorithm or flattening the logic. Alternatively, one can choose a fast but low-density technology such as silicon ECL or GaAs DCFL. The GaAs DCFL offers higher density than silicon ECL, which is the highest-performance bipolar silicon technology, but cannot yet compete with silicon CMOS, the densest silicon technology [7]. Compared with CMOS, GaAs is faster by a factor of two to three while power consumption favours GaAs only at clock frequencies higher than 100 MHz [7]. The target technology process in this work is the CMOS 0.18μm from TSMC using fully ASIC methodology, as the full-custom front-end kit is not available.

The ultimate performance can be substantially improved by optimized architectures: regular high-density structures operating at low voltages with clock gated. Optimization aims at finding a suitable balance between speed, power consumption, and silicon area. The area can be traded for higher throughput or lower power.

1.7 SCOPE OF THIS WORK

In this thesis, appropriate design methods have been undertaken for spanning the range from algorithm to VLSI implementation without sacrificing performance (throughput/speed) using a low-power sub-micron oriented design. We investigated the options arising from very different fields, among these are low-power design methodologies and flows, long operands arithmetic in large $GF(2^m)$, cryptographic primitives and cryptanalysis. Our task was focused on trying to bring together some approaches such as memory-less design, massively/partially parallel architectures, high regularity design, low-voltage and clock gated circuit, power analysis and power optimization techniques for digital CMOS VLSI design at the gate level.

The major topics which, have been investigated are:

- Low-power automated design methodologies and flows.
- Arithmetic in large Galois field $GF(2^m)$ targeting hardware development and integration of low energy-delay product architectures and providing a public-key cryptographic schemes (i.e. modular exponentiation over large $GF(2^m)$).
- Exploring block and stream ciphers from the hardware point of view, mainly the class of LFSRs based stream cipher.
- Exploring the state of the art in the construction of secure Keyed Hash Functions KHF from a one-way hash functions for application in SSSC.
- Designing SSMG (Self-Synchronized Mixture Generator) based on a KHF suitable for hardware implementation and used for packet fingerprint identification. The packet digest is then used for initializing the stream cipher.
- Securing Real-Time MPEG-2 stream using different schemes of selective and full encryption to cover the main aspects related to the performance analysis during the encoding phase (encryption) and Real-time playback (decryption) of different strong ciphers. Our SSMG have been validated using software implementation in the frame of MPEG-2 video stream encryption.
- Development of PK-SSMG cryptosystem: dedicated for securing multimedia contents in conformance to MPEG-2 specifications for applications such as DVB and DBS. PK-SSMG is a public-key stream based encryption in the class of discrete logarithm schemes (arithmetic in Large Galois Field $GF(2^m)$, including the SSMG. The architecture is programmable on security and implements a security level up to 1279-bit key size. Beyond the arithmetic unit, the general structure and behaviour have been optimized in order to implement a low-power architecture.
- Efficient hardware implementation of Rijndael block cipher on FPGA as an IP (the external interface of the module (architecture) could be coupled with a microprocessor bus) for data confidentiality using memory-less architecture. The algorithm is implemented in iterative looping approach including the key-schedule and the round function. The study includes ASIC for power analysis and FPGA-based performance evaluation of this new architecture.

1.8 COMPLEXITY

Although, it appear easy or more convenient to build cryptographic primitives from scratch, it is by far the desirable and advised method. Cryptography cover a vast span of knowledge coming from different fields: number theory, complexity theory, information and Coding theory, probability theory, abstract algebra (finite field theory) and formal analysis. A strong mathematical back-

ground is required. A little knowledge is dangerous, inexperienced cryptographers almost always design flawed systems. A good cryptographic system strikes a balance between what is possible and what is acceptable. Designing with security objectives in mind to achieve performances constraints of new cryptographic primitives is by far an easy task. What can help the cryptographer is designing systems based on existing secure or well-studied primitives design. With this in mind, though there exist a huge solutions for providing data information security services, it leads me to believe that some cryptographic primitives are not well designed for hardware implementation. Is it possible to reduce the hardware complexity and increase the performances of such primitives?.

Furthermore, some primitives are not yet well addressed and a lack of solutions is noticed. This is the case of self-synchronizing stream cipher. Is it possible to design a secure or strong stream cipher?.

In the other hand, other than classical performance evaluation and optimization, mainly area and speed, is it possible to design high performances architectures (low-power/high speed) implementing strong cryptographic primitives?.

In this work we try to answers these questions and give some solutions by finding the good compromise between speed, area, power and security. The latter constraint rely on the security of strong, well defined and enough cryptanalyzed building blocks, while an effort is conducted toward designing new primitives components based on these blocks and the way to combine them. Practical proposals are provided and implementations are done, which provide some interesting results that could be an added value to both cryptography field and low-power design.

1.9 STRUCTURE OF THIS THESIS

The outline of this thesis is as follows. Chapter 2 presents some facts related to power consumption in CMOS circuit, methodologies and techniques for designing for low-power. Chapter 3 provides knowledge of cryptographic primitives and their hardware modelization based on Finite State Machine (FSM). In Chapter 4 finite field arithmetic mainly multiplication and exponentiation operations are presented. It focuses on implementing a suitable low-energy oriented architecture performing the modular exponentiation operation over large $GF(2^m)$ and trying to investigate the possibilities of further improvements in the circuit performance, complexity and especially power consumption by reducing the switching activities using digit-serial and clock gating techniques.

In Chapter 5 we explore stream cipher design with a particular emphasis on adding self-synchronization mechanism without increasing the length of the ciphertext. A special *cipher feedback* technique is presented and implemented, which performs the function of self-synchronizing mixture generator in combination with clock controlled LFSR stream cipher for pseudo-random stream generation. The approach is particularly secure and useful for applications requiring data transmission over a network/broadcasting channel. It keep both sender and receiver in synchronization without increasing the bandwidth and keeping computational overhead (delay) as low as possible by reducing the critical path and increasing the clock frequency.

In Chapter 6 the SSMG architecture is combined with the resulting architecture implementing the modular exponentiation operation in large $GF(2^m)$ for building a public-key stream based cryptosystem PK-SSMG. The concept of PK-SSMG has been validated using software and hardware implementation. The architecture structure is a direct mapping of the cryptosystem algorithm. A first block stage performs the modular exponentiation in order to generate the key for the initialization of the stream cipher. A second fast block performs the streaming with self-synchronization

capabilities. The cryptosystem has been verified at the gate level and adapted for DVB and DBS applications aiming to provide confidentiality for multimedia content in conformance with MPEG-2 system layer standard. It supports SPI interface according to the MPEG-2 codec application requirements and doesn't increase the bandwidth. Some interesting results obtained at the gate-level and post-layout have been reported.

Further, different schemes of selective and full encryption have been investigated and are presented in Chapter 6, which cover the main aspects related to the performance analysis during the encoding phase (encryption) and Real-time playback (decryption) of different strong ciphers and our SSMG algorithm in the frame of MPEG-2 video stream encryption.

Chapter 7 introduces Rijndael block cipher and describes an efficient memory-less FPGA implementation. In this chapter the efficiency of the circuit generated by this technique is also discussed and compared with further existing implementations. Finally in Chapter 8, conclusions are drawn and suggestions are made for future work and improvement issues.

CHAPTER 2

2 DESIGNING FOR LOW-POWER/LOW-ENERGY

In this section, a survey of techniques for low-power CMOS design are presented. These are the most important techniques yielding to significant power savings that are used in this work. There exist many other general and specific techniques that are illustrated in the vast literature. An extensive and detailed overview of the scientific activity in this area is given in [8]. Further, we discuss methods of power analysis, estimation, and related CAD tools. We describe our methodology for power metric reporting and optimization. Our emphasis during power estimation is on pattern-independent simulation technique while our strategy for synthesizing circuits for low power consumption is to restructure/optimize the circuit to obtain low switching activity factors at nodes, which drive large capacitive loads.

2.1 MOTIVATION FOR LP DESIGN

Silicon technology advances have made it possible to pack millions of transistors, switching at high clock speeds, on a single chip. Consequently, power consumption has become an overall concern for the IC industry. It is no longer solely a problem for portable applications since low power, yet high-throughput and computationally intensive circuits are becoming a critical domain. One driving factor behind this trend is the growing class of personal computing devices as well as the rapidly increasing demand for portable devices. Another critical factor is that excessive power consumption is becoming the limiting factor in integrating more transistors on a single chip. The need for low power has caused a major paradigm shift, in which power dissipation is as important as performance speed and area.

The optimization of ICs with respect to power consumption has well known advantages. The primary factors for lower power dissipation in VLSI circuits are the cost associated with packaging and cooling in large high performance circuits as the heat may limit the feasible packaging and reduce the device reliability. It helps also to enhance the run time of battery operated portable applications for portable applications.

2.2 SOURCE OF POWER IN CMOS CIRCUIT

Power consumption in standard CMOS technology originates from two different sources:

- Static power is dissipated in several ways. It is caused by *through currents* which are caused by transistors operated in their saturation regions and by leakage currents. The largest percentage of static power results from source-to-drain sub threshold leakage, which is caused by

reduced threshold voltages that prevent the gate from completely turning off. Static power is also dissipated when current leaks between the diffusion layers and the substrate. For this reason, static power is often called leakage power.

- Dynamic power caused by charging and discharging capacitors during signal computation (Switching power). Short circuit power occurs also in the dynamic phase when both nMOS and pMOS transistors are conducting, and

Hence, the total power dissipated in a CMOS gate with a capacitive load C_{load} is given by,

$$P = \frac{1}{2} \cdot C_{load} \cdot V_{DD}^2 f \cdot N + Q_{sc} \cdot V_{DD} \cdot f \cdot N + I_{leak} \cdot V_{DD} \tag{2-1}$$

Where V_{DD} denotes the voltage swing, and f is the frequency of operation, N the activity factor, i.e., the number of gate output transitions per clock cycle. The factor Q_{sc} represents the quantity of charge carried by the short circuit current per transition and I_{leak} is the leakage current.

In traditional design the average power consumption of a CMOS gate is dominated by the switching activity (dynamic power) and contributes to more than 90% of the total power consumption [9]. In certain circumstances, however, static power can be significant and may even dominate the power consumption of a given circuit, as with certain memory structures. For recent technologies (deep sub-micron) short-circuit current and leakage current may be neglected, but this may change for future developments of high scaled integration [10]. As the device size and threshold voltage continue to decrease, the short-circuit power dissipation is no longer a negligible factor. Reducing the power consumption amounts to the reduction of one or more of these factors. In energy-efficient design, we seek to minimize the energy consumed per operation or the energy-delay product of the circuit, which is the factor of merit for high performance architectures.

From (2-1) it turn out that lower supply voltages can achieve extremely low power consumption. However, lowering supply voltage leads to performance degradation. Delays drastically increase as V_{DD} approaches the threshold voltages V_t of the device (see Equation2-2).

Since the delay time is proportional to $1/V_{DD}$, the supply voltage can be reduced to a certain value, so that the chosen frequency matches with the longest critical path. The propagation delay equation of a CMOS circuit is given by [11],

$$T_{delay} = \frac{C_{load} \cdot V_{DD}}{k \cdot (V_{DD} - V_t)^2} \tag{2-2}$$

Where k depends on the transistors aspect ratio (W/L) and other device parameters, V_t is the transistor threshold voltage.

When the propagation delay is less than the clock period by a factor δ, we can reduce the supply voltage by a factor β such that T_{clk} is equal to T_{delay}. Hence,

$$T_{clk} = \delta T_{delay}(V_{DD}) = T_{delay}(\beta \cdot V_{DD}) = \frac{C_{load} \cdot \beta \cdot V_{DD}}{k \cdot (\beta \cdot V_{DD} - V_t)^2} \tag{2-3}$$

Parallelism and pipelining can be exploited to improve the performance (to compensate for the increased gate delays) of low-voltage circuits [9][10]. Also, much higher reductions in power consumption are possible when using clock-gating technique in order to reduce the activity factor N in

(2-1). Further, increasing the concurrency of internal operations, and rearranging the gate topology from array-type to tree-type reduces the switching power.

2.3 CMOS LOW POWER DESIGN TECHNIQUES

Lowering the power of a VLSI circuit can be achieved at various levels: fabrication technology, circuit optimization, logic design, control and clocking strategies, architectural partitioning and layout, and the underlaying system's algorithm and interface. Mainly, these techniques could be subdivided into two main categories: high level and physical level power optimization techniques. In this section, instead of classifying these techniques per their category or level at which the optimization can be done. We expose different strategies to reduce the power according to parameters in (2-1) that contribute in the calculation of power dissipation in static CMOS circuits. It should be noted that using one of such techniques may have an effect by either reducing of increasing other parameters.

2.3.1 POWER SUPPLY AND TECHNOLOGY

2.3.1.1 VOLTAGE SCALING

Power is proportional to the square of the supply voltage V_{DD}. Although the relationship is complicated by the fact that N is also voltage dependent. This makes V_{DD} reduction as the most effective way for reducing power, and the industry has thus steadily moved to lower V_{DD}. This trend should and will continue. Clearly the design engineer does not normally have control over V_{DD} in an automated design unless a full- and or semi-custom methodology is used. However, developments in fabrication are already scaling down the voltage as soon as process geometries continue to shrink and new sub-100*nm* fabrication technologies are developed.

One of the main motivations in technology development has been to increase the levels of integration by reducing feature sizes. However, as gate lengths are reduced (without reducing voltage levels) the electric field strength increases in the gate region. Another issue with voltage scaling is that to maintain performance, threshold voltage V_t also needs to be scaled. At low V_t, leakage power starts becoming a bigger factor[12]. Lower V_{DD} also introduces noise and reliability concerns.

2.3.1.2 TECHNOLOGY MAPPING

Power savings from cell libraries can come from two sources: device sizing and logic and physical layout structures of cells. As the cells and transistor devices shrink into deep sub-micron <200 *nm*, the operating supply voltages can be reduced to an acceptable level of circuit functionality with a trade-off between delay and power consumption. This has forced also, interconnect to be an important factor of overall system power.

Mainstream design methodologies are emerging that actively seek to manage and minimize power in system-level ICs. While global approaches such as selecting low-voltage technologies and low-power logic families are effective, engineers can normally also significantly impact power through design itself, i.e., minimizing wire-lengths by exploiting locality (see Section 2.3.3.2).

2.3.2 REDUCING THE ACTIVITY FACTOR

Power is only expended when a node is switched; if switching is restricted to when information changes then power is minimized. This can be summarized by the term *transition avoidance*. As a first

observation, this argues against the use of circuit styles, which involve precharging and discharging as a part of logic evaluation.

2.3.2.1 GLITCH AVOIDANCE

With some digital circuit, there are spurious transitions (known as *glitches*), which occur typically in combinational circuits due to partially resolved functions or unequal delays of gates paths (due to signals with skewed arrival times). A typical example of power loss through spurious transitions is the ripple adder. In this logic design, each bit-adder unit passes its carry to the next unit in a carry-chain; the value of its own input is not, however, valid until all the less significant bits have been resolved; thus each carry-bit in the chain may change (along with the corresponding sum outputs) as the valid carry signal propagates along the chain.

In order to reduce these spurious transitions, the delays of paths that converge at each gate in the circuit should be roughly equal. By selectively adding unit-delay buffers to the inputs of gates in a circuit, the delays of all paths in the circuit can be made equal. This addition will not increase the critical delay of the circuit, and will effectively eliminate these needless transitions. However, the addition of buffers increases capacitance, which may offset the reduction in switching activity. Therefore, one should evaluate a cost function by introducing a minimal number of unit-delay buffers instead of reducing completely eliminate the spurious switching power.

2.3.2.2 CLOCK AND SIGNAL GATING

Large VLSI circuits contains units and control logic that are not often accessed in each clock cycle. Similarly, in an arbitrary sequential circuit, the values of particular registers need not be updated in every clock cycle. If simple conditions that determine the inaction of particular registers can be determined, then power reduction can be obtained by gating the clocks of these registers [10]. When these conditions are satisfied, the switching activity within the registers is reduced to negligible levels.

Some CAD tools are actually offering an automatic RTL clock gating that is easily configurable, automatically implementable, which allows maximal reduction in power requirements with minimal designer involvement and no software involvement. However, clock gating cannot be used indiscriminately since there are some issues that need to be considered. An important concern is that the disabled clock may not power up in time, or that modified clocks may generate glitches. This imposes strict timing constraints on the enable signals and calls for careful timing verification [8]. In addition, at clock frequencies of 100 MHz and above, clock skew becomes critical and every extra gate used to qualify the clock can potentially introduce timing critical skews. Appropriate buffers could be used to overcome this problem during clock tree construction. Large buffers generally have good insertion delay and skew characteristics, but consumed much more power than necessary.

Many other effective power-reduction approaches are available at the RTL. Signal gating is conceptually similar to clock gating but focuses on reducing transitions in non-clock signal. This scheme is often employed on decoders. Since a change on a signal decoder input results in two output pins switching state, it's desirable to block the outputs from switching when decoder updates aren't required. This can be accomplished by using an enable on a decoder. Another example concern the design of datapaths containing multipliers. Power can be saved by gating or registering the multiplier inputs. The registers are clocked only when the multiplier is active, thus reducing the number of unnecessary multiply operations.

2.3.2.3 POWER-DOWN MODES OF GLOBAL CLOCK GATING

Large reductions in power can be achieved using *power-down* techniques. These methods are used to disable large functional blocks when they're inactive. The blocks are disabled by mechanisms analogous to those used with clock gating, except that in this case the gating is done at a global level instead of at the register or local level. This then shuts off the operation of entire units such as execution units, bus controllers, or memory managers.

Compared to local clock gating, the use of global clock gating usually results in much larger power reductions. Power savings of 90 percent or more can be achieved in applications such as *laptop* computers and cell phones. This technique isn't limited to traditional low-power designs but can also be used in many high-performance applications to minimize overall average power consumption.

2.3.2.4 ALGORITHM OPTIMIZATION (REVIEWING)

Component power consumption corresponds to the usual algorithmic performance criterion of speed since algorithmic speed is a function of the number of operations and this translates into the component as the amount of switching. Thus, reducing the number of steps in a computation will naturally reduce the power of its implementation.

Further, the most obvious approach to reduce the switched capacitance, is to reduce the number of operations (and hence the number of switching events). While reducing the operation count typically has the effect of reducing the effective capacitance, the effect on critical path is case dependent. This trade-off is clearly illustrated when choosing between long operators arithmetic functions in bit-serial and bit-parallel architectures.

Another factor which can be useful under algorithmic transformations consist of operation substitution. Certain operations inherently requires less energy per computation than other operations. A prime example of transformation which explores this option and often used in software compilers is multiplications. These operations are substituted by additions. Although this situation is not of common use in CMOS LP design, some times it is possible to realize significant power savings using this approach.

2.3.3 REDUCING THE CAPACITIVE LOAD

2.3.3.1 PARTITIONING INTO BLOCKS

As general rule, it is best to partition large blocks into smaller ones. This principle can be of benefit when implementing large memories. The power consumption of a memory access operation is based upon the capacitances of bit and word lines, which run vertically and horizontally across the whole array. It, instead, when the memory array is broken down into sub-units (each with its own support circuitry) and only one unit is addressed with each access, then the product of activity and capacitance can be significantly reduced. The same technique is used in the *CoolRISC* [13] called memory pagination.

2.3.3.2 LOCALITY OF REFERENCE

This is a design philosophy in which signals are generated and used locally in terms of their physical location on the silicon surface, since the further a signal has to travel, the higher is the capacitance of that connection. With signals being processed locally, there is greater opportunity for parallel execution and thus greater throughput which could be traded-off for a lower supply voltage and so lower power consumption.

Another reason is related to the new fabrication technologies. Interconnections are achieved using metal layers. For large feature sizes the RC delay on such lines is relatively small compared with the transistor delays in the circuit. However, while transistor delays scale down with feature size, the RC delays actually increase as fringing capacitance begins to dominate the total capacitance (and doesn't scale) and the resistance increases as the interconnect lines become narrow. Thus, in submicron technologies, interconnect capacitance dominates total gate loading and resistance increases, because of shrinking wire widths that is, the communication delays predominate [14]. Using locality of reference as a style avoids the major potential source of delay. Architectural strategies based upon this idea may include: processing of data locally to where it is stored, communication only with physically adjacent functional units, and dedication point-to-point buses rather than shared ones. This technique is useful unless it does increase the total capacitance of such buses when compared with the capacitance of single shared bus.

2.3.3.3 CLOCK AND CONTROL SIGNALS DISTRIBUTION

In architectures with distributed processing, the question arises as to whether there should be global control and clock signals. The problem is the widely distributed clock signals. The global distribution network has a very large capacitance and is switched frequently. There are several possible strategies such as *true single phase clocking* (TSPC) [15] and asynchronous design using *self-timed* circuits [16]. These techniques are out of this work since the former is a custom design methodology and because of the lack of appropriate verification tools for the last one.

2.3.3.4 BUFFERING THE HEAVILY LOADED LINES

One recurrent problem is the design of circuitry to drive a relatively large capacitance (particularly external loads). The basic solution is a sequence of buffers with increasing gate widths; the design issue is what should be the size ratio α of each successive buffer. Once more, there exist a trade off between speed and power. With speed as the main consideration, one can choose a large buffers to overcome the driving capabilities. However, large buffers consume more power. If power is the main issue, one can evaluate a cost function to determine α, which give the best trade-off between power and delay. More detail related to this approach is given in Chapter 4.

2.3.4 REDUCING THE SHORT CIRCUIT CURRENT

We need to consider short-circuit current in two ways: first, how to minimize what is unavoidable, second how to avoid what is unnecessary.

2.3.4.1 RESISTIVE NETWORKS

Firstly, some logic styles deliberately use resistive networks formed from transistors to establish the value of the output signal (e.g. pseudo-NMOS). These styles cannot be used for low-power design. Secondly, some strategies for avoiding power loss involve generating multiple voltage levels using resistive networks either on-chip or at the system level. This static power loss must be carefully included in the evaluation of such strategies.

2.3.4.2 SWITCHING CURRENT

Even conventional static CMOS gate has a source of short-circuit currents. As input to a CMOS gate changes, there is a period during which both nMOS and pMOS networks are switched ON, that is when the input voltage is between $(V_{DD}-V_{tp})$ and V_{tn}, where V_{tp},V_{tn} are the threshold.voltages of the pMOS and nMOS transistors respectively. During this period, there is a short-circuit

current and so power dissipation. This is clearly dependent upon the rise and fall time of the input signal. For poorly designed circuits, this power loss can be significant. A simple rule-of-thumb for designers is to size the transistors of cells so that the delay in the output is the same as that of the input; with this strategy, the short-circuit power loss is reduced to insignificant factor of the dynamic power dissipation[17].

2.3.4.3 GLITCH PROPAGATION

In Section 2.3.2.1 spurious transitions were explained in terms of unit time delays and how to reduce the unequal paths delays that converge to gates in circuits. In fact the problem is compounded in that the output glitch propagates on to other stages. In practice this signal often takes the form of a slowly varying voltage, which hovers in the centre of its range causing short-circuit currents in the next gate. This is another source of power dissipation that contribute to increase the second term of (2-1) and a further reason to avoid logic glitches.

2.4 POWER ESTIMATION AND OPTIMIZATION

Although it is well accepted that design is not a purely top down process, it is a convenient method of describing the low power activities at different levels of the design data hierarchy as well as the flow of data and control. This section discuss the power issues in the context of a CAD system to support design for low power.

The EDA industry has produced several analysis tools that help designers quantify power problems and some optimization tools to quench excess dissipation. More tools are needed; they must fit existing methodologies and the tools must be internally consistent between analysis and optimization and across all levels of abstraction. Power problems are often trickier to solve than timing problems because power is more pattern-dependent than timing. Low-power design techniques often conflict with area and timing constraints. Power management is ultimately about optimization tools. Designers require automation to achieve fast objectives. Analysis should ideally be relegated to a back-end validation step.

2.4.1 POWER ANALYSIS AND OPTIMIZATION TOOLS

Fast and accurate power estimation tools are needed at each level of abstraction in order to analyse the power respectively energy consumption of a design and to validate that given design constraints are met. The existing power estimation tools reflect the trade-off between accuracy and speed: The faster the program the less accurate the results, especially if we move up in the hierarchy of abstraction levels.

A direct method for power analysis is to translate the given high-level architecture description to the gate, circuit or physical level; at which point reasonably accurate low-level power analysis tools can be used. This method is obviously infeasible if a large number of design alternatives have to be evaluated. While power estimation has some application at the early stages of a design process, power measurement only rough guesses are made about the total chip power.

Although high-level power estimates are critical in guiding the design process and meeting the power budget, **EDA industry has concentrated on power analysis tools, both in the areas of transistor, gate and RTL. Very few companies concentrate on RTL analysis.** There are at present few power estimation and optimization tools at the RTL and architectural levels of abstraction. Efforts are under way by both industry and academia to fill this gap. In the other hand, while CAD logic

synthesis and gate/transistor sizing tools have respectable but limited impact, the answer to power reduction lies in design techniques.

2.4.2 LP AUTOMATED DESIGN METHODOLOGY

2.4.2.1 PREPARING FOR RTL DESIGN

The first decision is the choice of fabrication technology. Low supply voltage is good. Small feature size is very good because of the low capacitance and increased device speed due to the short channel length. The technology library used in this work is CMOS 0.18μm process from TSMC with 0.9 and 1.8v operating voltage.

The general approach is based on evaluating and optimizing the algorithm in order to reduce the number of operation then evaluating different suitable architectures for implementation. At this level and with a judicious choice of architecture topology, power reduction can be achieved by minimizing the capacitance switched at different nodes. There are various topological choices for implementing a given function. Once the architecture is fixed, the next decision is the design partitioning. We should avoid architectures based on central processing units and always review the specified algorithm. Splitting the architecture into small independent units (avoiding high capacitance interconnects) operating in parallel if possible or adopting a partially parallel processing with pipelining (raising throughput).The method is to apply the principle of locality of reference. Other design strategies and choices are:

- Choosing Regular structures.
- Avoiding storage or memories (memory-less design), i.e., on-chip computations.
- Since energy efficient design is better than low power design, low energy is targeted.
- Since, the target is high performance circuits, the predominant metric is defined as Energy-delay product.
- Working at bit-level rather than using standard packages and functions.
- Preparing the design for clock gating.

Power consumption of currently used static CMOS-technologies is dominated by dynamic power consumption (except for very low-voltage technologies), i.e. the circuit activities need to be analysed. Precise simulation tools on circuit level like *SPICE* can not handle the complexity of large circuits and huge number of possible stimuli. For this reason our methodology for power calculation is based on circuit activity analysis at logic level.

2.4.2.2 PRE-SYNTHESIS DESIGN IMPACT

In an automated design, a significant percent of the power in a large ASIC is committed by the time the RTL design is finished. Effective system-level integration methodologies require power issues to be addressed prior to synthesis. It is important to observe that there are several types and sources of power, each having advantages and drawbacks that vary with the particular design. The most significant criteria are:

- **Circuit Function:** Power comes from several different types of structures among these are clocks, memory, I/O, datapath, control. Depending on the IC, some or all of these may be important. In a high-end communications chip, for instance, half the total power may be dissipated in the memories while in a processor, the clocks or datapath are more likely to dominate.

- **Static vs. dynamic power:** Most power-reduction methodologies address dynamic power, which occurs as charge is dumped into and out of capacitances during switching cycles. Dynamic power tends to dominate in large IC designs. We have to consider static power, which is largely independent of switching activity and can also have a significant impact.
- **Average vs. peak power:** Most chip temperature analysis relies on steady-state temperature estimates based on average switching activity in a design. Peak power is more difficult to calculate because it varies with the specific vector set applied; there is not any statistical average signal. Considerations such as electromigration and IR drop are much more related to peak power than to average power, and either one should consider a design revision.

Although capacitances, and under certain conditions even supply voltages, can be influenced by RTL/architecture design, the factor most accessible to us at this level is the effective switching frequency. A key concept is to minimize energy waste due to operations and signal transitions.

2.5 LOW POWER DESIGN FLOW

Figure 2-1 shows our ASIC low-power design flow using an EDA methodology that automates power analysis and optimization tasks and makes those tasks an integral part of the design flow from RTL to GDSII. This methodology is based on Synopsys tools for front-end part and Cadence tools for back-end or physical design, as shown with more details in Figure 2-2. This methodology has been validated to ensure that circuit meet power, timing and area goals. Obviously, in many cases the area constraints has been relaxed in order to make sure that this constraint is not limiting power optimization, at the same time the area level is controlled. The flow is based on scripts that automate the RTL optimization up to post-layout analysis with minimal intervention.

2.5.1 SIMULATION BASED POWER ESTIMATION AND OPTIMIZATION

Unlike static timing analysis, accurate power analysis is vector dependent in both probabilistic and simulation-based modes [18][19]. Therefore, the selected vector set for power analysis/optimization will dramatically affect the correlation between the power numbers derived from the power analysis/optimization tools and the actual power consumed by the application. The best vectors set depends on the application whether the average power or peak power is concerned. This rely on the application and implies different considerations when selecting the power vectors. The average power dependent applications typically include the battery operated applications for longer battery life, cost sensitive chips including high volume productions such as the consumer electronics in order to reduce the temperature and thus the packaging and cooling costs. Telecom and military applications deal with average power consumption in order to provide less electromigration, and therefore higher reliability. Peak power dependent applications typically include noise sensitive applications and power supply sensitive systems in case where the power supply will not be able to provide high peaks of current.

2.5.1.1 SELECTING THE BEST VECTOR SET

In this work we deal with the average power dependent applications, in which the power vectors should closely imitate the day-to-day operation of the design chip in its target system. The key is the closer the vectors are to the actual application's operation, the better correlation between the power number from power analysis tools and the actual system power consumption. These vectors can be chosen as a set of random vectors that reaches the maximum circuit's internal states (nets) for higher margin (exercises a majority of the logic in the design). Nevertheless, in most applica-

tions, the functional validation vectors are a good starting point for choosing the best power vectors.

2.5.1.2 ASIC LIBRARIES

Synopsys Power Tools gives an accurate power estimation and optimization as long as the technology library has power information and the design is handled at gate level including accurate timing and parasitics information. When the Technology Library is not characterized for power, it is possible to report power numbers. This is because net switching power is a function of pin capacitances, voltage, and toggling frequency. However, this report is not accurate since the internal cells power cannot be reported. Thus, the library cells must have lookup tables for internal power and pin capacitance. Furthermore, it should be characterized for leakage power as well in order to optimize for leakage power.

2.5.1.3 VSS-FAIF INTERFACE

Simulation to capture switching activity at both RTL and Gate level uses VSS-SAIF interface as shown in Figure 2-2. It allows to easily use SAIF files with VSS and Synopsys *DesignPower* tool for power back-annotation and power reporting. *DesignPower* and *PowerCompiler* uses a probabilistic engine to estimate the switching activity on the nets based upon the functionality of the design, the static probability and the toggle (switching) rate. Power analysis quality is directly related to the success of back-annotation of SA of the design. Power optimization is done using Synopsys *PowerCompiler* tool, which perform an incremental compile (resynthesize) for logic transformations.

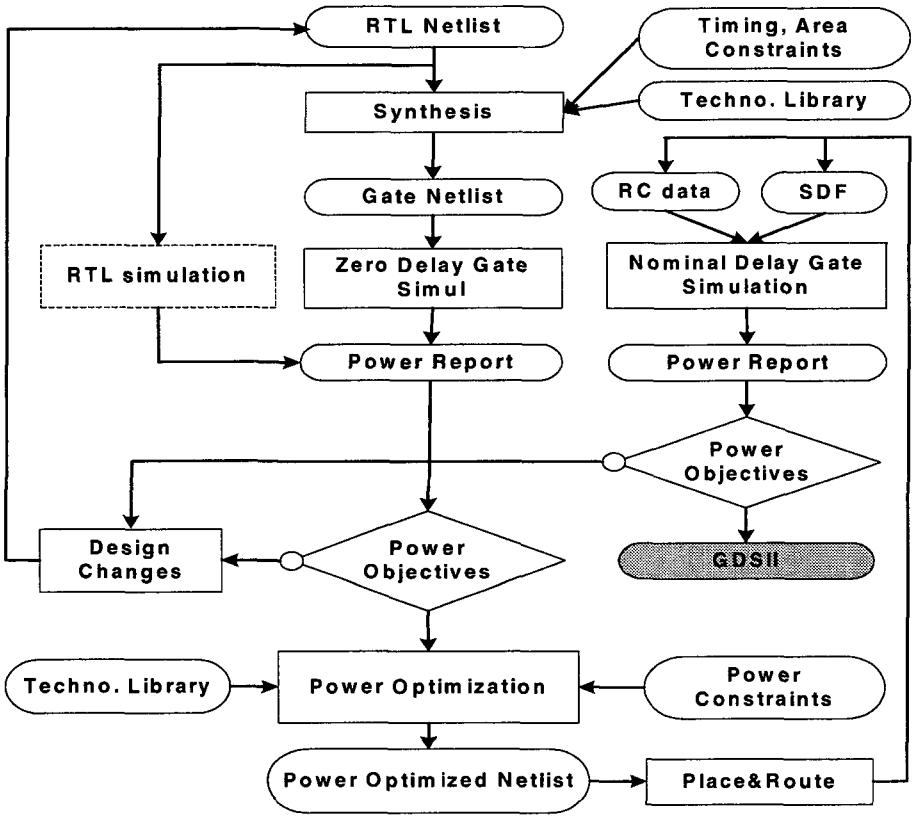


Figure 2-1: Low-power oriented design flow using sub-micron processes.

2.5.2 FRONT-END OR LOGICAL DESIGN

Generally, from the design specification a model is designed with a high-level language (typically C language). The C model allows resource analysis as well as the functional verification. From the resource analysis we can fix the architecture applied to design the circuit.

After fixing the architecture, we model the circuit at behavioural level with VHDL. We could also model the design within another level such as structural, PLA, etc. The behavioural description (RTL) is simulated before the synthesis step in order to verify its functionality, to correct the description and even in the worst case, modify the architecture.

For the synthesis part, the tool used in this work is Synopsys Design Compiler. With this tool, we can constraint the design with timing and/or area parameters. The source file is VHDL behavioural description (also accept structural, etc.), which is mapped in the logic level, with technology dependent standard cells. A logic netlist can be provided after logic synthesis in different formats. The logic netlist contains all logic connectivity of the design. Generally, we generate a VHDL format netlist for gate level simulation, which takes into account cell delay and of the whole design with VHDL Synopsys Simulator VSS. If the design meets the constraints we generate Verilog format netlist, which can be imported into floorplaning and place&route tool (Silicon ensemble). Design constraints can also be generated in SDF format in order to be imported into floorplanning tool.

2.5.2.1 BEHAVIOURAL/RT POWER ESTIMATION/REDUCTION

At the behavioural or RT levels switching activity can approximate the power consumption. Nets and buses switching activity factors obtained from the RT level logic simulation can quickly identify hot spots on which the designer's efforts should be focused in order to reduce the switching power the dynamic power by either changing the HDL description or the architecture. No accurate power estimation can be obtained at this level.

2.5.2.2 GATE LEVEL POWER CALCULATION AND OPTIMIZATION

At the gate level, an accurate power calculation is possible, limited only by the accuracy of the gate power models and the extracted parasitics. Early, in the design, estimated parasitics (based on wire load models which are much less accurate than parasitics extracted from the Place&Route tools at the physical level) coupled with a zero-delay simulation can provide enough accuracy compared to behavioural power estimation. Instead, post-wiring parasitics and nominal delay simulation are required for more accurate calculations later on. Gate level simulation, with timing, is more time consuming but the SA generated allows to capture activity of every element in the design and accounts for any glitches. Additionally, we can capture state and path dependent SA if the targeted library has state and path dependent power numbers. State dependency accounts for varying modes of operation; for example, the power a RAM dissipates varies depending on Read or Write mode. Path dependency accounts for varying power consumption based on various input to output path.

Much higher power reduction can be obtained at the gate level using power optimization techniques provided by Synopsys Power Compiler tool. Power optimization depends on many factors including power constraints, design specific (whether design rules and timing constraints are met or not), target ASIC library: typically a library with rich set of cells can provide better power results. Optimization is performed by means of buffers resizing and/or insertion, logic transformations, cells replacement, etc. At this point a decision have to be taken in order to change the architecture or to continue the flow if the power budgets are met (see Figure 2-1). Once the gate level design is

optimized on power the resulting netlist is placed and routed then the estimated parasitics are extracted and used for a nominal delay simulation. At this point, final accurate calculation of power are obtained.

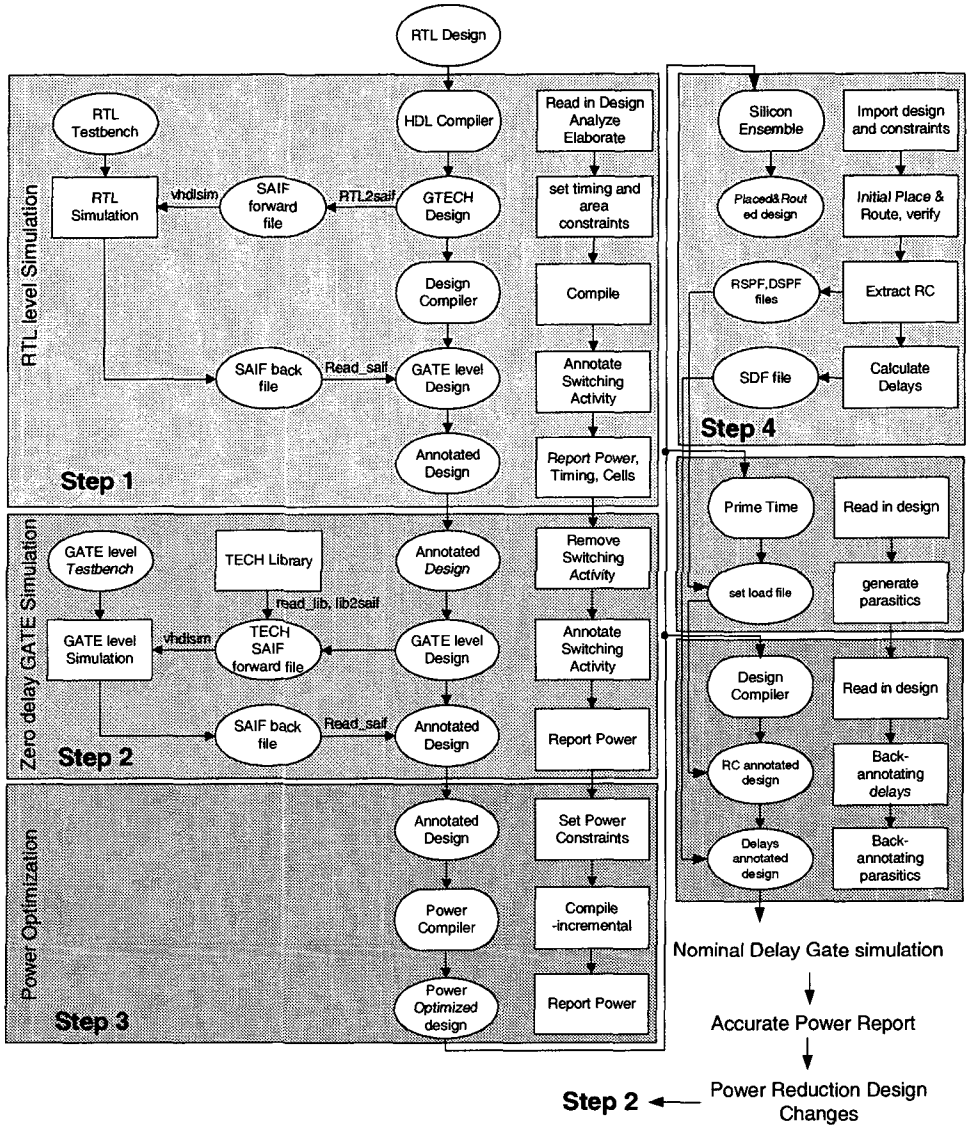


Figure 2-2: Detailed LP deep submicron design flow

2.5.3 BACK-END OR PHYSICAL DESIGN

Generally, designs are partitioned in different level of hierarchy. So the main idea of the floorplanning step is to process a block placing in order to optimize the interconnection between them. You can extract from your floorplanned design important parameters, which can be back annotated into the logic synthesis and simulation tools in order to re-evaluate the circuit. Back-annotation at this level helps to give more accurate performance estimation of the design. Back-annotation step can be performed under the ASIC designer's control and multiple and lengthy iterations between ASIC designers and foundry can be avoided. The design constraints (SDF file) can also be imported in order to lead the floorplan strategy. Once you have the topology information, a PDEF file can be generated as well as the estimated SDF file and back-annotated into synthesis tool. This step take

into account the physical information of the design and re-optimize it, if necessary with new physical parameters. Using Silicon Ensemble for floorplan, place and route a timing-based design.

When the design is accurately floorplanned and the design constraints are met, the floorplan can be placed and routed. Placement and routing tool imports DEF netlist, PDEF and SDF file from Floorplanner. After final place & routing step, we can extract exact design parameters which must satisfy the design constraints.

2.5.3.1 TIMING AND PARASITICS BACK-ANNOTATION

In many cases we will get the most accurate results with *PrimeTime*, if we back-annotate load data (using parasitics information generated at the physical level) and SDF files. This is because SDF files contain only cells and interconnections delays and in most cases setup and hold checks. It is useful for static timing analysis only. However, SDF files do not contain any information about capacitance or transition times. Without back-annotated capacitance, tools such as *PrimeTime* calculates transition times and wire capacitances based on wire load models, which are much less accurate than back-annotated loads. In addition, if the SDF is missing any timing checks, back-annotated loads can be used to calculate accurately the setup and hold timing checks, which are dependent on transition times. Also, if the SDF is missing any cell delays (i.e. providing interconnect SDF but not cell delay), back-annotated load can be used to calculate these delays because cell delays are dependent on input transition times as well as output loads. This situation happen when the SDF doesn't cover all of the timing arcs in the design, which could be the result of how timing arcs are modelled in the technology library.

2.5.3.2 PRIORITY IN BACK ANNOTATION

If both SDF and back annotated lumped RCs are available for a particular net or cell timing arc, SDF will take precedence in calculating the net or cell timing for that arc. Back annotated lumped RCs will also take precedence over wire load models.

CHAPTER 3

3 STATE OF THE ART IN DESIGNING CRYPTOGRAPHIC PRIMITIVES

This chapter describes some basic concept of cryptography and gives a brief summary and definitions related to the basic cryptographic primitives and their security. We focuse on the design of four cryptographic primitives: block ciphers, stream ciphers, hash functions and public-key schemes and expose their provable security properties. An overview of the design principles is given, which includes the global structure and modelization based on Cryptographic Finite State Machine CFSM. Also the basic of cryptanalysis attacks are ovorviewed and the main security concerns of such primitives are described. That is called cryptanalysis and it's important to enumerate the most useful attacks. The design of modern cryptographic primitives is guided by the security requirements and methods of the corresponding available attacks. For a complete overview and more details please refer to [68][69]

3.1 CRYPTOGRAPHIC PRIMITIVES

Depending on the required security services, there exist many cryptographic primitives from which to choose. Common primitives are stream/bloc ciphers for data confidentiality, hash functions and keyed hash functions (KHF) for data integrity and digital signature and public-key schemes for key generation and distribution.

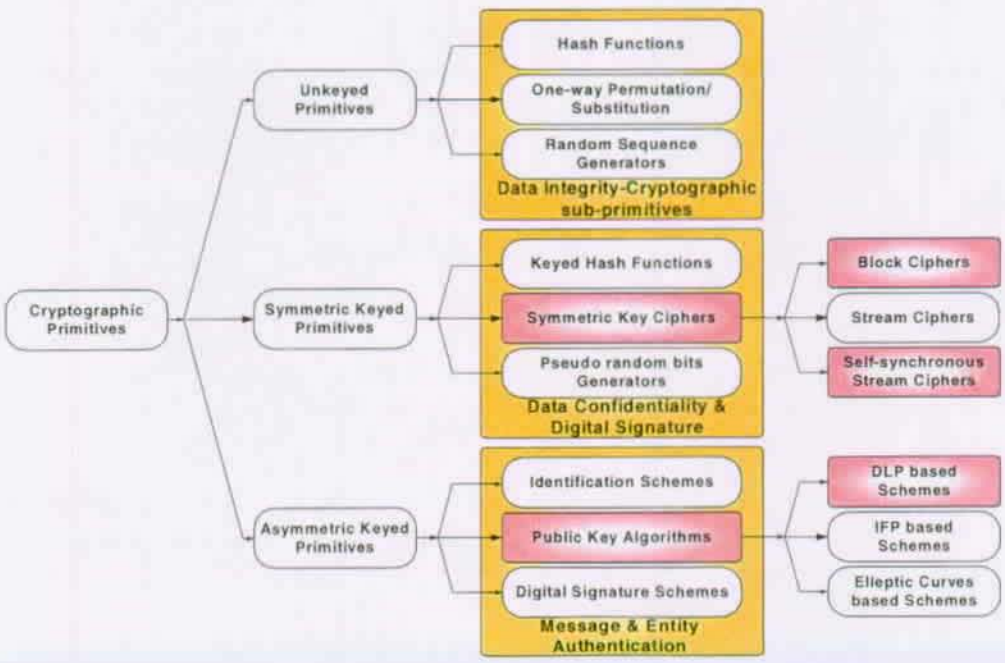


Figure 3-1: Cryptographic primitives

3.2 INFORMATION SECURITY OBJECTIVES

Information security services includes entity authentication, integrity, repudiation prevention, access control and confidentiality.

3.2.1 DATA CONFIDENTIALITY OR PRIVACY

Is a service used for rendering the data unintelligible to keep the content of information from all but those authorized to have it.

3.2.1.1 ENCRYPTION/DECRYPTION PROCESS

Encryption is the process of transforming a message (*plaintext*) into a form that is so it is *unintelligible* to anyone but the intended recipient. Encryption algorithms are invertible transformations so that encrypted messages (*ciphertext*) can be decrypted. All transformations using the encryption algorithm also called a *cipher* are referred to generally as a *family* of transformations. Within the family of transformations, modern encryption systems use a *key* or *seed* a parameter, which selects a particular transformation from the family of transformations or used as initial state. Cryptography is the science of making this transformation as intricate as possible, so that reversing it without certain key information is difficult, if not impossible. An important property of encryption algorithms is that unique keys will define unique encrypted messages or ciphertext.

Several encryption schemes are in wide use today they are mainly subdivided into tow symmetric ciphers that uses the same key is used for both encryption and decryption and asymmetric schemes that uses two different keys, one for encrypting and another for decryption. Although asymmetric schemes solve the problem of key exchange, they are rarely used for data confidentiality purpose to their bad performances. They are several order of magnitude slower compared to symmetric ciphers. On the other hand, according to their security and functional properties, symmetric key ciphers can be subdivided into two different primitives *block* and *stream* cipher.

3.2.1.2 CRYPTOSYSTEM

A cryptosystem can be defined as an encryption algorithm, a decryption algorithm and a specific key which produces an unique invertible transformation using the given algorithm. Ideally, a cryptosystem uses the key and the algorithm to produce a flat distribution for all properties of the message to be encrypted, hiding all natural redundancies of the language that up the message. It should appear to the attacker that the message represents random information.

3.2.2 DATA INTEGRITY AND AUTHENTICATION

Data integrity is a service which address the unauthorized *alteration* of data and prevent data manipulation (*insertion*, *deletion*, *substitution*) by unauthorized parties. Message authentication includes data *integrity*. It is a service which provide data origin *authentication* with respect to the original message source created at some time in the past.

Unkayed hash functions take a message as input and produce an output referred to as a *hash value* or *digital fingerprint* or *message digest* which serves as a compact representative image of the message. A one-way hash is a number of fixed length with the following characteristics:

- The value of the hash is unique for the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.

- The content of the hashed data cannot, for all practical purposes, be deduced from the hash, which is why it is called *one-way*.

These functions are used frequently in combination with digital signature schemes (see Section 3.2.3) to provide data integrity where a message is hashed first and then the resulting hash value as a representative of the message is protected (*signed*) in place of the original message. The problem of preserving the integrity of a potentially large message is thus reduced to that of a small fixed-size hash value. A distinct class of hash functions, called message authentication codes (MACs) may be viewed as keyed hash functions which take two distinct inputs, a message and a secret key, and produce a fixed-size digital fingerprint output of the message.

3.2.3 DIGITAL SIGNATURE AND VERIFICATION

Encryption and decryption address the problem of eavesdropping but do not address the other two problems tampering and impersonation. Digital signature is a service which address the problem of providing the digital counterpart to a handwritten signature. Signatures must be verifiable. Public-key techniques are generally used for digital signature generation and verification.

In fact, tamper detection and related authentication techniques rely on a one-way hash. With public key schemes it's possible to use the private key for encryption and public key for decryption. Although this is not desirable when we are encrypting sensitive information, it is a crucial part of digitally signing any data. Instead of encrypting the data itself, the signing counterpart creates a one-way hash of the data, then uses the private key to encrypt the hash value. The encrypted hash, along with other information, such as the hashing algorithm, is known as a digital signature. Figure 3-2 shows a simplified scheme of the way a digital signature can be used to validate the integrity of signed data.

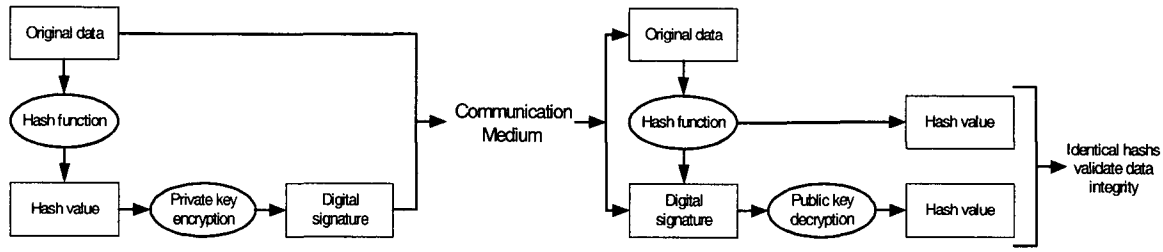


Figure 3-2: Digital signature for data integrity validation

The digital signature, which is basically a one-way hash of the original data is encrypted with the signer's private key and transferred to the recipient. To validate the integrity of the data, the receiver first uses the signer's public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data (information about the hashing algorithm is sent with the digital signature, although this isn't shown in the figure). Finally, the receiver compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. If they don't match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that doesn't correspond to the public key presented by the signer.

If the two hashes match, the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. Confirming the identity of the signer, however, also requires some way of confirming that the public key really be-

longs to a particular person or other entity. One solution is based on the key certification (see Section 3.4.3.1).

The significance of a digital signature is comparable to the significance of a handwritten signature. Once signing some data, it is difficult to deny doing so later, assuming that the private key has not been compromised or out of the owner's control. This quality of digital signatures provides a high degree of non-repudiation. That is, digital signatures make it difficult for the signer to deny having signed the data. In some situations, a digital signature may be as legally binding as a handwritten signature.

3.2.4 ENTITY AUTHENTICATION AND KEY ESTABLISHMENT PROTOCOLS

This service address techniques which provide shared secret key between two more parties for subsequent use as symmetric keys for encryption, message authentication, and entity authentication. These techniques are mainly based on public-key schemes and employ digital signature for authentication as described in the section above. Authenticated key transport (or entity authentication in this case) may be viewed as a special case of message authentication with privacy, here the message in Figure 3-2 is replaced by the key.

Key establishment is subdivided into key transport, which is the protocol where one party creates a secret value and securely transfers it to the others and key agreement is the protocol where a shared secret *ID* derived by two parties (or more) in such way that no other party can predetermine the resulting value. Diffie-Hellman key agreement [85] (see Appendix A) is a fundamental technique providing unauthenticated key agreement.

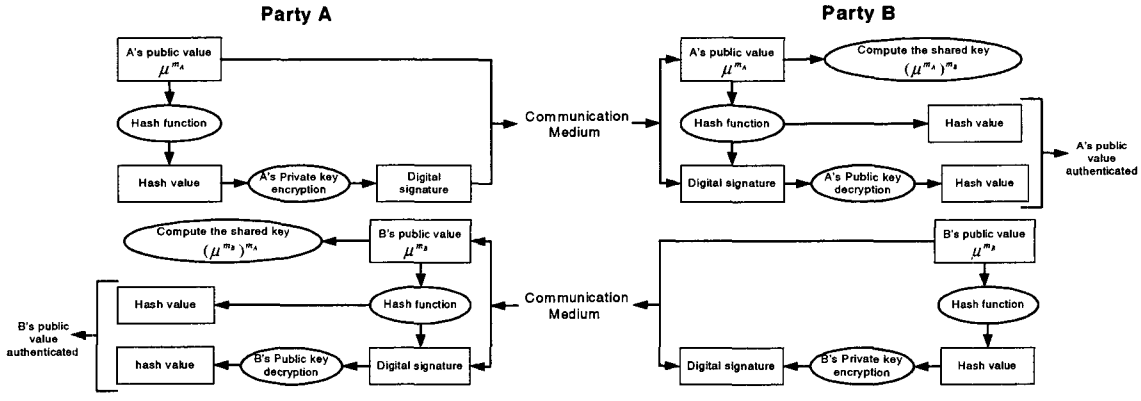


Figure 3-3: Diffie-Hellman Key establishment protocols (key authentication and key agreement)

An example of the key establishment protocols is the authenticated Diffie-Hellman key agreement protocol. The immunity against the man-in-the-middle attack (see Section 3.3.1.1) is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures and public-key certificates (see Section 3.4.3.1). The two parties involved in key agreement according to Diffie-Hellman algorithm described in Appendix A, each obtain a public/private key pair and a certificate for the public key. As described in Figure 3-3, during the protocol, both parts authenticate themselves (key authentication) using digital signature scheme on their public value generated by the Diffie-Hellman key agreement then send it to each other. Both part verify the digital signature of each other before establishing the secret key.

3.3 SECURITY CONSTRAINTS

Security requirements are defiant for each cryptographic primitives. A system is said to be computationally secure if the task of determining the key is computationally infeasible or *intractable.* A *perfectly secure* system is useful because the attacker does not have enough information from the ciphertext to determine the key. On the other hand, *computationally secure* systems are also useful because while the attacker has enough information in the ciphertext to determine the key, he does not have enough time to complete the task. Good cryptographic algorithms can never be fully tested. It seems to be common practice of government organizations and some companies to spread bad news about any encryption method other then theirs. When the strength of a cipher greatly exceeds the effort required to obtain the same information in another way, the cipher is probably strong enough and the mere fact that information has escaped does not necessarily mean that a cipher has been broken.

3.3.1 CRYPTANALYSIS (ATTACK STRATEGIES)

Cryptanalysis can be defined as general ways in which a cryptanalyst may try to *break* or *penetrate* the secrecy of a cryptographic primitive. These are not algorithms; they are just approaches called *attack*s as a starting point for constructing specific algorithms. Because there is no theory which guarantees strength for any conventional cipher, ciphers traditionally have been considered *strong* when they have been used for a long time without knowing how to break them easily. Cryptanalysis seeks to improve this process by applying the known attack strategies to new ciphers and by actively seeking new ones. The result is typically some value for the amount of *effort* which will achieve a break (even if that value is impractical); this is the strength of the cipher. But while cryptanalysis can prove weakness for a given level of effort, cryptanalysis cannot prove that there is no simpler attack. That is, **lack of proof of weakness is not proof of strength**.

Many academic attacks are essentially theoretical, involving huge amounts of data and computation. But even when a direct technical attack is practical, that may be the most difficult, thoroughly expensive and time consuming way to obtain the desired information. Success is never assured and resources are always limited. Consequently, approaches other than a direct technical attack for obtaining the hidden information (or the key!) can be more effective: making a paper copy, stealing a copy, cunning and bribery, theft and electromagnetic monitoring. No cipher can keep secret something, which has been otherwise revealed. Information security thus involves far more than just cryptography and even a cryptographic system is more than just a cipher. Even finding that information has been revealed does not mean that a cipher has been broken.

3.3.1.1 ATTACKS CLASSIFICATION

Classically, attacks were neither named nor classified; there was just here is a cipher and here is the attack. While this gradually developed into named attacks, there is no overall attack taxonomy. Currently, attacks are often classified by the information available to the attacker or constraints on the attack and then by strategies, which use the available information. Not only ciphers, but also cryptographic hash functions can be attacked, generally with very different strategies. When we are to attack a cipher, which enciphers plaintext into ciphertext or deciphers the opposite way, under control of a key, the available information necessarily constrains the attack strategies to these described as follow:

Attack	Description
Known Plaintext	A type of attack that allows the ciphering transformation to be examined directly in which the cryptanalyst has some quantity of related plaintext and ciphertext of an arbitrary message not of his choosing. It is surprisingly reasonable that the opponent might well have some known plaintext (and related ciphertext). The particular message of the sender is said to be compromised. In some systems, one known ciphertext-plaintext pair will compromise the overall system, both prior and subsequent transmissions and resistance to this is characteristic of a secure code.
Ciphertext Only	The attacker has only the encoded message from which to determine the plaintext, with no knowledge whatsoever of the latter. A ciphertext only attack is usually presumed to be possible and a code's resistance to it is considered the basis of its cryptographic security.
Known Plaintext:	We have some, or even an extremely large amount, of plaintext and the associated ciphertext.
Defined Plaintext:	We use arbitrary messages to be ciphered and capture the resulting ciphertext. We use also Chosen Plaintext and Adaptive Chosen Plaintext.
Defined Ciphertext	We can use arbitrary messages to be deciphered and see the resulting plaintext. We can use also Chosen Ciphertext and Adaptive Chosen Ciphertext.
Chosen Key	We specify a change in any particular key bit, or some other relationship between keys.
Fault Analysis	We induce random faults into the ciphering process, and use those to expose the key.
Meet-in-the-Middle	Given a two-level multiple encryption, search for the keys by collecting every possible result for enciphering a known plaintext under the first cipher, and deciphering the known ciphertext under the second cipher; then find the match.
Differential Cryptanalysis	Find a statistical correlation between key values and cipher transformations (typically the Exclusive-OR of text pairs), then use sufficient defined plaintext to develop the key.
Linear Cryptanalysis	Find a linear approximation to the keyed S-boxes (see Section 3.4.1.2) in a ciphe, and use that to reveal the key.
Key Schedule	For use in block cipher cryptanalysis. Choose keys which produce known effects in different rounds (see Section 3.4.1.2).
Birthday	Usually a hash attack (see Section 3.4.5.1). Use the birthday paradox, the idea that it is much easier to find two values which match than it is to find a match to some particular value.
Formal Coding	Algebraic attack: from the cipher design, develop equations for the key in terms of known plaintext, and then solve these equations.
Correlation	In a stream cipher, distinguish between data and confusion, or between different confusion streams, from a statistical imbalance in a combiner.
Man-in-the-Middle	We can subvert the routing capabilities of a computer network, and pose as the other side to each of the communicators. Usually used as a key authentication attack on public key systems (see Section 3.4.3.1).

Table 3-2: Attack strategies on cryptographic primitives

It is normal to assume that at least *known-plaintext* is available; often, *defined-plaintext* is assumed. Sometimes the attack strategy is thought to be obvious, given a particular informational constraint and is not further classified. Many attacks try to isolate unknown small components or aspects so they can be solved separately, a process known as divide and conquer and usually used to cryptanalyze the stream cipher's PRBG.

Computationally secure systems are rated by their resistance to three types of attack, *ciphertext-only attack*, *known-plain text attack* and *chosen-plain text attack*. In each type of attack, the attacker is given the encryption and decryption algorithms and with this information attempts to decrypt the message. In the ciphertext only attack, the attacker is also given the encrypted message to examine. Any system failing this attack is considered totally insecure. In a known-plaintext attack, the attacker is given the algorithm, the original message and its encrypted form. Cryptosystems which survive this attack are considered reasonably secure. A chosen plain text attack gives the attacker the algorithm and the ability to encrypt and decrypt any message chosen. Cryptosystems which survive this attack are considered very secure.

3.3.2 ASYMMETRIC VS SYMMETRIC PRIMITIVE SECURITY

Most modern public-key ciphers, whose security relates to some long-studied mathematical problem that is believed to be difficult to solve, i.g., the integer factoring problem or finding discrete logarithms or large integers (see Section 3.4.3), the security of most modern symmetric-key pseudo-random bit generators PRBG (see Section 3.4.2.2.1) do not relate to any widely-studied, hard-to-solve problems. Rather, PRBGs implemented as stream cipher, are designed in an ad-hoc fashion to resist known cryptanalytic attacks. Consequently, the designs, analysis and implementation of reliably secure PRBGs is regarded as exceptionally difficult and is often regarded as more of an art than a science.

Security analysis that constrains the designs of most symmetric key algorithms in the class of block cipher, mainly the key size and data block size, are Brute-force, codebook attack.

3.3.2.1 BRUTE-FORCE OR EXHAUSTIVE KEY SEARCH

In this attack we try to decipher ciphertext under every possible key until readable messages are produced. A way to improve brute force is to try these keys one-by-one form a list of the most-likely keys rather than the search from the key space, also called *dictionary* method. This is the best known attack against symmetric key systems. Nothing else seems to work. Thus, the attack is purely exponential. Thus, for a k-bit symmetric cipher, the expected time to break it is $T(k) = 2^{k-1}$. The space requirements are trivial: a few kilobytes suffice.

The main drawback with this method is the time and cost of attack. A machine using 10^6 chips running at around 10 nanoseconds per cycle was proposed by Diffie and Hellman [20] for finding a DES key (**56-bits long**) [1] in about a day through exhaustive search. Such a machine was generally thought to be too ambitious for the then current technology. Two decades later, Winner, in 1993, designed a $1 million DES cracking machine [21], which would crack a DES, key in 3.5 hours. A late-1997 version of this machine [22] would be capable of finding DES keys in 35 minutes, on average. A $10,000 version of this machine would be capable of finding DES keys in 2.5 days, on average.

3.3.2.2 Codebook Attack

A form of attack in which the opponent simply tries to build or collect a codebook of all the possible transformations between plaintext and ciphertext under a single key. This is the classic approach. The usual ciphertext-only approach depends upon the plaintext having strong statistical biases, which make some values far more probable than others, and also more probable in the context of particular preceding known values. Such attacks can be defeated if the plaintext data are randomised and thus evenly and independently distributed among the possible values. This may have been the motivation for the use of a random confusion sequence in a stream cipher (see Section 3.4.2.2.1)

When a codebook attack is possible on a block cipher, the complexity of the attack is controlled by the size of the block to be encrypted/decrypted (that is, the number of elements in the codebook) and not the strength of the cipher. This means that a codebook attack would be equally effective against either DES or Triple-DES in the latter [23] the block size is the same simply **the input data is, in effect, encrypted three times** using tree different key.

One way to avoid a codebook attack is by having a large block size, which will contain an unsearchable amount of plaintext *uniqueness* or entropy. Another approach is to randomise the plaintext block, often by using an operating mode such as CBC (see Section 3.4.1.3).

3.4 BUILDING CRYPTOGRAPHIC PRIMITIVES WITH THE CFSM MODEL

From hardware point of view (structures), cryptographic primitives can be modelled as a cryptographic finite state machine (CFSM) that could be build in as a chip or part of a chip.

The properties required from the CFSM depend on the type of algorithm for which it is used. In the following section we will show how three different cryptographic primitives can be implemented using a CFSM: block ciphers, stream cipher and cryptographic hash functions.

3.4.1 BLOCK CIPHERS (SUBSTITUTION PERMUTATION CIPHERS)

A block cipher requires the accumulation of data (in a block) before ciphering can begin. Other than simple transposition ciphers, the basic idea is based on ciphers designed to emulate a keyed simple substitution with a table of size far too large to realize. A block cipher operates on a block of data (for example, multiple bytes) in a single ciphering, as opposed to a stream cipher (Section 3.4.2), which operates on bytes or bits as they occur. Block ciphers can be called "*codebook-style*" ciphers.

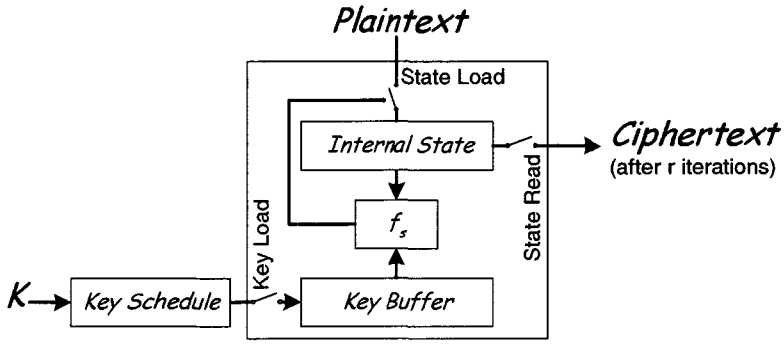


Figure 3-4: Cryptographic finite state machine: implementing of a Block Cipher

3.4.1.1 BLOCK CIPHER PROPERTIES

3.4.1.1.1 DATA DIFFUSION

This is property of an ideal block cipher in which a change of even a single plaintext bit will change every ciphertext bit with probability 0.5. In practice, a good block cipher will approach this ideal called *overall diffusion*. This means that about half of the input bits should change for any possible change to the input block, even for differences of just one bit or that the ciphertext will appear to change at random even between related message blocks. Thus, finding message relationships which might be used to attack the cipher. Overall diffusion is present in a block cipher, but not in a stream cipher. Data diffusion is a simple consequence of the keyed invertible simple substitution nature of the ideal block cipher.

Overall diffusion can be measured statistically in a realized cipher and used to differentiate between better and worse designs. It does not, by itself, define a good cipher, but it is required in a good block cipher.

3.4.1.1.2AVALANCHE EFFECT:

A small change in either the plaintext or the key produce a significant change in the ciphertext. In DES a one bit change in either the key or plaintext produce on the average 35 changed bits in the ciphertext.

3.4.1.1.3COMPLETENESS EFFECT:

Each ciphertext bit is a complex function of all input bits (in a block).

3.4.1.2 BLOCK CIPHERS CONSTRUCTION

Almost all single key block cipher proposals are of the *iterated type*. Generally in block ciphers, non-linear substitutions operations are alternated with mixing functions in order to diffuse and confuse the redundancy. These operations are grouped into a single round transformation and the strength of a cryptographic primitive can be obtained by repeating this simple round transformation a specific number r of iterations using different key each iteration. These keys involved in the rounds are generated using a key schedule unit which based on the master key it update the key for each round.

- Non-linearity essential to every strong cryptographic primitive. The simplest basic non-linear component consist of using lookup tables or S-boxes. Other techniques are based on mathematical structures and logic unit such as exponentiation/inversion over finite field, addition and multiplication, Feistel network, addition and rotation, data dependent rotation and Boolean functions.
- Diffusion in order to spread local changes. Linear transformations are very well suited for this purpose. The simplest solution consist of bit permutation (transposition), rotation, PHT (pseudo-Hadamard transform) and diffusion operations based on MDS (Maximum Distance Separable) linear codes [69]. Some techniques consist of combining linear and non-linear operations in such a way that changes are spread quickly through the block.

A block cipher based CFSM is illustrated in Figure 3-4 where, the state updating function f_s is the round function. The plaintext is loaded into the internal register and converted into the ciphertext by performing r iteration. The state updating function must obviously be invertible for decryption purpose. As will be shown in Chapter 7, the inverse of the round function may have different datapath with complexity comparable to that of the round function.

3.4.1.3 MODE OF OPERATIONS

Block ciphers are used in several operating modes. From the point of view of hardware implementations, these modes can be divided into two major categories:

- Non-feedback modes, such as Electronic Code Book (ECB) mode and counter mode.
- Feedback modes, such as Cipher Block Chaining (CBC) mode, Cipher Feedback (CFB) mode, and Output Feedback (EFB) mode.

In the non-feedback modes, encryption of each subsequent block of data can be performed independently from processing other blocks Figure 3-5. In particular, all blocks can been encrypted in parallel. In the feedback modes, it is not possible to start encrypting the next block of data until encryption of the previous block is completed. As a result, all blocks must be encrypted sequentialy, with no capability for parallel processing.

The limitation imposed by the feedback modes does not concern decryption, which can be performed on several blocks of ciphertext in parallel for both feedback and non-feedback operating modes.

According to current security standards, the encryption of data is performed primarily using feedback modes, such as CBC and CFB. Non-feedback modes, such as ECB, are used primarily to encrypt session keys during key distribution (key establishment, see Section 3.2.4). As a result, using current standards does not permit to fully utilize the performance advantage of the hardware implementations of secret key cryptosystems, based on parallel processing of multiple blocks of data.

The situation could be remedied by including in the block cipher interleaved modes of operation. In these modes, *N* streams of the plaintext blocks, each composed of blocks separated by N positions, are encrypted independently, using classical feedback.

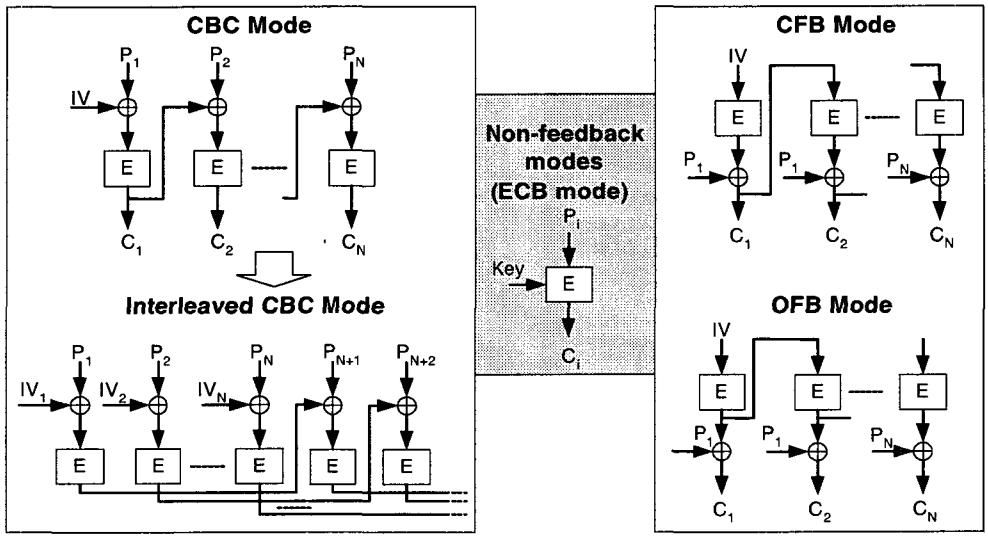


Figure 3-5: Block Cipher operating mode (encryption).

3.4.2 STREAM CIPHERS

This primitive is often used when it is necessary to encrypt large amount of data very quickly. It directly handles messages of arbitrary size, by ciphering individual elements, such as bits or bytes. This avoids the need to accumulate some amount of data or multiple data elements (into a block) for ciphering to complete as is necessary in a conventional block cipher. But note that a stream cipher can be seen as an operating mode, a *streaming* of a tiny block transformation (1-bit cipher feedback mode CFB with r=1) as shown in Figure 3-6 where, E denotes the block cipher.

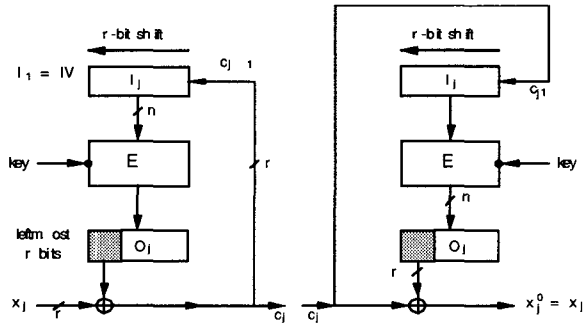


Figure 3-6: Streaming using Block Cipher in CFB mode.

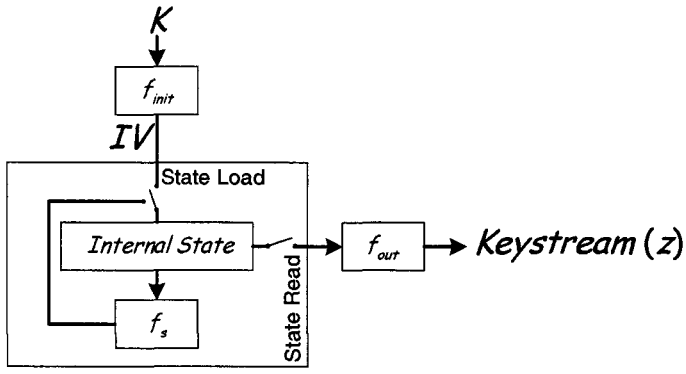


Figure 3-7: Cryptographic finite state machine: implementation of a PRG

3.4.2.1 PROPERTIES

3.4.2.1.1STREAM CIPHER DIFFUSION

In a stream cipher, data diffusion may or may not occur, but if it does, it is necessarily one-way (from earlier to later bit). Since bits are ciphered one-by-one, changing part of the plaintext can affect that part and possibly later parts of the ciphertext; this is called *stream cipher signature*. In a few stream cipher designs, the value of one message byte may change the enciphering of subsequent message bytes; this is called *forward data diffusion*. In contrast, changing even the last bit in a block cipher block will generally change about half of the earlier bits within the same block (overall diffusion of block ciphers). Changing a bit in one block may even affect later blocks if we use a stream meat-cipher composed of block cipher transformations, like CBF in Figure 3-6.

A conventional stream cipher generally does not need data diffusion for strength, as does a block cipher. In a block cipher, it may be possible to separate individual components of the cipher if their separate effects are not hidden by diffusion but a stream cipher generally re-uses the same transformation and has no multiple data components to hide.

3.4.2.2 STREAM CIPHER CONSTRUCTION

The classic stream cipher is simple, consisting of a keyed random bit generator, which produces a random-like *confusion sequence* or *running key* or *key-stream*. The sequence is then combined with plaintext data in a simple additive combiner to produce ciphertext.

The first stream cipher, which marks the start of modern cryptography, is the *Vernam* cipher [24]. It directly combines a stream of plaintext data with a pseudo-random confusion stream using what we know of as mod 2 addition (Boolean logic exclusive-OR). However, the Vernam cipher succumbs to the known-plaintext attack (Section 3.3.1.1). The ultimate stream cipher is the One-Time-Pad (OTP), in which a random source produces sequence that is never reused. Assuming perfect randomness in the source the cipher is mathematically proven to be unbreakable. The confusion sequence in the OTP is the key and it is as long as the data, which is infeasible in practice due to the costs of transfer and storage of the keying material.

3.4.2.2.1 CONFUSION SEQUENCE

The key distribution problem for One-Time Pad suggests that one might use an algorithm to generate the random sequence needed as the key (transfer of only a short seed would then be needed). However, no algorithm using a finite state machine can produce a truly random sequence, since

the finiteness forces the sequence to be periodic. The best we can do is use a *pseudo-random sequences.* The use of pseudo-random bit generators PRBGs is central to stream ciphers, where the PRBG constitutes most of the system. The sequence generator may be the most complex part of the design. Although cryptographic applications obviously impose special requirements, one approach is to use one of the PRBGs, which have been developed for statistical use. The CFSM model can easily be used to build a PRG as shown in Figure 3-7. The system is initialled by loading the Initial State IV into the internal register. The IV is generally a function of the Key and other fixed parameters. Each clock cycle the state is updated and some bits are presented at the output as pseudo-random bits.

There are so many PRBG designs and so many different claims for them. In [27] Ritter gives a survey of pseudo-random sequence generators for cryptographic applications with extensive literature that it is difficult even to compare the claims and no survey can provide complete details of all designs. This includes:

- Chaos, random behaviour based on non-linear dynamic equations [28][29].
- Cellular Automata, a sort of 1-dimensional *Life game* [30]
- "x^2 mod N" generator of Blum, Blum, and Shub [31] it is claimed to be *polynomial-time* unpredictable and cryptographicaly secure.
- PRBGs based on Linear/Non-Linear Feedback Shift Registers LFSR (finite fields Algebra).
- Additive PRBG proposed by Knuth [77] and Marsaglia [32] is a generalization of the LFSR and used normal arithmetic addition (with internal carry operation) instead of GFSR XOR (no carry) operations. Especially efficient software implementations with a long, mathematically proven cycle length.

3.4.2.2.2COMBINER

The combiner is a mechanism which mixes the plaintext and key-stream into a single result. Reversible combiners are used for decryption. The ciphertext is then deciphered into plaintext using a related inverse combiner. The most commonly used combiner is an additive combiner (exclusive-OR). Irreversible or non-invertible combiners are often used to mix multiple RNG's into a single confusion sequence, also can be used to select the running key from the PRBG, case of the output function f_{out} shown in Figure 3-7.

In real stream cipher the re-use of the confusion sequence is extremely dangerous. In general, the most adopted technique is to use a complex PRBG with very long period sequences which, obviously increase the complexity of the PRBG. Another alternative is to use a stronger combiner, such as Latin square [25] or Dynamic Substitution [26] combining. This can drastically reduce the complexity required in the confusion generator, which normally provides all stream cipher strength. These stronger combiners are non-linear, with substantial state. We may elect to use multiple combining in sequence, or a selection among different combiners. Neither of these approaches makes much sense with an additive combiner. Moreover, if a simple additive combiner is used, the transformation becomes weak upon the second character ciphered, or immediately, under known plaintext (Section 3.3.1.1) (because the known plaintext can be subtracted from the ciphertext, thus completely exposing the confusion sequence), making strength dependent on the confusion sequence. More complex transformations imply the need for correspondingly less strong confusion sequences.

3.4.2.3 SECURITY OF STREAM CIPHER

The recovery of the key or CFSM internal state of the PRBG must be computationally infeasible. It follows that the security can be solely based upon the secrecy of the key, or solely upon the

secrecy of the internal state (a mathematical function is used to generate a pseudo random stream of bits based upon a small number of bits that comprise the secret Initial State of the generator). The former property allows for re-initialization in the clear between two users who share the same secret key. Because practical PRBGs always have a finite number of states or finite cycle, this cycle length must be large enough to avoid repetitions in the output sequence. PRBG based on the LFSRs have the advantage that almost all internal states lie on one cycle, hence choosing LFSRs with long periods increases the number of states and solves this problem. This fact may requires a longer key depending on the initialization function f_{init} since the initial state is function of the key.

In the other hand, the influence of the key on the updating function should be as great as possible. This is realized if the updating of a state A with two different keys K_1 and K_2 never gives rise to the same state, more formally $\forall A : f_s(A, K_1) = f_s(A, K_2) \Rightarrow K_1 = K_2$.

3.4.2.4 LFSRS BASED STREAM CIPHERS

The LFSR's sequences are often used in Coding theory including Cryptography and Error Correcting Codes due to their high linear complexity, long period and good statistical properties (see Appendix A and the work by Golomb [34] for a general analysis of LFSRs). There have been a number of proposals in the class of the LFSR-based synchronous stream ciphers. A good overview of the activity in this area is given by Rainer Rueppel in [33]. A cryptographic view is available in Beker and Piper [35], Berlekamp [36].

Rueppel has codified stream ciphers design criteria:

- Long period with no repetitions (Section 3.4.2.3)
- Large linear complexity based on size of equivalent LFSR (Section A.1.3)
- Statistically random
- Confusion (output bits depend on all key bits)
- Diffusion
- Use of highly non-linear Boolean functions

Because of the weakness of the data-confusion combiner (exclusive-OR) and the simplicity of the LFSR (Berklamp-Massey algorithm [37] will recover the unknown state of a simple n-bit LFSR and its primitive polynomial, with just $2n$ known bits), the need to generate a more "complex"sequence is a necessity. This led to the idea of using multiple LFSR's and somehow mixing them so that the ultimate complexity was the product of the individual complexities. A number of proposals exist and some successful stream ciphers do use LFSRs, employing certain useful techniques and some degree of non-linear elements to overcome the existing cryptanalytic attacks and produce a highly confusing sequences. Some practical proposals include:

- Stuttering involves clocking the register a variable and unpredictable number of times between outputs (only select some of the output bits) [38].
- Clocking a set of LFSRs at different rate and combining their output [39]
- Using LFSRs that control their own clock [40].
- Using multiple shift registers and combining outputs from them in non-linear fashion (multiplexing algorithm of Jennings [41], see Appendix A).
- Controlling the clocking of some LFSRs by another, the case of the alternative stop-and-go generator [42].

Although some schemes falls to some attacks revealing a key redundancy or recovering the internal state of the generator, attacks on other schemes does not substantially weaken the generators. In Figure 3-8 are reported some useful LFSR based stream cipher employing certain techniques enumerated above and presenting a good cryptographic properties. The description of

each cipher is reported in Table 3-3; the period and linear complexity of such primitives are depicted in Table 3-4.

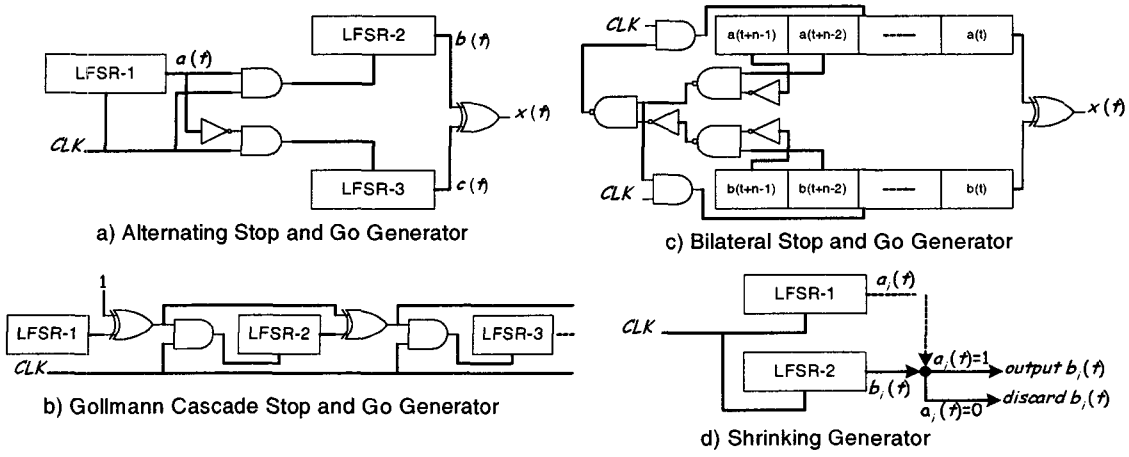


Figure 3-8: Examples of LFSR based Stream Cipher

Stream Cipher	Description
a)	This generator uses three LFSRs of different length L_1,L_2,L_3. LFSR-2 is clocked when the output of LFSR-1 is 1; LFSR-3 is clocked when the output of LFSR-1 is 0. The output of the generator is the XOR of LFSR-2 and LFSR-3. This generator has a long period and large linear complexity see Table 3-4. A correlation attack (see Section 3.4.2) against LFSR-1 have been found, but it does not substantially weaken the generator. There have been other attempts at keystream generators along these lines.
b)	The Gollmann cascade is a strengthened version of a stop-and-go generator. It consists of a series of LFSRs, with the clock of each controlled by the previous LFSR. If the output of LFSR-1 is 1 at time t - 1, then LFSR-2 clocks. If the output of LFSR-2 is 1 at time t - 1, then LFSR-3 clocks, and so on. The output of the final LFSR is the output of the generator. If all the LFSRs have the same length, n, the period and linear complexity of a system with k LFSRs is reported in Table 3-4.
c)	This generator uses two LFSRs, both of length n. The output of the generator is the XOR of the outputs of each LFSR. If the output of LFSR-2 at time $t-1$ is 0 and the output at time $t-2$ is 1, then LFSR-2 does not clock at time t. Conversely, if the output of LFSR-1 at time $t-1$ is 0 and the output at $t-2$ is 1, and if LFSR-1 clocked at time t, then LFSR-2 does not clock at time t. No evident key redundancy has been observed in this system
d)	The shrinking generator uses a different form of clock control than the previous generators. Take two LFSRs: LFSR-1 and LFSR-2 of length L_1,L_2. Clock both of them. If the output of LFSR-1 is 1, then the output of the generator is LFSR-2. If the output of LFSR-1 is 0, discard the two bits, clock both LFSRs, and try again. This idea is simple, reasonably efficient, and looks secure. If the feedback polynomials are sparse, the generator is vulnerable, but no other problems have been found. One implementation problem is that the output rate is not regular; if LFSR-1 has a long string of zeros then the generator outputs nothing. A Buffering technique can be used to solve this problem.

Table 3-3: Description of LFSR based Stream Cipher.

Stream Cipher	Period (T)	Linear Complexity (L)
a)	$2^{L_1}(2^{L_2}-1)(2^{L_3}-1)$	$(L_2+L_3)2^{L_1-1} \le L \le (L_2+L_3)2^{L_1-1}$
b)	$(2^n\text{-}1)^k$	$n(2^n\text{-}1)^{k-1}$
c)		
d)	$2^{L_1-1}(2^{L_2}-1)$	$L_2 2^{L_1-2} \le L \le L_2 2^{L_1-1}$

Table 3-4: PRPG's Period and Linear Complexity.

3.4.3 PUBLIC KEY CIPHERS

Public key ciphers are generally block ciphers, with the unusual property that one key is used to encipher and a different, apparently unrelated key is used to decipher a message. So if we keep one of the keys secret or *private*, we can release the other key (the *public* key) and anyone can use that to encipher a message to us. Then we use our private key to decipher any such messages. It is interesting that someone who enciphers a message to us cannot decipher their own message even if they want to.

Public-key cryptographic systems have proven to be effective and more manageable than symmetric key systems in a large number of scenarios. Since the invention of public-key cryptography in 1976 by Whitfield Diffie and Martin Hellman [85], numerous public-key cryptographic systems have been proposed. All of these systems based their security on the difficulty of solving a mathematical problem. Over the years, many of the proposed public-key cryptographic systems have been broken and many others have been demonstrated to be impractical. Today, only three types of systems are considered both secure and efficient. Examples of such systems and the mathematical problems on which their security is based, are:

a.) Integer factorization problem (IFP): RSA and Rabin-Williams.
b.) Discrete logarithm problem (DLP): U.S. government's Digital Signature Algorithm (DSA), Diffie-Hellman key agreement scheme, ElGamal encryption and signature schemes, Schnorr signature scheme, and Nyberg-Rueppel signature scheme.
c.) Elliptic curve discrete logarithm problem (ECDLP): the elliptic curve analog of the DSA (ECDSA) and the elliptic curve analogs of the Diffie-Hellman key agreement scheme, the ElGamal encryption and signature schemes, the Schnorr signature scheme and the Nyberg-Rueppel signature scheme.

Each of these systems is capable of providing confidentiality, authentication, data integrity and non-repudiation. It must be emphasized that none of these problems have been proven to be intractable (i.e.,difficult to solve in an efficient manner). Rather, they are believed to be intractable because years of intensive study by leading mathematicians and computer scientists has failed to yield efficient algorithms for solving them.

Because public key ciphers operate on huge values, they are very slow and so are normally used just to encipher hash values for digital signature (see Section 3.2.3) or a random message key for key establishment protocol see Section 3.2.4. The message key is then used by a conventional secret key cipher, which actually enciphers the data.

3.4.3.1 CERTIFICATION

At first glance, public key ciphers apparently solve the key distribution problem. But in fact they also open up the new possibility of a man-in-the-middle attack. To avoid this, it is necessary to assure that one is using exactly the correct key for the desired user. This requires authentication (validation or certification) via some sort of secure channel, and that can take as much effort as a secure secret key exchange. A man-in-the-middle attack (see Section 3.3.1.1) is extremely worrisome, because it does not involve breaking any cipher, which means that all the effort spent in cipher design, analysis, mathematical proofs and public review would be completely irrelevant.

3.4.3.2 PUBLIC KEY CRYPTOSYSTEMS PROPERTIES

3.4.3.2.1EXPONENTIAL VERSUS POLYNOMIAL TIME ALGORITHMS

Above all else, the difficulty of a problem must be defined. What does it mean for a mathematical problem to be difficult? A mathematical problem is difficult if the fastest algorithm to solve the problem takes a long time relative to the input size.

To analyse how long an algorithm takes, computer scientists introduced the concept of polynomial time algorithms and exponential time algorithms. Roughly speaking, an algorithm runs quickly relative to the size of its input if it is a polynomial time algorithm and slowly if it is an exponential time algorithm. Therefore, easy problems equate with polynomial time algorithms and difficult problems equate with exponential time algorithms.

It is important to notice the words *relative to the input size* in the definition of polynomial time and exponential time algorithms. All problems are straightforward to solve if the input size is very small, but we are interested in how much harder a problem gets as the size of the input grows. For example, adding two numbers is straightforward, as is factoring two numbers. However, addition is an example of an easy problem, because there is an algorithm to add numbers, which runs in polynomial time, meaning that it would not take very long to add two enormous numbers. On the other hand, factoring is a hard problem because, in general, factoring a large number takes a very long time. Thus, when looking for a mathematical problem on which to base a public-key cryptographic system, cryptographers are searching for a problem for which the fastest algorithm takes exponential time. In broad terms, the longer it takes to compute the best algorithm for a problem, the more secure a public-key cryptosystem based on that problem will be.

The most upsetting long-term threat to DL cryptosystems that we can foresee right now comes from quantum computers. Peter Shor in [43] showed that if such machines could be built, integer factorization and discrete logs (including elliptic curve discrete logs) could be computed in polynomial time. This result has stimulated an explosion in research on quantum computers. While there is still some debate on whether quantum computers are feasible, no fundamental obstructions to their constructions have been found and novel approaches are regularly suggested. The one comforting factor is that all experts agree that even if quantum computers are eventually built, it will take many years to do so (at least for machines on a scale that will threaten modern public key systems) and so there will be advance warning about the need to develop and deploy alternate systems.

In the other hand, because lower complexity bounds are hard to achieve, it is possible in DLP for example that exponential attacks can be replaced by subexponential attacks [44] and subexponential attacks to be improved by polynomial time attacks. These algorithmic attacks, however, are becoming increasingly complicated and it's not clear whether their smaller (expected) running times can be effectively attained in large field implementation. The DLP, thus, will serve as a reliable source for secure protocols as long as the cryptographers are not running out of appropriate groups where no subexponential time algorithm is known. For achieving provable security, lower bounds in solving the corresponding problems are needed but these are either too difficult to establish or restricted to a model not compliant with reality.

3.4.3.2.2DLP VERSUS IFP

RSA the first usable public key cryptosystem, introduced in [50] requires calculation of a private key derived from the least common multiple of two large prime numbers. This particular cryptosystem is based on the difficulty of factoring very large numbers and today (determining the two prime numbers), it is still the most widely used public-key cryptosystem in the world. Since then, in the field of computational number theory, major work has been done towards efficient integer fac-

torization. As a consequence, new types of public-key algorithms have arisen. The most important competitors to RSA are schemes based on the Discrete Logarithm (DL) problem. Originally, the DL problem was considered in the multiplicative group of a finite field, especially a prime field or a field of characteristic 2, since the later is most appropriate for implementations.

The main concern is whether DL based cryptographic schemes are indeed more secure than the IF based schemes. Although the mathematical principle of IF and DL schemes is quite different, the algorithmic attacks of these schemes are though related. Bruce Schneier in [69] states that computing discrete logarithms is closely related to factoring. If we know how to calculate efficiently discrete logs, then we can factor (that is the basis of Shor's quantum factoring algorithm [43]). Further, one note the interesting statement in [48], while there are algorithms for factorization that do not generalize to give discrete logarithm algorithms (the *Schnorr-Lenstra algorithm*, for example, see Appendix A), the converse is not the case. In other words, all currently known algorithms for solving the DLP can be applied to the IFP, whereas the reverse is not always the case. It turn out that discrete logarithms are at least as hard as factoring and likely to remain so.

On the other hand, breaking such systems is quite another matter, here we fall into the statement of handling an exponential time algorithm, which requires a sieving process to handle the large amount of involved calculations. At this point, number of approaches have been undertaken. The most promising one, which may have a major impact, is based on the massive distributed computing over the Internet since, there is huge computing power available in the idle time of the computers on the Internet and that power can be harnessed easily. Given the rapid growth in such computing power and the relatively slow increase in the running time of the number field sieve with the size of the modulus in DLP, cryptosystem should be build in generous safety margins (larger field sizes) to take into account the current state of the art and compensate for expected growth in computing power as well as improvements in algorithms.

3.4.3.3 DLP IN $GF(p)$ AND $GF(2^m)$

There are two general types of Galois Fields with cryptographic significance, $GF(p)$ with p prime, and *the extension field* $GF(2^m)$ where the field size m is large. It is unfortunate that these two systems, though related, are both often discussed in the same breath, since theory in one field isn't necessarily applicable in the other field. In this work special attention is devoted to the extension field $GF(2^m)$ basically because it is easier to implement in hardware. The DLP is not equally difficult for instances which use $GF(2^m)$ as those which use $GF(p)$, where the sizes 2^m and p of the fields are approximately equal. Until the last decade, there have not been any mathematical discoveries, which state that the DLP over $GF(2^m)$ is easier or harder than the DLP over $GF(p)$. Unfortunately, recent discoveries [49] show that DLP in the field $GF(2^m)$ are much easier to compute. Although the fields $GF(p)$ with p prime appears to offer relatively high level of security, this does not restrict the use of the corresponding extension field in cryptographic applications. In fact, through survey and analysis of known algorithms for solving DLP [45][46][47], the field size m for which $GF(2^m)$ is used in a cryptosystem, has to be carefully chosen such that 2^m-1 is a large prime called *Mersenne* prime. A good overview of the scientific activity in this area and main results and conclusions are given by *Odlyzko in* [48].

3.4.3.4 DLP BASED CRYPTOSYSTEMS CONSTRUCTION

In what follows a description of our DLP based scheme, mainly exponentiation over $GF(2^m)$ is given.

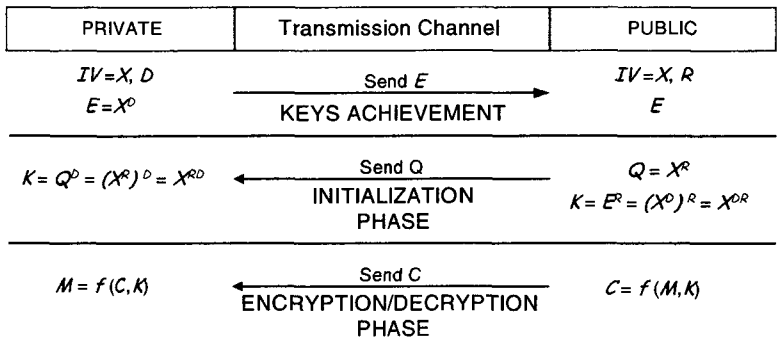


Figure 3-9: Public and Private Operation Scheme of DLP based Cryptosystem.

Choosing some (arbitrary, non-zero) bit patterns, X defined as an initial vector IV and D the Decryption Key, also called the Private Key. It is not transmitted to anyone else, but is needed each time we want to decrypt an encrypted message, so it must be stored (or at least we must remember how to regenerate it, for example from a memorized *passphrase*).

Computing $E = X^D$. E is the Encryption Key, which can be published to anyone who wants to send us an encrypted message; it is also called Public Key.

Choose a (true) random bit pattern R. R is the Random Key and is never stored, transmitted or reused. Compute $Q = X^R$. Q is the Open Key, and can be transmitted in-the-clear as part of a header that precedes an encrypted message or published to the private party.

- *Encryption:*

Compute $K = E^R$. K is the Closed Key or Secret Key. Use K to initialize the encryption engine's "mixture generator" and then use the mixture generator to operate upon the information to be encrypted (combining phase).

- *Decryption:*

Compute the secret key from the Open Key and the private key $K = Q^D$, that will be identical to the K value computed above during encryption. Use K to initialize the encryption engine's "mixture generator" and then use the mixture generator to operate upon the encrypted information (decombining phase). Because the mixture generator is initialized to the identical state used to start encryption, it is able to undo the encryption processing and recreate the original message.

All operations are performed modulo field generating polynomial of $GF(2^m)$.

3.4.4 CRYPTOGRAPHIC HASH FUNCTIONS (CHF)

Hash functions compress strings of arbitrary lengths to strings of fixed lengths (typically 128 bits or 160 bits for modern cryptography). There have been many proposals on how to design and analysis this cryptographic primitive. One can refer to the doctoral dissertation of Bart Preneel [51]for a huge treatment of this field. A survey on cryptographic hash functions can be found in [52].

All cryptographic hash functions proposals are build by means of chaining transformations as a state updating function of a finite state machine. The message M (input) is padded and fed block by block (m_i for $1 \le i \le n$) to a process governed by these chaining transformations that transform the chaining state parametrized by the message block into another state. The initial state IV is spec-

ified (fixed) and the final chaining state h_n is used to determine the hash result. This traditional CHF is shown in Figure 3-10, where f_h is the transformation (compression) function.

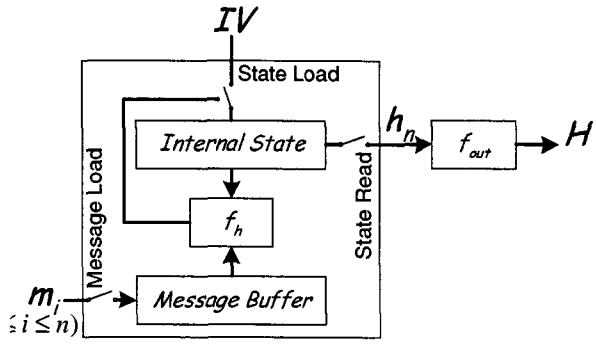


Figure 3-10: Cryptographic finite state machine: implementation of a CHF

In many proposals the compression function is realized by combinations of a block cipher and some simple operations. Others are CHF based on cellular automaton [53][54] and Algebraic Matrices [55]. Besides these there are many dedicated designs. The best known are MD5 [56], SHA [57] and RIPEMD [58].

3.4.4.1 SECURITY PROPERTIES

Hash functions must satisfy the following properties:

- Preimage resistance: it should be hard to find a preimage for a given hash result. In other terms it should be computationally infeasible to find any input which hashes to any pre-specified output.
- Second preimage resistance: it should be hard to find a second preimage for a given input.
- Collison resistance (collison free): it should be hard to find two different inputs with the same hash result.

While these properties are simple, experience has shown that achieving them is quite hard. In [61], Ivan Dämgard presented in computational complexity theoretic framework a method to construct a collision resistant hash function by using as chaining transformation a collision resistant compression function f_h with a fixed-length input m_i as shown in Figure 3-10, where the hash result H can be expressed as:

$$h_i = f_h(h_{i-1}, m_i) \quad 0 \leq i \leq t \tag{3-4}$$

where, the arbitrary length message M is divided into t blocks $m_1, m_2, \ldots, m_t$ and each of which is processed in one round, h_i is the *chaining variable* between iteration i-1 and iteration i (with bit length $n, n \geq m$) on the fixed b-bits message block m_i, $h_0 = IV$ is the initial state vector. The output hash value is $H(M) = f_{out}(h_t)$ where f_{out} is the output function. As example in the CBC-CHF, as specified in [59], the output transformation function f_{out} consists of selecting the leftmost m bits of h_t. Another example in [60] consists of selecting a multiple w of 16 of lefmost bits of h_t.

From Equation 3-4, it turns out that generating a collision for such a CHF involves either generating a collision for f_h or solving a problem with comparable complexity. Consequently, the component that is crucial for both performance and resistance against CHF cryptanalysis (which

guaranty these properties) is the chaining transformation f_h . Dämgard in[61] gave three examples of collision free functions to use in the chaining transformation. However, for two of these examples collisions have been found in [62] [63]. Moreover, in [54], Joan Daemen showed that collisions have been found for the third Dämgard's compression function. The weak point lies in the uncertainty whether generating a collision is in fact a hard problem. Application of the Dämgard's principle imposes important restrictions.

In his doctoral thesis dissertation [64], Joan Daemen proposes a generic model, which consists of the compression function f_h with a fixed-length input based on an invertible chaining transformation collisions. He showed that the Dämgard design principle can only be applied if the chaining transformation is not invertible.

3.4.5 KEYED HASH FUNCTIONS (KHF)

In KHF a hash function take as a distinct secondary input a secret key. Such hash functions, commonly known as Message Authentication Codes (MACs), have received widespread use in practice for data integrity and data origin authentication, e.g. in banking applications.

These are conventional cryptographic algorithms, which allow a receiver to establish the source of integrity of data received. Using KHFs, it is possible to produce a fixed length digital signature that depends on the whole message and ensures authenticity of the message. To produce digital signature for a message M, the digest of M, given by $H(M)$, is calculated and then encrypted with the secret key of the sender. Encryption may be performed either by using a public key or a private key algorithm. Encryption of the digest prevents active intruders from modifying the message when using only a hash value. Since active spoofer may intercept the transmitted message, modify it as he wishes and resend it appended with the digest recalculated for the modified message.

3.4.5.1 REQUIREMENTS FOR KEYED HASH FUNCTIONS

We consider the following properties for keyed hash functions:

3.4.5.1.1SECURITY REQUIREMENTS

- It should use a secret key of at least 128 bits to prevent exhaustive key search.
- It should produce a message digest with at least 128 bits to thwart birthday attacks (Section 3.4.2).
- It should uniformly distribute the message digest in the message digest space. This thwarts statistical attacks.
- It should require $O(2^m)$ known text-MAC pairs to find a second preimage.

3.4.5.1.2DESIGN HEURISTICS

- It should use the secret key many times in the hashing process (especially at the beginning and the end).
- It should use every bit of the message several times in the hashing process (redundancy of the message).
- The underlying round function should be analysed very carefully to thwart the attacks that are based on the weaknesses of the round function, such as fixed-point attack. It needs spe-

cial care when the design is based on an existing algorithm, such as an encryption algorithm or a hash function.

3.4.5.2 CONSTRUCTION OF KEYED HASH FUNCTIONS

KHFs can be constructed from scratch, from an encryption algorithm (block or stream cipher) as they already have a secret key and from an existing hash function. KHF are preferred to be used in authentication schemes compared with the schemes based on encryption algorithms, since an encryption algorithm might not satisfy the security requirements of keyed hash functions. In this section, we show how a pre-existing hash function can be used to construct a keyed hash function. This implies that, some security requirements of the designed keyed hash function rely on the security of the underlying hash function.

There are several way on how a pre-existing hash function can be used to construct a KHF (the way for key injection). The secret key can be introduced in the IV, in the compression function f_h, and in the output function f_{out}. There exist mainly three proposals: the secret prefix, secret suffix and envelope method.

3.4.5.2.1THE SECRET PREFIX METHOD

The secret prefix method consists of prepending a secret key K_1 to the message m_i before each iteration of the hashing operation: $h_i = f_h(h_{i-1}, K_1 \| m_i)$ $0 \le i \le t$, where $\|$ denotes concatenation. The keyed hash value is hash $KHF(M) = f_{out}(h_t)$. This method was suggested for some hash functions, e.g. MD5 under the name MD2.5 and has been pointed out in several papers that this MAC is insecure: a single text-MAC pair contains information essentially equivalent to the secret key, independent of its size since an attacker may append any blocks to the message and update the KHF.

3.4.5.2.2THE SECRET SUFFIX METHOD

A second proposal is to append a secret key K_2 to the message m_i before each iteration of the hashing function: $h_i = f_h(h_{i-1}, m_i \| K_2)$ $0 \le i \le t$. The keyed hash value is hash $KHF(M) = f_{out}(h_t)$. This method is weak if a second preimage attack on the underlaying hash function is feasible [65] using a forgery attack [67]. Further, it should be noted that an attacker can remove the feedforward in the last iteration in Equation 3-4, since the chaining variable entering this iteration can be computed using m_i (message sub-block) only.

3.4.5.2.3THE ENVELOPE METHOD

The envelope method combines the prefix and suffix methods. One prepends a secret key K_1 and appends a secret key K_2 to the message input m_i before the hashing function: $h_i = f_h(h_{i-1}, K_1 \| m_i \| K_2)$ $0 \le i \le t$. The keyed hash value is hash $KHF(M) = f_{out}(h_t)$. The main concern with this method is that an Exhaustive Key Search attack [52] can be used to determine K_1 if internal collisions can be found for the chaining variable. After this, the envelope method is effectively reduced to the secret suffix method. Further, choosing $K_1 \neq K_2$ much additional security [65].

3.4.5.2.4NEW CONSTRUCTION

The weakness of the three existing proposals necessitate extreme care to be exercised in constructing a MAC from a hash function since, unkeyed hash functions are not typically designed for use as MACs. Preneel and Oorschot proposed in [65] new constructions, with the following goals:

- The secret key should be involved at the beginning, at the end and in every, and in every iteration of the hash function.

- The deviation from the original hash function should be minimal (to minimize implementation effort and maximize on confidence previously gained).
- The performance should be close to that of the hash function.
- The approach should be generic, i.e. should apply to any hash function based on the same principles.

These guidelines have been considered in the design of the SSMG's KHF reported in chapter 5.

CHAPTER 4

4 FINITE FIELD ARITHMETIC FOR CRYPTOGRAPHY

In this chapter Galois finite fields and finite field arithmetic operators are introduced. Definitions and main results underlying finite field, $GF(2^m)$ theory and field-generating polynomial $p(x)$ are presented. Further, $GF(2^m)$ arithmetic architectures carrying out frequently used operations in cryptography are discussed, mainly, multiplication and exponentiation over large prime field. In addition, new circuits are presented which are low-energy, highly regular and programmable with respect to $p(x)$. We demonstrate digit-serial (or partially parallel) architectures and rearranging the gate topology from array-type to tree-type helps to achieve up to 90% saving of energy delay product compared to their equivalents bit-serial.

4.1 FINITE FIELD ARCHITECTURES FOR CRYPTOGRAPHY

Finite field arithmetic architectures are the basic building blocks in many applications involving cryptography. Many popular public-key algorithms including ElGamel encryption and signature schemes [84], Diffie-Hellman key-distribution Scheme [85] and other variant of discrete log based cryptosystems [86] relies on exponentiation in large finite field, either over integers modulo a prime odd p (implementation over $GF(p)$), or a polynomial field (implementation over $GF(2^m)$). $GF(2^m)$ was the preferred implementation, basically because it is easier to implement in hardware [69], [75].

$GF(2^m)$ exponentiation is a difficult task to carry out efficiently in software for large finite field since it is a time consuming operation.For physical security and performance reasons it is often advantageous to implement this operation in hardware.

Exponentiation operation can be computed by repeated square-and-multiply operations (S&M algorithm) [77] as a series of modular multiplications over $GF(2^m)$. Therefore, multiplication in $GF(2^m)$ is usually considered the crucial operation which, determines the speed or throughput of a DL based cryptosystem.

The complexity of such large prime field multipliers is related to the field-size and field-generating polynomial. Also, the long arithmetic operators exhibit in general a great activity and dissipate consequent shares of the power supply. Thus, the design of efficient dedicated, low energy, finite field multipliers improves the overall system performance of $GF(2^m)$ exponentiator.

The usual approach to reduce time complexity and improve the performance is to use parallel multipliers. However, the hardware complexity of a bit-parallel multiplier is proportional to m^2. Its area and energy consumption increase dramatically as the field order m increase since. So far, normal base (Section 4.2.6) and polynomial base (Section 4.2.4) representations have been to reduce the complexity. Optimal normal base representations can be of special interest in this context because of their moderate complexity. However, base cannot be constructed for any field [97].

Digit-serial technique an alternative to the bit-parallel approach, process multiple bits *digit* of an entire word, referred to as the *digit-size*, in one clock-cycle. This technique is suitable for the implementation of moderate sample rate systems where, the area and power consumption are critical. It was first used to implement a Galois Field multiplier in [82]. However, the architecture of the multiplier is based on semi-systolic 2-D array multiplier architecture [78], in which a large amount of gates and registers have to be mapped yielding to increase both area and power consumption when large field-size is used. In this chapter a new digit-serial, high performance $GF(2^m)$ multipliers are developed and implemented. The proposed architectures are mapped on low-power/low-voltage technology. A technique such as gating the clock is used to analyse the amount of power savings using different digit-size.

In the first part under Section 4.4, we present two low-energy, highly regular architectures performing a large prime $GF(2^m)$ multiplication. The first one is area-efficient digit-serial architecture, when field-generating polynomial $p(x)$ is a trinomial. The second architecture is digit-serial Linear Systolic Array (LSA) and programmable on $p(x)$. The parallel algorithm inside of each digit cell reduces both the global cycle time for the first architecture and the switching activity in the second one. An analysis of the performance comparison is described as function of the *digit-size*. A comparison is made with the bit serial architecture based on the performance improvement with respect to computation delay and energy consumption of one multiplication operation over $GF(2^{607})$. Thus, the factor of merit for performance measurement is defined as the product of energy times the delay. Gate level simulation shows that the energy delay products are greatly reduced for both architectures.

In Section 4.5, a digit-level pipelined, linear systolic array exponentiator, implementing the S&M algorithm. is derived from the LSA digit-serial multiplier. The architecture is regular, expendable to any field order and programmable on $p(x)$. It allows the input elements to enter a linear systolic array in the same order and the system only requires one pipelined control signal. The squaring and multiplication operations are overlapped at a high system frequency in order to reduce the total delay of the computation. An analysis of the performance comparison is described as function of the digit-size. The energy-delay product computed on gate level implementations shows an energy efficient circuit at the expanse of increased area.

4.2 FINITE FIELD FUNDAMENTALS

4.2.1 FINITE FIELDS

A field is essentially a set of elements in which it is possible to add, subtract, multiply and divide field elements and always obtain another element within the set. A finite field is a field containing a finite number of elements.

4.2.2 GALOIS FIELDS

It can be shown that the set of integers $\{0, 1, 2,..., p\text{-}1\}$ where p is a prime, together with modulo p addition and multiplication forms a field [71]. Such a field is called the finite field of order p, or $GF(p)$, in honour of Evariste Galois [72]. In this thesis only binary arithmetic is considered, where p is constrained to equal 2. This is because, as shall be seen, by starting with $GF(2)$, the representation of finite field elements maps conveniently into the digital domain. Arithmetic in $GF(2)$ is therefore defined modulo 2. It is from the base field $GF(2)$ that the extension field $GF(2^m)$ is generated.

4.2.3 THE EXTENSION FIELD $GF(2^m)$

Before introducing $GF(2^m)$, some definitions are required.

Definition 1. A polynomial $p(x)$ of degree m over $GF(2)$ is a polynomial of the form:

$$p(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1 x + p_0 \tag{4-5}$$

where the coefficients p_i are elements of $GF(2) = \{0,1\}$. Polynomials over $GF(2)$ can be added, subtracted, multiplied and divided in the usual way [70]. A useful property of polynomials over $GF(2)$ is that ([70], pp.29)

$$p^2(x) = (p_0 + p_1 x + p_2 x^2 + \ldots + p_m x^m)^2 = p_0 + p_1 x^2 + p_2 x^4 + \ldots + p_m x^{2m} = p(x^2) \tag{4-6}$$

Also, if $p(x)$ is an irreducible polynomial (see Definition 2) and the primitive element (see Definition 5) α is a root of $p(x)$ then,

$$\alpha^m = \sum_{i=0}^{m-1} p_i \alpha^i \tag{4-7}$$

The notion of an *irreducible* polynomial is now introduced.

Definition 2. A polynomial $p(x)$ over $GF(2)$ of degree m is irreducible if $p(x)$ is not divisible by any polynomial over $GF(2)$ of degree less than m and greater than zero.

To generate the extension field $GF(2^m)$, an irreducible, monic polynomial of degree m over $GF(2)$ is chosen, $p(x)$ say. Then the set of 2^m polynomials of degree less than m over $GF(2)$ is formed and denoted F. It can then be proven that when addition and multiplication of these polynomials is taken modulo $p(x)$, the set F forms a field of 2^m elements, denoted $GF(2^m)$ [71]. Note that $GF(2^m)$ is extended from $GF(2)$ in an analogous way that the complex numbers **C** are formed from the real numbers **R** where in this case, $p(x) = x^2 + 1$.

To represent these 2^m field elements, the important concept of a *basis* is now introduced.

4.2.4 THE POLYNOMIAL BASIS AND PRIMITIVE ELEMENTS

Definition 3. A set of m linearly independent elements $\beta = \{\beta_1, \beta_2, \ldots, \beta_{m-1}\}$ of $GF(2^m)$ is called a basis for $GF(2^m)$.

A basis for $GF(2^m)$ is important because any element $a \in GF(2^m)$ can be represented uniquely as the weighted sum of these basis elements over $GF(2)$. That is

$$a = a_0\beta_0 + a_1\beta_1 + \ldots + a_{m-1}\beta_{m-1} \text{ where } a_i \in GF(2) \tag{4-8}$$

Hence the field element a can be denoted by the vector $(a_0, a_1, \ldots, a_{m-1})$. This is why the restriction $p = 2$ has been made, since the above representation maps immediately into the binary field.

There are a large number of possible bases for any $GF(2^m)$ [71]. One of the more important bases is now introduced.

Definition 4. Let $p(x)$ be the defining irreducible polynomial for $GF(2^m)$. Take α as a root of $p(x)$, then $\{1, \alpha, \ldots, \alpha^{m-1}\}$ is the polynomial basis for $GF(2^m)$.

For example consider $GF(2^4)$ with $p(x) = x^4 + x + 1$. Take α as a root of $p(x)$ then $\{1, \alpha, \alpha^2, \alpha^3\}$ forms the polynomial basis for this field and all 16 elements can be represented as

$$a = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3$$

where the $a_i \in GF(2)$. These basis coefficients can be stored in a basis table of the kind shown in Appendix B.1.

Definition 5. An irreducible polynomial of degree m is a primitive polynomial if the smallest positive integer n for which $p(x)$ divides $x^m + 1$ is $n = 2^m - 1$.

If α is a root of $p(x)$ where this polynomial is not only irreducible but also primitive, then $GF(2^m)$ can be represented alternatively by the set of elements $GF(2^m) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$, $(n = 2^m - 1)$. In this case α is called a primitive element and $\alpha^n = 1$. The relationship between powers of primitive elements and the polynomial basis representation of $GF(2^4)$ is shown in Appendix B.1.

The choice as to whether to represent field elements over a basis or as powers of a primitive element usually depends on whether a hardware or a software implementation is being adopted. This is because $\alpha^i\alpha^j = \alpha^{i+j}$, where this indices addition is modulo 2^m-1 and so can easily be carried out on a general purpose computer. Multiplication of field elements using the primitive element representation is therefore simple to implement in software, but addition is much more difficult. For implementation in hardware however a basis representation of field elements makes addition relatively straight forward to implement. This is because

$$A = B + C = (b_0 + c_0) + (b_1 + c_1)x + \ldots + (b_{m-1} + c_{m-1})\,x^{m-1}$$

and so addition is performed component-wise modulo 2. Hence, a $GF(2^m)$ adder circuit comprises 1 or m XOR gates depending on whether the basis coefficients are represented in series or parallel. This is an important feature of $GF(2^m)$ and one of the main reasons why finite fields of this form are so extensively used.

4.2.5 THE DUAL BASIS

The dual basis is an important concept in finite field theory and was originally exploited to allow for the design of hardware efficient RS encoders [73]. Moreover, subsequent research has allowed the use of dual basis multipliers to be adopted throughout the encoding and decoding processes.

Definition 6. Let $\{\lambda_i\}$ and $\{\mu_i\}$ be bases for $GF(2^m)$, let f be a linear function from $GF(2^m) \rightarrow GF(2)$, and $\beta \in GF(2^m)$, $\beta \neq 0$. Then $\{\lambda_i\}$ and $\{\mu_i\}$ are dual to each other with respect to f and β if

$$f(\beta\lambda_i\mu_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{4-9}$$

In this case, $\{\lambda_i\}$ is the standard basis and $\{\mu_i\}$ is the dual basis.

Theorem 1. Every basis has a dual basis with respect to any non-zero linear function

f: $GF(2^m) \rightarrow GF(2)$, and any non-zero $\beta \in GF(2^m)$.

For example consider $GF(2^4)$ with $p(x) = x^4 + x + 1$ and take α as a root of $p(x)$. Then $\{1, \alpha, \alpha^2, \alpha^3\}$ is the polynomial basis for the field. Now taking $\beta = 1$ and f to be the least significant polynomial basis coefficient, $\{1, \alpha^3, \alpha^2, \alpha\}$ forms the dual basis to the polynomial basis. In fact by varying β there are 2^m-1 dual bases to any given basis and the dual basis with the most attractive characteristics can be taken. This is usually taken to mean the dual basis that can be obtained from the polynomial basis with the simplest linear transformation.

4.2.6 NORMAL BASIS

A normal basis for $GF(2^m)$ is a basis of the form $\{\beta, \beta^2, ..., \beta^{2^{m-1}}\}$ where $\beta \in GF(2^m)$. For every finite field there always exists at least one normal basis [74]. Normal basis representations of field elements are especially attractive in situations where squaring is required, since if $(a_0, a_1,...,a_{m-1})$ is the normal basis representation of $a \in GF(2^m)$ then $(a_{m-1}, a_0, a_1,...,a_{m-2})$ is the normal basis representation of $a^2 \bmod p(x)$. This property is important in its own right but also because it allows for hardware efficient Massey-Omura multipliers to be designed [94]. The normal basis representation of $GF(2^4)$ is given in Appendix B.1.

4.2.7 EXPONENTIATION OVER $GF(2^m)$: S&M ALGORITHM

Unlike addition and multiplication, exponentiation in $GF(2^m)$ is similar to the operation for ordinary integers. Given an arbitrary element $A = \sum_{i=0}^{m-1} a_i x^i \in GF(2^m)$, and Y a non-zero element in $GF(2^m)$ the exponentiation function is defined as

$$Y = A^E, 0 \le E \le 2^m - 1 \tag{4-10}$$

A popular algorithm for computing this operation, and the one that appears to be most suitable for hardware implementation, is the square-and-multiply (S&M) algorithm as described below,

Let $(e_0, e_1, \cdots, e_{m-2}, e_{m-1})$ be the binary representation of the exponent E such that $E = \sum_{i=0}^{m-1} e_i 2^i$. Then, by Equation 4-10 we have

$$Y = \prod_{i=0}^{m-1} \left(A^{2^i}\right)^{e_i} = \prod_{i=0}^{m-1} U_i \tag{4-11}$$

where

$$U_i = \begin{cases} A^{2^i} & \text{if } e_i = 1 \\ 1 & \text{if } e_i = 0 \end{cases} \tag{4-12}$$

Thus, the algorithm breaks the exponentiation operation into series of squaring and multiplication operations in $GF(2^m)$ and can be expressed as follow:

Let U_i be the square term and M_i the product term.

$$U_{-1} = A$$
$$M_{-1} = 1$$
$$\text{for } i = 0, \cdots, m-1$$

$$U_i = [U_{i-1}]^2 \tag{4-13}$$

$$M_i = \begin{cases} 1 \cdot M_{i-1} & \text{if } e_i = 0 \\ U_{i-1} \cdot M_{i-1} & \text{if } e_i = 1 \end{cases} \tag{4-14}$$

The final result is $Y = A^E = M_{m-1}$

Let T_s be the setup time of the exponentiator (i.e. time corresponding to serial data transfers) and N the number of multiplication operations (including squaring operations) per exponentiation. The exponentiation time T_e can be expressed as $T_e = T_s + NR^{-1}T_{clk}$, where R is the throughput rate of the multiplier(s) and T_{clk} is the clock period. In terms of computation effort, assuming a random

input, the number of ones in the exponent is equal to $m/2$, so N can be expressed as $(m-1)+m/2 \approx 3m/2$ for large m. However, using synchronous exponentiator N becomes $(m+1)+m \approx 2m$ since the multiplication in Equation 4-14 for $e_i = 0$ has to be carried out.

4.3 PRIMITIVE POLYNOMIALS AND FIELD SIZE FOR FINITE FIELD BASED CRYPTOSYSTEMS

Although, all practical DL based public-key schemes require operations in relatively large Finite Fields; e.g., $m > 500$ bits (see Section 3.4.3.3 and [48][49]), the field-size m is selected so that 2^m-1 is a large prime (a *Mersenne* prime). There are certain values of field-size for which the period of the LFSR is the maximum, namely 2^m-1, which is the all possible states of m-bit vector, excluding the all-zero state. A maximum length sequence occurs if the reduction polynomials for constructing extension field corresponding to the LFSRs are prime elements of $GF(2^m)$ [75].

In addition, arithmetic in $GF(2^m)$ can usually be implemented more efficiently if the chosen irreducible polynomial has few non-zero terms. Since the least significant coefficient of any prime polynomial must always be non-zero (otherwise the polynomial has 0 as a root), the Hamming weight of the prime polynomials of degree at least 2 with few non-zero coefficients, must be odd and have at least 3 coefficients (prime polynomials of low hamming weight). Polynomials of Hamming weight 2, 3, 4 are called *trinomials*, *quadrinomials* and *pentanomials* respectively. An irreducible trinomial of degree m must be of the form x^m+x^k+1, where $1 \le k \le m-1$. In fact, both the complexity and energy consumption of mod $p(x)$ operation could be significantly reduced by selecting k with smaller value and less Hamming weight [76]. Table 4-5, gives some practical values for parameter k and, the field-size m, for which an irreducible trinomial of degree m in Finite Field exists.

m	k
521	32,48,158,168,353,363,473,589
607	105,147,273,334,460,502
1279	216,418,861,1063
2281	715,915,1029,1252,1366,1566
3217	67,576,2641,3150
4423	271,369,370,649,1393,1419,2098, 2325,3004,3030,3774,4053,4054, 4152

Table 4-5: Most useful irreducible trinomials x^m+x^k+1, for each large Mersenne prime m, $512 \le m \le 4423$

Further, since elements in one representation can be efficiently converted to elements in the other representation by using an appropriate change-of-basis matrix, the intractability of the DLP isn't affected by the choice of representation.

4.4 ARCHITECTURES FOR MULTIPLICATION OVER LARGE $GF(2^m)$

Various architectures have been proposed to perform modular multiplication operation efficiency in $GF(2^m)$. Different basis representation have been used to obtain some interesting realizations [75][98][89][78][88]. The parallel approaches, aren't to be enumerate here, since for large m the multiplier has to be of serial type. In fact for an arbitrary $GF(2^m)$ the gate count for a bit-parallel multiplier using either a PD or NB is proportional to m^2. In that case, area complexity increases dramatically for large field size.

A synthesis comparison among DB, NB and SB is given in [79][80]. There, the gate count is a guideline for the implementation complexity. It shows that, multipliers based on NB and DB requires basis conversion. Moreover, the area of NB multiplier grows dramatically as the order of the field goes up when optimal normal basis doesn't exist. Even when an optimal normal basis is chosen, the size complexity is proportional to $3m$. Also, both DB and NB are not highly modular or expendable [81]. Instead, the SD multiplier does not require basis conversion, its size and time complexity are proportional to m and it is readily matched to any input or output system [79]. The polynomial basis multiplier can be implemented using different architectures. The systolic and/or semi-systolic multipliers are described in [89][78]. These architectures are pipelined and regular 2-D systolic arrays based on approach similar to the bit-serial one (MSR) [75]. Their hardware implementations use m bit-serial parallel multipliers resulting in expensive hardware offering a high bit rate. The systolic multiplier described in [89] and [78] is thus not very attractive for large Finite Field.

4.4.1 BIT-SERIAL PB MULTIPLIERS

The MSR architecture described in [75] is a simple and area efficient (LFSR based multiplier) way of implementing SB multiplication over large Galois field. The nice bit-slice depicted in Figure 4-11, simplifies the VLSI design for large field size. The input elements $A(x)$ and $B(x)$ and the output product $C(x)$ are bit serial and the computation proceeds in bit-parallel fashion by convolution and reduction modulo an irreducible polynomial $p(x)$ of degree m. For more details about the algorithm, see [75].

The multiplication is performed with order $O(m)$ in both computation time and implementation area. $2m$ clock cycles are required between the first-in and first-out of computation and two-bits control signal is required. The MSR architecture is programmable with respect to the primitive polynomial $p(x)$ and field order m, using extra gates in each multiplier cell [75]. The complexity can be further reduced for implementations that use irreducible polynomials with few coefficients such as trinomials or pentanomials.

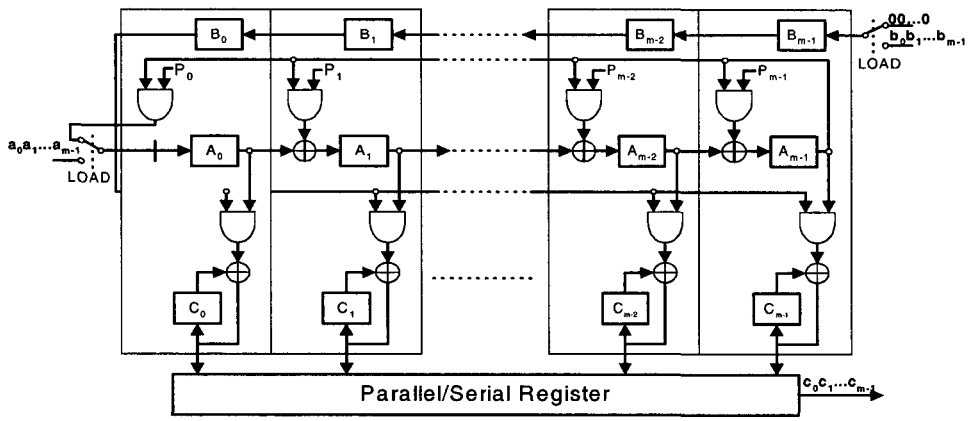


Figure 4-11: MSR m-bit multiplier architecture

In addition, it is easy to make the multiplier work as squarer since squarer can be realized as a bit-serial multiplier. This, simplify the design of the MSR based exponentiator in which, squaring can be carried out concurrently with multiplication[75].

Another architecture LSA performing SD multiplication in $GF(2^m)$ is described in[88] and. reported in Figure 4-12. It is bit-level pipelined and has a longest delay path independent of m. This is most suited to applications where m is large or where high clock frequencies are required. Further, the architecture allows the input elements to enter a linear systolic array in the same order and the system only requires one pipelined control signal. The architecture is also highly regular, expendable to any field size and programmable with respect to the primitive polynomial $p(x)$. The multiplication is performed in bit serial manner using the recursive algorithm in Equation 4-15. For more details about the algorithm, see [88].

$$c_i^{(k)} = \begin{cases} c_{m-1}^{(k-1)} p_i + a_i b_{m-1-k} + c_{i-1}^{(k-1)}, & 0 < i \le m-1; \\ c_{m-1}^{(k-1)} p_0 + a_0 b_{m-1-k}, & i = 0 \end{cases} \qquad (4\text{-}15)$$

$$c_i^{(-1)} = 0 \quad for \quad 0 \le i \le m-1$$

Where c_i^{m-1} for $0 \le i \le m-1$ (output of the last cell-k) are the coefficients of the product $C(x) = A(x) \cdot B(x) \bmod p(x)$ and p_i for $0 \le i \le m-1$ are the coefficients of the polynomial generator.

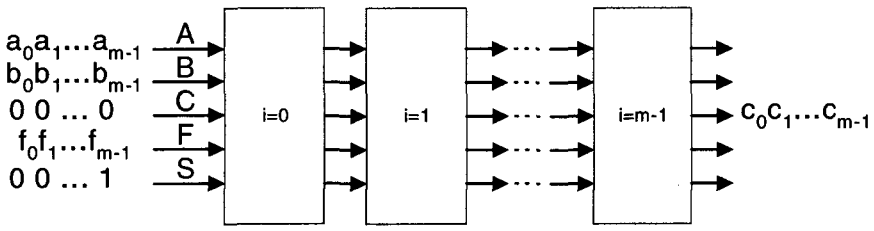


Figure 4-12: Linear systolic array multiplier

The circuit diagram of CELL-k is shown in Figure 4-13. Two internal registers b and c are used to hold the bit coefficients b_{m-1-k} and $c_{m-1}^{(k-1)}$ along the operation using the s_{in} signal which mark the start of the multiplication. These coefficients are then used to compute CELL-k output (Equation 4-15) at next clock cycle when $s_{in} = 0$. Three registers a, p and s are used to give one time unit delay to the input bit coefficients a_i, p_i and s_i. The outputs are then triggered using a next register output stage in master slave manner as shown in Figure 4-13.

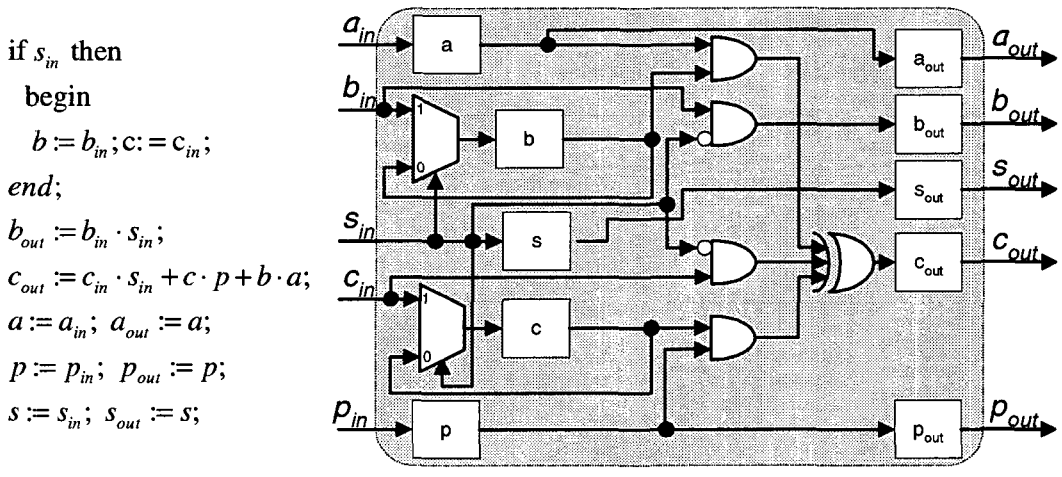


Figure 4-13: LSA basic processing cell and its algorithm

The m-bit multiplication time takes $3m$-1 clock cycles. At $2m$ clock cycles after a_m-1 and b_m-1 enter the leftmost cell; the results will start coming out from the rightmost cell at the rate of one coefficient every clock cycle.

This multiplier is programmable with respect to the primitive polynomial $p(x)$. The algorithm is often advantageous because of its efficient implementation time. The critical path in this architecture is just the sum of one full-adder and one NAND gate delay.

4.4.2 ARCHITECTURE LEVEL TRANSFORMATION

In this section we extend the MSR and LSA bit serial multipliers to a generalized digit-serial architecture, which is array-type at the digit-level using parallel multiplication algorithm inside of each digit cells. These architectures are obtained by folding the bit-serial multipliers.

4.4.2.1 DIGIT-SERIAL MSR MULTIPLIER

The presence of long loaded lines (for large m) in architecture reported in Figure 4-11, affects directly the maximum clock frequency and consequently the system performance. This can be avoided (reduced) using digit-serial technique by folding the bit-serial MSR architecture. However, the MSR digit-serial architecture cannot be pipelined below digit-level because of the presence of the feedback loops in the MSR bit-serial architecture. This linear dependency in mod $p(x)$ degree reduction operation can be easily implemented using trinomials as field-generating polynomials.

The transformation approach involves treating the multiplier operands as digits: the m bits of data operands are processed in units (digits) of size D using $d = \lceil m/D \rceil$ slices. Let

$$A = \sum_{i=0}^{m-1} a_i x^i, B = \sum_{i=0}^{d-1} B_i x^{Di}$$

where

$$B_i = \begin{cases} \sum_{j=0}^{D-1} b_{Di+j} x^i, & 0 \le j \le d-2 \\ \sum_{j=0}^{m-1-D(d-1)} b_{Di+j} x^i, & j = d-1 \end{cases} \tag{4-16}$$

then

$$C = A \cdot B \mod p(x) = A \cdot \sum_{i=0}^{d-1} B_i x^{Di} \mod p(x) \tag{4-17}$$

This results on array-type multiplication, which can be performed in the following way:

$$\begin{aligned} C = &\left[B_0 A(x) \mod p(x)\right] + \left[B_1 (A(x) \cdot x^D \mod p(x)\right] \\ &+ \left[B_2 \cdot (\cdot x^D (A(x) \cdot x^D \mod p(x)))\right] + \cdots \\ &+ \left[B_{d-1} (x^{D(d-2)} (A(x) \cdot x^D \mod p(x)))\right] \end{aligned} \tag{4-18}$$

We define now the polynomials Z_{-j} as:

$$Z_{-j}(x) = \sum_{i=0}^{d-1} z_{i,j} (x^D)^i = (x^D)^j A(x) \mod p(x), j = 0,1,...,m-1 \tag{4-19}$$

where $z_{i,j} \in GF(2)$. Then:

$$C(x) = \sum_{j=0}^{d-1} B_j Z_{-,j}(x) \tag{4-20}$$

and in matrix notation:

$$C = \begin{pmatrix} C_0 \\ C_1 \\ . \\ . \\ C_{d-1} \end{pmatrix} = \begin{pmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,d-1} \\ z_{1,0} & z_{1,1} & \cdots & z_{1,d-1} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ z_{d-1,0} & z_{d-1,1} & \cdots & z_{d-1,d-1} \end{pmatrix} \bullet B = Z \bullet B \tag{4-21}$$

Where Z is a d by d digit matrix. The columns of Z are the d consecutive states of a Galois-type digit-parallel LFSR with feedback polynomial $p(x)$, that has been initially loaded with A ($=Z_{-,0}$). The product is therefore obtained by first computing $B_0 Z_{-,0}$ and storing the result in d stage register of D-bits size. Next we clock the LFSR, compute $B_1 Z_{-,1}$, add it to $B_0 Z_{-,0}$, and store the result and so forth. After d clock cycles the product is available in the lower register. The general form of the circuit is shown in Figure 4-14 for large Mersenne prime, using trinomial primitive polynomial with appropriate choice of parameter k (m, k are selected from Table 1.). The structure is kept simple and highly regular.

The explanatory notes for the italic line / across the signal lines denote the weights of the corresponding signals, i.e., $< D - X_1 >_{LSB}$ means that the corresponding line carries the D-X_1 least significant bits of the corresponding signal. The values of X_1 and X_2 are reported in Table 4-6 with respect to the value of parameter k, the field-size m and digit-size D.

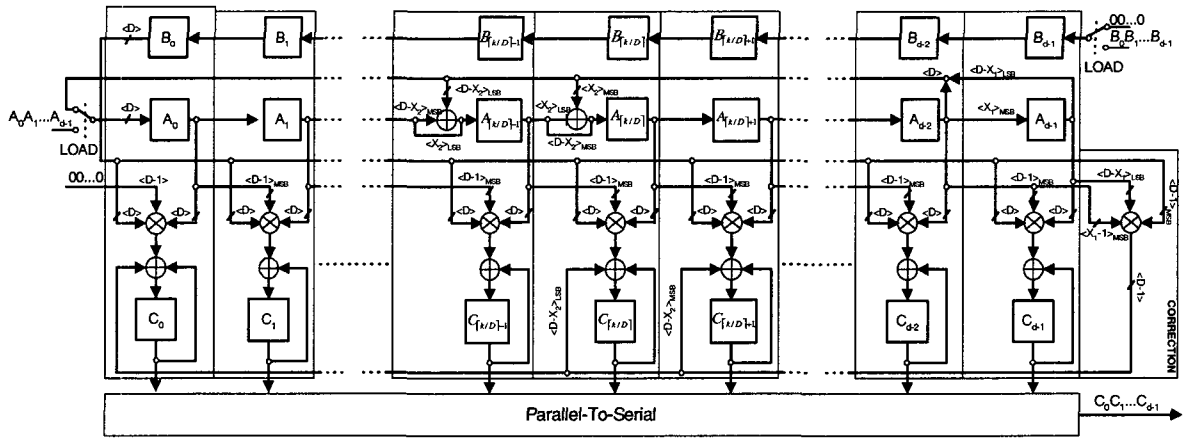


Figure 4-14: Digit-serial MSR multiplier for field-generating polynomial $p(x)=1+x^k+x^m$

The LFSR performs the computation in Equation 4-2011, i.e., $A(x)$ multiplied by x^D followed by mod $p(x)$. The partial product generator denoted by ⊗ computes $B_i Z_{-,i}$ in (12). The accumulator is denoted by ⊕ and performs the sum operation in Equation 4-21; it consists of XOR gates rearranged from array-type to tree-type and storage elements, where the partial product $B_i Z_{-,i}$ and the intermediate result are accumulated using the binary-tree of XOR gates. At each cell, only D LSB-

bits of the partial product are computed. At the last cell, a correction must be done in order to reduce the degree of the result from $m+D-2$ to $m-1$. This can be done efficiently in one step using a feedback signal. The polynomial degree is reduced using AND and XOR gates (Equation 4-7). The total computation time takes $3d$ clock cycles between the first-in digit and the last-out digit.

m	X_1			k	X_2		
	D=4	D=8	D=16		D=4	D=8	D=16
521	7	7	23	32	0	0	0
				48	0	0	16
				158	6	12	30
				168	0	8	8
607	1	1	1	105	1	9	9
				147	3	3	19
				273	1	1	17
1279	1	1	1	216	0	8	24
				418	2	2	2
1281	7	15	31	715	3	11	11
				915	3	3	19
				1029	5	5	5
3281	7	15	15	67	3	3	3
				576	0	0	0
4423	1	9	25	271	7	15	15

Table 4-6: X_1 and X_2 values for digit size D = 8,16,32

4.4.2.2 LINEAR DIGIT-SERIAL SYSTOLIC ARRAY MULTIPLIER

Digit-serial LSA architecture is obtained by folding the bit-serial multiplier. We propose here a methodology to design a programmable digit-serial Finite Field multiplier shown in Figure 4-12.

Consider the structure of the bit-serial multiplier. The transformation approach involves treating the bits in this multiplier as digits. Therefore, the inputs bit a_i, b_i, c_i and p_i for $0 \le i \le m-1$ to CELL-k in Figure 4-13, are replaced by digits forms A_i, B_i, C_i, P_i for $0 \le i \le d-1$ where,

$$(A_i,B_i,C_i,P_i)=\begin{cases}\sum_{j=0}^{D-1}(a_{Di+j},b_{Di+j},c_{Di+j},p_{Di+j})x^j, & 0\le i\le d-2\\ \sum_{j=0}^{m-1-D(d-1)}(a_{Di+j},b_{Di+j},c_{Di+j},p_{Di+j})x^j, & i=d-1\end{cases} \tag{4-22}$$

and

$$\begin{aligned}A(x)&=\sum_{i=0}^{m-1}a_ix^i=\sum_{i=0}^{d-1}A_ix^{Di},\quad B(x)=\sum_{i=0}^{m-1}b_ix^i=\sum_{i=0}^{d-1}B_ix^{Di}\\ C(x)&=\sum_{i=0}^{m-1}c_ix^i=\sum_{i=0}^{d-1}C_ix^{Di},\quad P(x)=\sum_{i=0}^{m-1}p_ix^i=\sum_{i=0}^{d-1}P_ix^{Di}\end{aligned} \tag{4-23}$$

where, D denotes the digit-size and d the total number of digits, $d = \lceil m/D \rceil$.

Suppose that the resulting architecture can be implemented on a linear digit-serial systolic array, as shown in Figure 4-15. The inputs digit-words A_i, B_i, C_i, P_i are fed into the multiplier in the same order for i decreasing and from the MSB to the LSB. If m is not divisible per D, the zero padding is performed at the LSB positions for $i = 0$.

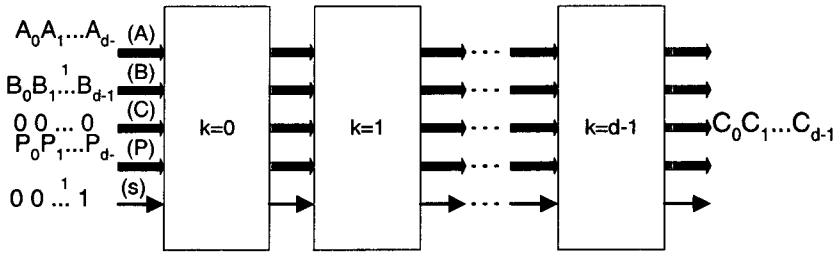


Figure 4-15: Digit-serial, linear systolic array multiplier

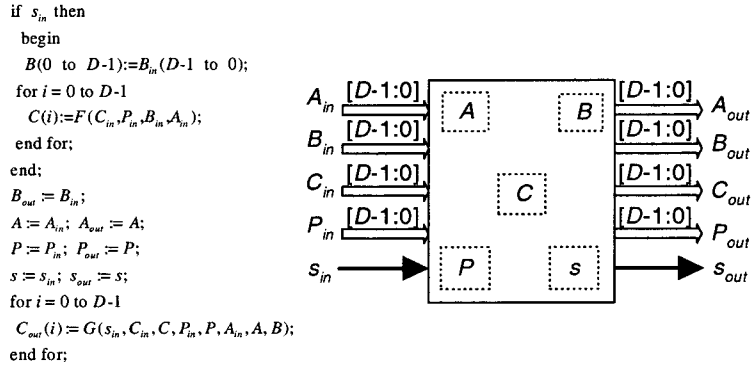


Figure 4-16: LSA digit-serial basic processing cell and its algorithm

The system now consists of d identical cells for $D \cdot d$-bit multiplication in $GF(2^m)$. It inputs the data at the leftmost cell and outputs the results at the rightmost cell at the rate of one digit every clock cycle. The basic processing element CELL-K of the multiplier is shown in Figure 4-16. Two D-bit registers A, P and 1-bit s registers are used to give one time unit delay to the input data A_i, P_i and s_i at each CELL-k. The s signal is used to denote the start of a multiplication.

The algorithm is obtained by grouping each set of D cells from the LSA multiplier in Figure 4-12, then computing the outputs of each of these grouped cells after D steps (clock cycles). These are the outputs of the resulting digit cell. This is illustrated in the example bellow.

Example 1.

Consider the computation of $C(x) = A(x) \cdot B(x) \mod p(x)$ over $GF(2^7)$ where $A(x) = 1 + x^2 + x^4 + x^6$, $B(x) = x + x^3 + x^5$ and $p(x) = 1 + x^3 + x^7$.

Consider now the basic processing element CELL-k and its algorithm given in Figure 4-13. Let each CELL-k be represented using the I/O signals and the state of its internal registers as shown in Figure 4-17.

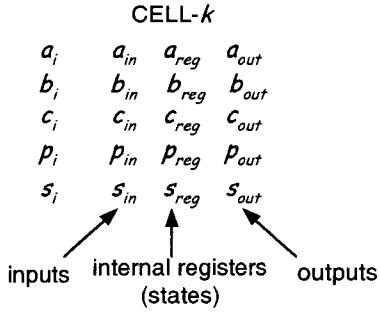


Figure 4-17: LSA CELL-K multiplier representation

The I/O signals and the state of internal registers at each CELL-k for each computation step are reported in Figure 4-17. The steps reported at the right side of each CELL-6 represents the steps corresponding to the digit-serial architecture for D=3.

Therefore, by folding the bit-serial computations in Figure 4-17, and after $3d$-1 steps the output of the multiplier expressed in digit form is as follows:

$$c_{out} \rightarrow 000\ 000\ 000\ 000\ 000\ \mathbf{011\ 001}\ 000$$

Thus, the C internal register of each CELL-k in the digit-serial multiplier are expressed as follows:

LSA multiplier cell LSA multiplier time step

$$C(0) = c_{in}(0,0) \rightarrow C(0) = C_{in}(2)$$

$$C(1) = c_{in}(1,2) = c_{out}(0,1) = c_{in}(0,1) \cdot \bar{s}_{in}(0,1) + c(0,0) \cdot p(0,0) + b(0,0) \cdot a(0,0)$$
$$= c_{in}(0,1) \cdot \bar{s}_{in}(0,1) + c(0) \cdot p(0,0) + b(0) \cdot a(0,0)$$
$$\rightarrow C(1) = C_{in}(1) + (C_{in}(2) \cdot P_{in}(2)) + (B_{in}(2) \cdot A_{in}(2))$$
$$C(2) = c_{in}(2,4) = c_{out}(1,3) = c_{in}(1,3) \cdot \bar{s}_{in}(1,3) + c(1,2) \cdot p(1,2) + b(1,2) \cdot a(1,2)$$
$$= c_{in}(1,3) \cdot \bar{s}_{in}(1,3) + c(1) \cdot p(1,2) + b(1) \cdot a(1,2)$$

where, $c_{in}(1,3) = c_{out}(0,2) = c_{in}(0,2) \cdot s_{in}(0,2) + c(0) \cdot p(0,1) + b(0) \cdot a(0,1)$

$$\rightarrow C(2) = (C_{in}(0) + (C_{in}(2) \cdot P_{in}(1)) + (B_{in}(2) \cdot A_{in}(2))) + [(C_{in}(1) + (C_{in}(2) \cdot P_{in}(2)) + (B_{in}(2) \cdot A_{in}(2))) \cdot P_{in}(2)] + (B_{in}(1) \cdot A_{in}(2))$$

Note that all the states of C registers must be computed during one clock cycle. The C_{out} register's bits at the output of each CELL-k, are then expressed as follows:

$$C_{out}(2) = ((((C_{in}(2)+(C(0) \cdot P(0))+(B(0) \cdot A(0)))+(C(1) \cdot P(1)) + (B(1) \cdot A(1))) \cdot \bar{s}_{in}) + (C(2) \cdot P(2)) + (B(2) \cdot A(2)));$$
$$C_{out}(1) = ((((C_{in}(1)+(C(0) \cdot P_{in}(2))+(B(0) \cdot A_{in}(2))) + (C(1) \cdot P(0)) + (B(1) \cdot A(0))) \cdot \bar{s}_{in}) + (C(2) \cdot P(1)) + (B(2) \cdot A(1)));$$
$$C_{out}(0) = ((((C_{in}(0)+(C(0) \cdot P_{in}(1))+(B(0) \cdot A_{in}(1))) + (C(1) \cdot P_{in}(2)) + (B(1) \cdot A_{in}(2))) \cdot \bar{s}_{in}) + (C(2) \cdot P(0)) + (B(2) \cdot A(0)));$$

We can extend these expressions to a D-bit digit words and drive a generalized algorithm by formulating the expression of F and G functions.

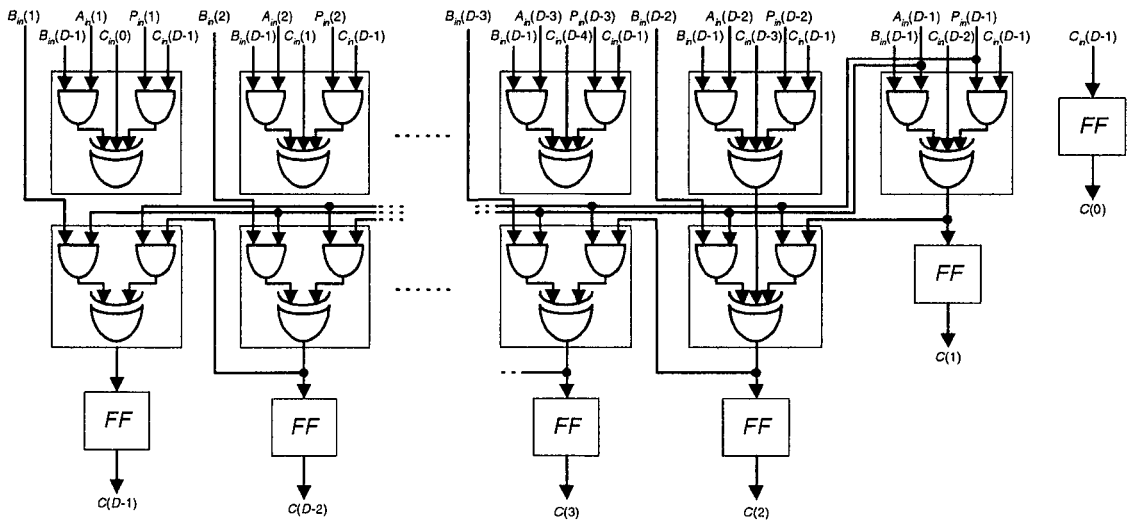


Figure 4-18: Circuit diagram of the F function

Figure 4-19: Folding bit-serial computations

In Figure 4-16, F denotes the function processing the state of C internal register. The circuit diagram of F function is shown in Figure 4-18, where FF denotes a flip-flop. Note that the critical path is $(D\text{-}1)(T_{\text{XOR-3}}+T_{\text{NAND-2}})$.

The G function process the state of the output register C_{out}. The corresponding circuit diagram for one output coefficient is shown in Figure 4-20. The critical path in this architecture is increased to $D.T_{\text{XOR-3}}+T_{\text{XOR-2}}+2T_{\text{NAND-2}}$

The d-bit multiplication implemented within architecture shown in Figure 4-15, takes $3d$-1 clock cycles. At $2d$ clock cycles after A_{d-1} and B_{d-1} enter the leftmost cell, the results will start coming out from the rightmost cell at the rate of one digit every clock cycle.

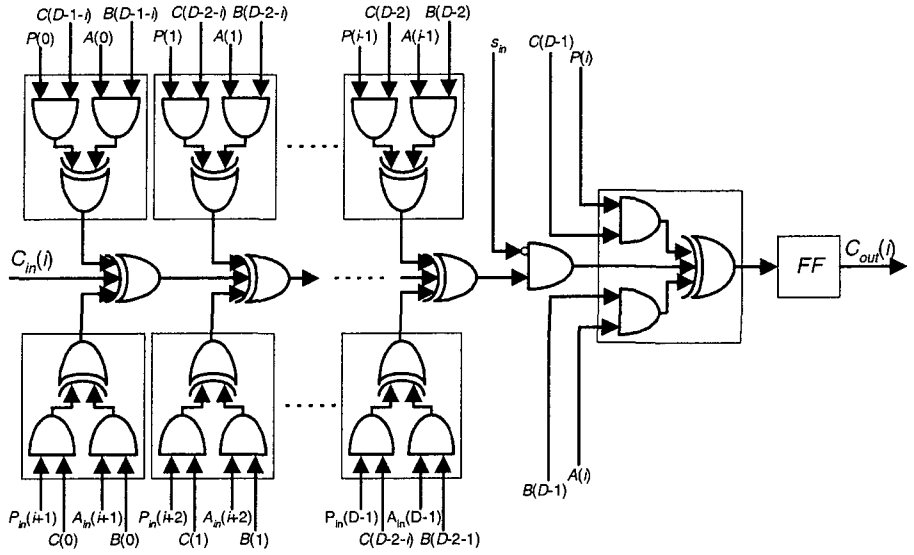


Figure 4-20: Circuit diagram of the G function for one bit output

4.4.3 IMPLEMENTATION ISSUES AND COMPARISON

Clock gating technique can be used for power-efficient implementation of registers that are disabled during some clock cycles, when such registers maintain the same value through multiple cycles (see Section 2.3.2), such as the internal slave registers c and b in Figure 4-13 and C and B shown in Figure 4-16. These registers have their own load controlled by s_{in} signal. This technique works well for data-flow logic, where clocking requirements can be predetermined at least one cycle ahead. Thus, the clock gating enable signal s_{in} must be valid halfway into the cycle to gate off the capture clock. To overcome this problem, we require that the internal registers b, c, C and B be triggered faster than the master registers b_{out}, c_{out}, B_{out} and C_{out} using different clock edges. This requires one more clock pulse, resulting in 2-phase non-overlapping clocking scheme.

The MSR, LSA and clock gated LSA architectures have been implemented at gate level using different digit-size D=1,4,8,16 in order to perform a comparison in terms of speed, area and energy consumption of $GF(2^{607})$ multiplier with $p(x)=1+x^{273}+x^{607}$ as primitive polynomial. We mapped our circuits into a deep sub-micron (0.18µm) target library from XEMICS (Coolib) that contains logic-gates optimized for power, operating at two different power supply 1.8v and 0.9v.

Different types of power dissipation components are estimated using gate level simulations on a set of random stimulus. Since *low-energy* design is more important than *low-power* design, the energy and energy-delay product is computed. The performance characteristics including total delay, area in term of gates and energy-delay product and are reported in Figure 4-21, Figure 4-22 and Figure 4-23 respectively. The power numbers are reported in Appendix C.1.

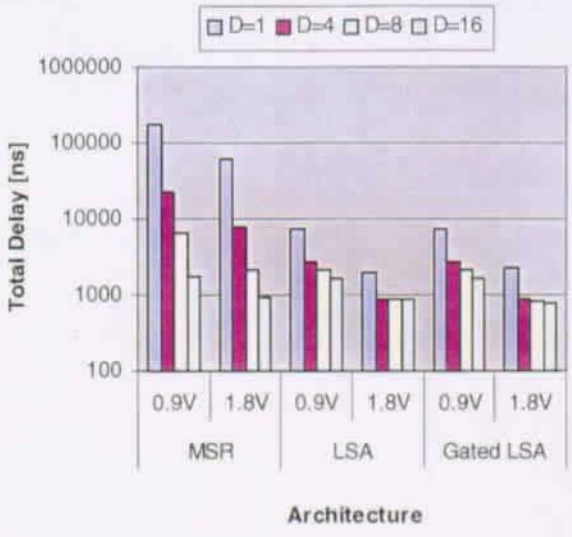


Figure 4-21: Total delay comparison as function of the digit-size for one GF(2^{607}) multiplication

The long signals that are distributed to all slices in Figure 4-11 are susceptible to degradation due to the large capacitive loads. For example, the serial input multiplier bit h_i is a long line that has to drive m AND gate. This signal must drive up to 11.76pF capacitive load. The source of this line will be trying to push current into the entire load, experiencing a very substantial RC delay, which increases considerably the critical path and then the total delay as shown in Figure 4-21. We can consider using fast buffers to isolate heavy loads. However, buffering the architecture can severely degrade system performance. It increases the critical path and creates the problem of skewed signals. The parallelism inside each digit-cell in Figure 4-14 contributes to reduce the load on such long heavily loaded signals, i.e., when the chosen digit-size is 8 the capacitive load is significantly reduced to 1,35pF per bit-line for the most heavily loaded line (input multiplier digit-word B_j), experiencing over 72% improvement in circuit speed when operating at 1.8v and 96% when operating at 0.9v.

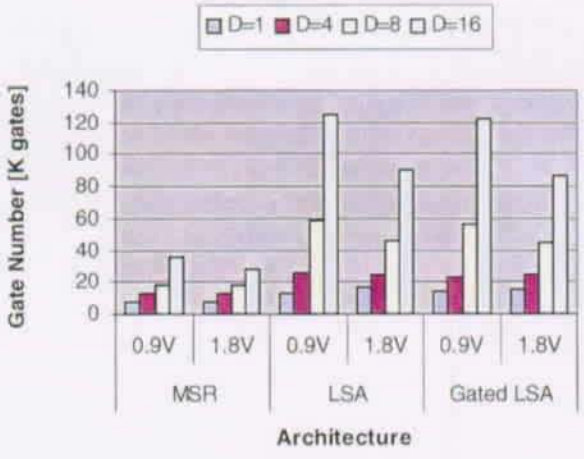


Figure 4-22: Area in gates of the MSR, LSA and clock gating LSA digit-serial GF(2^{607}) multipliers as function of the digit size

Bit-level pipelining approach makes the LSA multiplier architecture more advantageous in term of clock frequency and computation time. Both total delay and energy consumption are reduced. On one hand, latency decreases linearly with the digit-size but the critical path increases linearly in almost the same rate (Figure 4-21), resulting in a constant total delay for digit-size equal or larger then 4. On the other hand, the area increases dramatically (Figure 4-22) due to the large number of latches used to temporary hold the internal and the output data in master-slave manner. This means that the level of parallelism is limited by the area constraints.

The most interesting result is obtained when comparing the Energy-Delay and the Energy-Delay-Area products. The performance characteristic reported in Figure 4-23, shows that Energy-Delay products are significantly reduced for both LSA and MSR architecture when digit-size increase. High gain is obtained for D=16 when operating at 0.9v and more than 99% reduction is noticed. However, when comparing the characteristic reported in Figure 4-24, the optimum gain for LSA architecture is obtained for D=4 when operating at 1.8v due to the dramatic increase in circuit area for larger digit-size. This is not the case when operating at 0.9v since the energy is significantly reduced.

Beside the programmability with respect to the primitive polynomial of the LSA architecture, the MSR present the best performance characteristic only for digit-size equal or larger then 8 due to the large critical path for small digit-size.

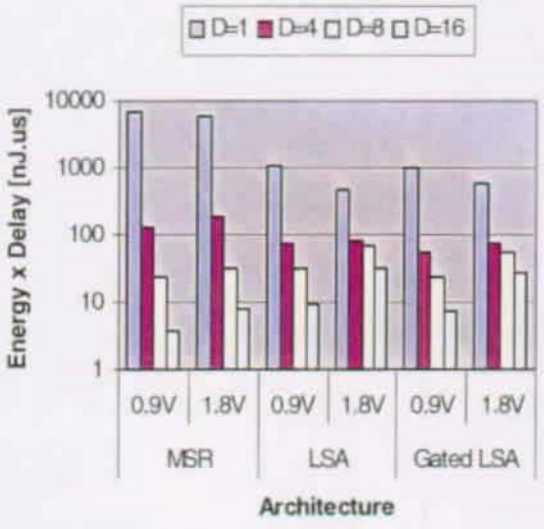


Figure 4-23: Energy-Delay product comparison between MSR and LSA digit-serial GF(2^{607}) multipliers

Clock gating technique inserted for LSA multiplier achieves a substantial reduction in both the Energy-Delay (over 28% at 0.9v and 17% at 1.8v for D=8) and the Energy-Delay-Area product (over 30% at 0.9v and 20% at 1.8v for D=8) for only 22% of clock gated registers. Clock gating reduces the number of gates in such architecture (multi-bit registers) when digit-size increase. It helps to eliminate the feedback loops and multiplexers used to feed back the output of each internal storage elements back to the input of synchronous load-enable registers. Such feedback loops and multiplexers are replaced by only one integrated cell (latch based clock gating cell), which results in 3.5% and 4.3% reduction in gate number at 0.9v and 1.8v respectively, when the chosen digit-size is 8.

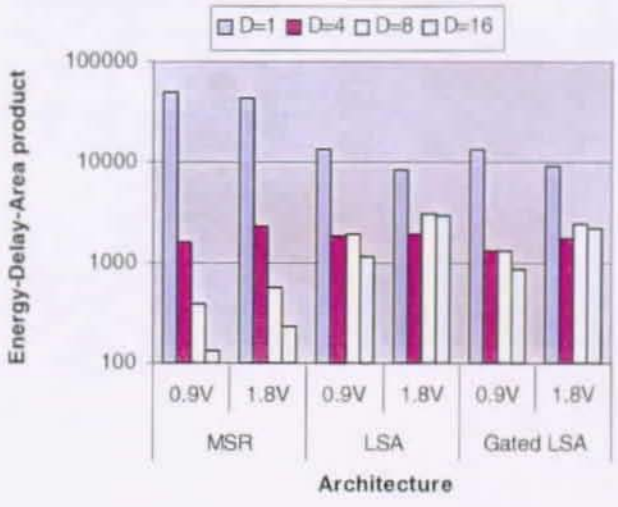


Figure 4-24: Energy-Delay-Area product comparison between MSR and LSA digit-serial $GF(2^{607})$ multipliers

When reducing the operating voltage by a factor $\delta = 2$ the switching power is reduced by factor δ^2 in Equation 2-1, from Equation 2-2 assuming that $V_{DD} >> V_t$ the delay is increased by factor δ. If the switching power is a dominant factor then, the power saving is counterbalanced by the increased delay since the energy-delay product is proportional to δ^2.

4.5 EXPONENTIATION OVER LARGE $GF(2^m)$

4.5.1 LOW POWER ARCHITECTURES FOR LARGE $GF(2^m)$ EXPONENTIATION

One new method for computing $GF(2^m)$ exponentiation is presented in [87]. It is based on pattern matching and recognition technique. The resulting circuit is a multistage linear static pipeline and it can produce one result every clock cycle. However it has latency of $O(2^m)$. The area and complexity increase systematically when large field is used. This technique is more suitable for small field-size ($m \leq 8$) as mentioned in the paper. Instead, S&M algorithm also called the binary method described in Section 4.2.7 breaks the exponentiation operation into a series of squaring and multiplication operations in $GF(2^m)$. This is the most adopted technique to design large finite field exponentiation.

4.5.1.1 REDUCING NUMBER OF OPERATIONS

Some interesting approaches have been proposed in [91] and [92] in order to reduce substantially the average number of multiplications involved in the computation of A^E in Figure 4-10 (thus reducing the power consumption) using the *canonical bit recording* technique or *signed digit* SD number representation. Technique in [91] can be easily applied to the exponentiation over $GF(2^m)$. Efficient implementation of α^E over $GF(2^m)$ using SD technique and based on bi-directional linear feedback shift registers is proposed in [92]. However, the algorithm is limited to the exponentiation of a Primitive Root α and generally the SD technique require the construction of canonical signed-digit vector of the exponent E which, introduce much extra power and area cost since one additional bit is required for each scanned exponent bit. Also, both techniques described in [91] and [92] require

the availability of the inverse A^{-1}. The cost of this operation far exceeds the time gained by the use of the SD technique.

4.5.1.2 SPEEDING UP THE COMPUTATION

The conventional approach for speeding up the exponentiation over $GF(2^m)$ and reducing the number of operations uses a lookup table (LUT). This method consist of precomputing the field elements A^2 (A conjugates) in Equation 4-11 and storing them in circulating registers or in a RAM (area complexity ~ m^2) and multiplied together according to the exponent using fully parallel multiplication tree [75][96]. For large m the multiplication has to be performed using bit-serial multiplier (area complexity ~ m for both PB and NB, see Table 4-7). Furthermore, setup times corresponding to serial data transfers of the exponent and A's conjugates bits have to be considered, which increase the latency to less than one exponentiation per m^2 clock cycles. Thus, no area and timing gain is obtained using this approach even when m is moderately large. Another approach relies on the Montgomery multiplication in $GF(2^m)$ [93] as a fast method for multiplying two polynomials. However, the algorithm includes some operations similar to those in the bit-serial SB multiplication algorithm [75] or MSR digit-serial one described in Section 4.4.2.1 with more processing steps. This approach is more suitable for implementation in software as claimed in the paper and doesn't offer any obvious advantages for hardware implementation.

4.5.1.3 BASES CHOICE

Since the exponentiation algorithm relies on polynomial multiplication in $GF(2^m)$, the design of area efficient low-energy finite field multipliers can lead to dramatic improvement on the overall performance of $GF(2^m)$ exponentiator. Table 4-7 shows some key parameters for different architectures of serial $GF(2^m)$ multipliers using different bases.

It is well known that squaring operation in normal basis NB can be achieved by a cyclic-shift circuit [75][94]. However multiplication in NB requires the computation of the f function described in [94] which must be found by computer and hardwared. The complexity of this function grows dramatically as the order of the field goes up. Massey and Omura have developed a NB multiplication algorithm, which has been implemented in a pipelined architecture by Wang et al. [94] based on AND-XOR PLA and has a throughput rate of one multiplication per m clock cycles. As m increases, the signal propagation delay across the PLA (critical path) also increases. So, the area and total delay may actually be larger owing to the slower clock rate.

From Table 4-7, it is clear that normal basis multiplication using the serial Massey-Omura multiplier in [75] requires the least number of gates, unfortunately this is for optimal normal basis multipliers which can be realized for only ~23% of the fields $GF(2^m)$, $2 \le m \le 1200$ [97]. Another point to consider is that the NB multiplier is not highly modular and expendable. Further, it and is not programmable with respect to the $p(x)$ due to the f function, which depends on the choice of normal basis thus, on the choice of $p(x)$.

Multiplier/Basis	Complexity min/max	Length of CP	Latency min/max	Regularity	Easy to expand
MSR/SB[75][95]	4m/6m	3[a]	m/2m	High	Yes
LSA/SB[88]	17m/17m	5	3m/3m	High	Yes
Berlekamp/DB[98]	≥4m/6m	≥2+⌈ln(m)⌉	m/m	Low	No
MO/NB[94]	≥5m	≥2+⌈ln(m)⌉	m/m	Low	No

Table 4-7: Key parameters for different architectures of serial $GF(2^m)$ multipliers using different bases

a. Critical path in term of gates but in fact it is determined by the driving capability of heavily loaded signals that are distributed through the architecture.

On the other hand, the DB multiplier requires basis conversion, which requires extra gates. Although the Berlekamp DB multiplier [98] has a simple and regular structure it is not trivial to adapt it to different choice of $p(x)$ or to expand it to different field sizes because of the presence of two different bases dual and polynomial. Also the critical path depends on the logic for computing the inner product, which depends strongly on the field order and $p(x)$. This means that the speed decreases, although slowly, with m.

Even the PB multipliers have the most favourable properties for purpose of VLSI implementation, the critical path of the MSR multiplier described in Section 4.4.1 is limited by the driving capabilities of heavily loaded signals which degrade the performance as demonstrated in Section 4.4.3. On the other hand, the linear systolic array (LSA) multiplier described in [88] and reported in Figure 4-12 shares some interesting characteristics with the MSR multiplier. It has the advantage of efficient implementation time at the cost of increased complexity in term of gates.

4.5.2 LSA BASED ARCHITECTURE FOR $GF(2^m)$ EXPONENTIATION

In this section we present a new exponentiator circuit implementing S&M Algorithm and based on a linear systolic array multiplier (LSA) described above in Section 4.4.1. The successive SB squaring and multiplication operations are performed at high system frequency using only one multiplier in order to reduce the area complexity of the exponentiator. For VLSI implementation of high performance architecture, a digit-serial technique is applied on that multiplier in order to reduce both the switching activity and total power (thus, reducing the energy-delay products of the exponentiator) at the expense of increased area.

4.5.2.1 LINEAR SYSTOLIC ARRAY EXPONENTIATOR

In the LSA multiplier reported in Figure 4-12, the input operands (MSB-first), the control signal as well as the primitive polynomial coefficients are outputted at each pipelined stage. These signals with the result (MSB-first) could be feed-backed into the multiplier in order to perform successive multiplication and squaring operations according to S&M algorithm. The cells contain registers configured in master slave manner in which, internal data registers are used to give one more time unit delay to the operands at each cell. The first output bit of the multiplication result is available after $2m$ clock cycles. After m clock cycles the multiplier operands are completely loaded into the architecture allowing the load of the operands for the next operation (squaring or multiplication), while performing the computation of the current operation.

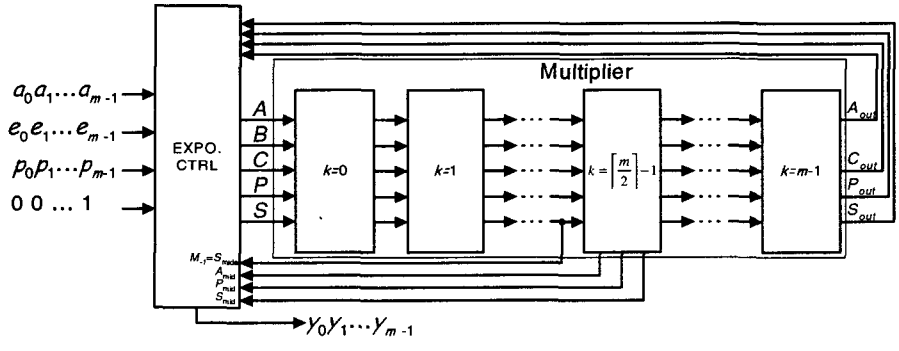


Figure 4-25: $GF(2^m)$ LSA based exponentiator for odd field size m

Thus, the operations in Equation 4-13 and Equation 4-14 could be overlapped to reduce the latency and thus the exponentiation time T_e. The general structure of the LSA based exponentiator is depicted in Figure 4-25, where p_i for $0 \le i \le m-1$ are the primitive polynomial coefficients and S is a control signal (only the first bit among the m bits contains the value one).

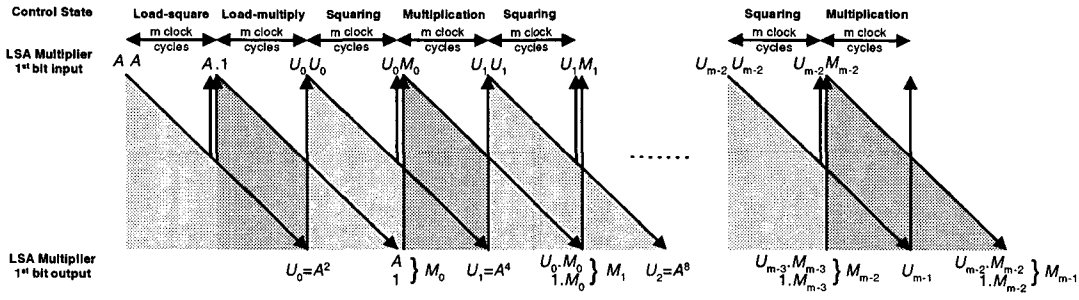


Figure 4-26: Overlapping the squaring and multiplication operations

The m-bit multiplication time takes $3m$-1 clock cycles. At $2m$ clock cycles after a_{m-1} and b_{m-1} enter the leftmost cell the results C_{out} will start coming out from the rightmost cell at the rate of one coefficient (MSB-first) every clock cycle.

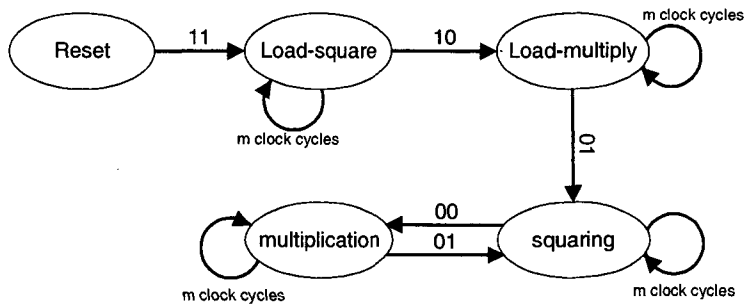


Figure 4-27: The exponentiator operation modes

The squaring is performed continuously. The feedback signal outputted at CELL-$\lceil m/2 \rceil$-1 contains the square term (operand) A_{mid}, which is used for computing the product $U_{i-1} \cdot M_{i-1}$ in Equation 4-14, according to the primitive polynomial coefficients P_{mid} and control signal s_{mid}. If m is odd, these feedback signals are outputted from the internal registers of CELL-$\lceil m/2 \rceil$-1 as shown in Figure 4-25, otherwise they are taken at the output of this cell. The squaring and multiplication bits results C_{out} are outputted iteratively from the last cell and used for computing the expression in Equation 4-13 and Equation 4-14 according to the primitive polynomial coefficients P_{out} and control signal S_{out}. Signal S_{mida} in Figure 4-25 is the multiplier control signal S that defines the operating modes of the exponentiator, it generate two internal control signals, load and square, which defines 4 different operating modes as shown in Figure 4-27. If m is even then is outputted from the internal register s of CELL-$\lceil m/2 \rceil$-1. It is also used as $GF(2^m)$ unit operand M_{-1} for the computation of the $U_{-1} \cdot M_{-1}$ in Equation 4-14. A transition in this signal at the input of CELL-$\lceil m/2 \rceil$-1 means that the LSB of the inputs operands are completely loaded into the multiplier allowing the load of next operands. Note that the multiplication $1 \cdot M_{i-1}$ $(0 \le i \le m-1)$ in Equation 4-14 is not performed. We can cope with this using the feedback operand A_{out} as the product term M_{i-1}.

We start the computation in *load-square* mode. In this mode we perform a squaring operation on the value of the base A. Once the coefficients are loaded into the multiplier after m clock cycles, we switch the exponentiator to the *load-multiply* mode. In this mode we perform the multiplication operation $U_{-1} \cdot M_{-1}$. Once the M_{-1} coefficients are loaded into the multiplier after m clock cycles, we switch to the *squaring* mode. In this mode we perform the computation in Equation 4-13, then we switch to the *multiplication* mode in which we perform the computation in Equation 4-14. We repeat successively these two last operating modes until the end of operation. This is described in the diagram reported in Figure 4-26.

For signals stability, different clock edges are used to handle the feedback signals in Figure 4-25. Control states are then generated within the clock pulse and feedback signals are triggered by the inverted clock at the exponentiator controller. The exponent E is fed into a LIFO stack (buffer) of size m. The buffer content is shifted one time when the test in Equation 4-14 is performed. Hence, the buffer is completely empty after the output of the final result.

The setup time T_s of the exponentiator is equal to m clock cycles corresponding to serial data transfers of the inputs. The throughput rate R of the multiplier is equal to one multiplication operation per m clock cycles. At m clock cycles, after the LSB of the inputs A, E, P and S enter the exponentiator architecture, the exponentiation result Y will start coming out from the rightmost cell after $2m \cdot m$ clock cycles at the rate of one bit per clock cycle. Hence, the exponentiation time T_e becomes $2(m+1) \cdot m \cdot T_{clk}$. Compared with the MSR based exponentiator described in [94], witch perform the computation in $(m+2) \cdot m$ clock cycles from the first-in bit to the last-out bit, the latency is increased by m^2 clock cycles. However, as demonstrated above in Section 4.4.3, the MSR based multiplier presents long heavily loaded signals that increase the critical path and the total delay for one multiplication operation is 96% higher. Hence, the latency gain of the MSR based exponentiator is counterbalanced by the clock period. Furthermore, the area complexity of both exponentiators is almost the same $\sim 17m$.

4.5.2.2 DIGIT-SERIAL LINEAR SYSTOLIC ARRAY EXPONENTIATOR

The generated digit-serial architecture is obtained from the LSA digit-serial multiplier (Figure 4-15). The resulting exponentiator circuit is linear array-type at the digit-level based on the parallel multiplication algorithm inside of each digit cells.

The successive squaring and multiplication operation are then implemented in digit-serial manner using the architecture depicted in Figure 4-28. The operands are digit-word inputs. Thus, the inputs bit a_i, e_i and p_i for $0 \le i \le m-1$ in Figure 4-25, are replaced by digits forms A_i, P_i, and E_i for where

$$(A_i, P_i, E_i) = \begin{cases} \sum_{j=0}^{D-1} (a_{Di+j}, p_{Di+j}, e_{Di+j}) \cdot x^j, & 0 \le i < d-1 \\ \sum_{j=0}^{m-1-D(d-1)} (a_{Di+j}, p_{Di+j}, e_{Di+j}) \cdot x^j, & i = d-1 \end{cases} \quad (4\text{-}24)$$

and

$$(A, P, E) = \sum_{i=0}^{m-1} (a_i, p_i, e_i) \cdot x^i = \sum_{i=0}^{d-1} (A_i, P_i, E_i) \cdot x^{Di} \quad (4\text{-}25)$$

The exponent E is expressed in binary form and fed into a LIFO stack (buffer) of size $\lceil m/D \rceil D$. The input digit-words A_i, P_i, and E_i, are fed into the multiplier in the same order for i decreasing

and from the MSB to the LSB. If m is not divisible per D, then a zero padding is performed at the LSB positions for i=0 for A_i and P_i and MSB positions for i=d-1 for the exponent E_i. The s signal is used to denote the start of the multiplication.

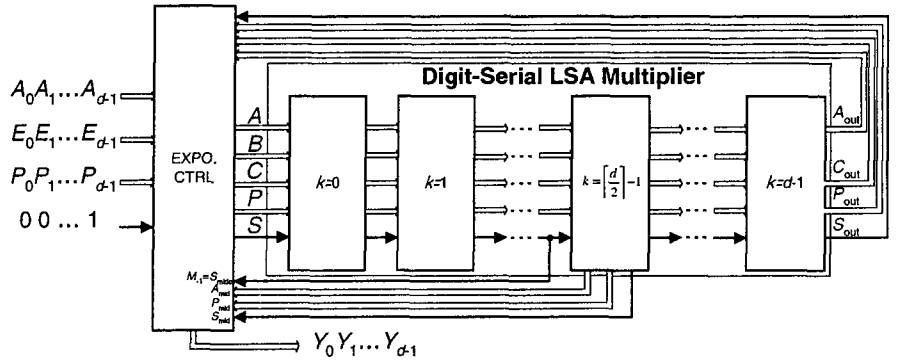


Figure 4-28: Digit-Serial LSA multiplier based exponentiator

The exponentiator operates as the previously described bit-serial LSA based, where each CELL-k of the multiplier process one digit of an entire word in one clock-cycle. Note that the architecture shown in Figure 4-28, is still programmable with respect to the primitive polynomial $p(x)$.

At d clock cycles after the least significant digit-words A_0, E_0, P_0 and the LSB of s enter the exponentiator, the result Y_i will start coming out after $2m \cdot d$ at the rate of one digit every clock cycle. Thus, the exponentiation time T_e takes $2(m+1)d$ clock cycles.

4.5.3 IMPLEMENTATION RESULTS AND COMPARISON

The clock free and clock gating LSA exponentiators have been implemented at the gate level using different digit-size D=1,4,8 in order to perform a comparison in terms of delay, area and energy consumption for one exponentiation over $GF(2^{607})$ with $p(x)=1+x^{273}+x^{607}$ as primitive polynomial. These architectures are mapped on TSMC CMOS 0.18μm technology at a targeted supply of V_{DD}=1.8v and 0.9v. Different types of power dissipation components are estimated using gate level simulations on a set of random stimulus in order to compute the energy and energy-delay product. The performance characteristics including total delay, area in term of gates and energy-delay product are reported in Figure 4-29, Figure 4-30, and Figure 4-33 respectively. The cost function is reported in Figure 4-34 and defined as the Energy-Delay-Area products versus the digit-size. The power numbers of different circuits are reported in Appendix C.2.

On the one hand, the critical path of the architecture shown in Figure 4-25 is just the sum of one full-adder and one NAND gate delay. For larger digit-size the critical path is limited by the F function shown in Figure 4-18 and increases linearly with the digit-size as $(D\text{-}1)(T_{XOR\text{-}3}+T_{NAND\text{-}2})$. In the same time, the latency decreases linearly with the digit-size in almost the same rate resulting in almost a constant total delay as shown in Figure 4-29.

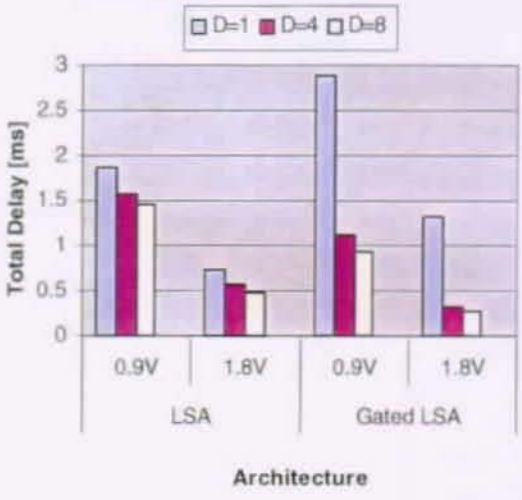


Figure 4-29: Total Delay as function of the digit size for one LSA digit-serial GF(2^{607}) exponentiation

On the other hand, Figure 4-30 shows that the area increases dramatically with the digit-size due to the large number of logical gates and latches used to temporary hold the internal and the output data in master-slave manner. This means that the level of parallelism is limited by the area constraints (digit-size).

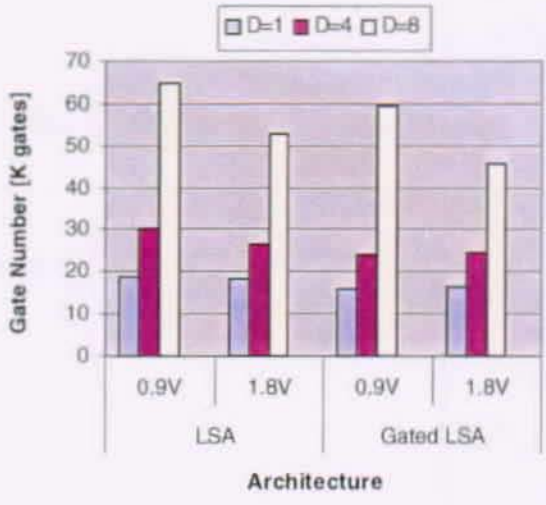


Figure 4-30: Area in gates of the LSA digit-serial GF(2^{607}) exponentiator as function of the digit size

Clock gating technique achieves a substantial reduction in both the total delay and number of gates (Figure 4-30), for digit-size equal or larger then 4. It helps to eliminate the feedback loops and multiplexers used to feed the output of each internal storage elements back to the input for synchronous load-enable. Such feedback loops are replaced by only one integrated cell (latch based clock gating cell). Thus the area is reduced in all cases but for the bit-serial architecture (D=1) these clock gating cells (latches) add a substantial delays which increase the critical path.

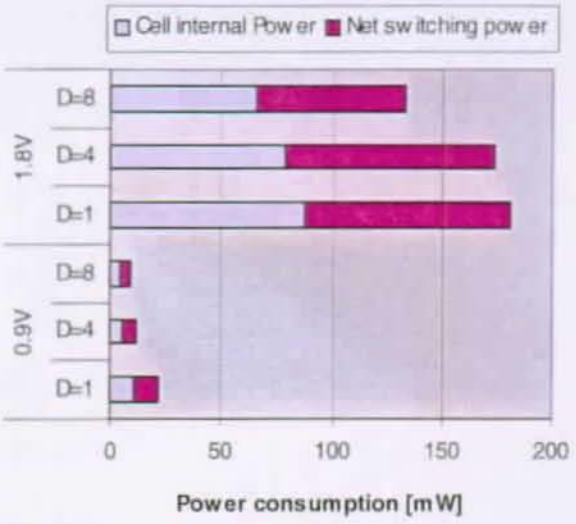


Figure 4-31: Different types of power dissipation components as function of the digit size of the LSA digit-serial $GF(2^{607})$ exponentiator

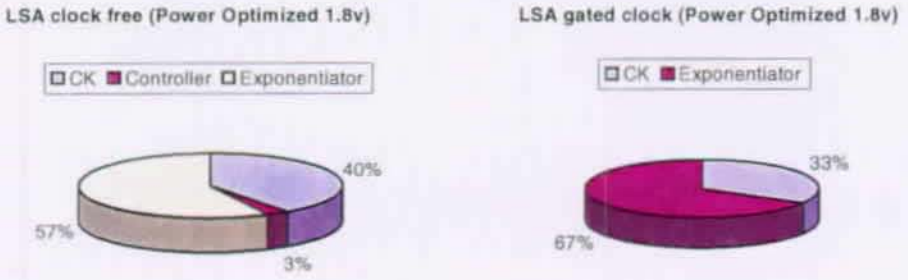


Figure 4-32: Power profile for LSA digit-serial (D=4) exponentiator (1.8v)

If the switching power is dominant then the power consumption can be reduced by factor β^2 when reducing the operating voltage by a factor β Equation 2-1. From Equation 2-2 the delay is increased by factor β assuming that $V_{DD} >> V_t$. Therefore, the power saving is counterbalanced by the increased delay since the energy-delay product is proportional to δ^2. This is not the case for V_{DD}=1.8V and β=2 since V_{DD} is close to the threshold voltage V_t. Hence the delay increases and the short circuit power is no longer a negligible factor (switching power no longer the dominant factor) as shown in Figure 4-31.

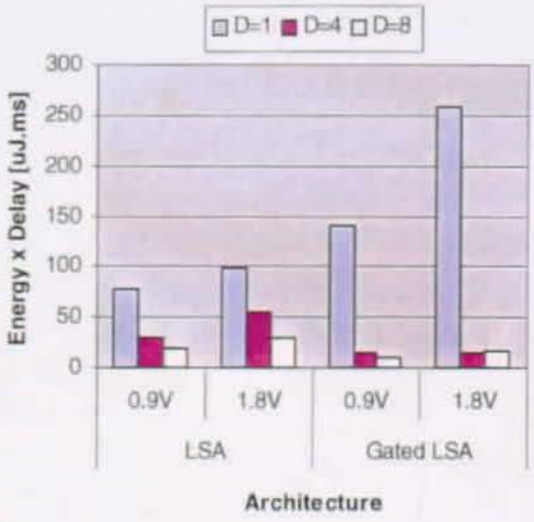


Figure 4-33: Energy-Delay product as function of digit-size of the LSA digit-serial GF(2^{607}) exponentiator

The most interesting result is obtained when comparing the Energy-Delay and the cost function versus the digit-size. The performance characteristic reported in Figure 4-33, shows that Energy-Delay products are significantly reduced when digit-size increase. High gain is obtained for D=8 when operating at 0.9v and more than 75% reduction is noticed (more than 92% when gating the clock) since the power is significantly reduced when operating at lower voltage. However, when comparing the characteristic reported in Figure 4-33 (cost function), the optimum is obtained for D=4 when operating at both 1.8v and 0.9v due to the dramatic increase in circuit area for larger digit-size. The highest gain is obtained when gating the clock as the number of gates and delay are substantially reduced.

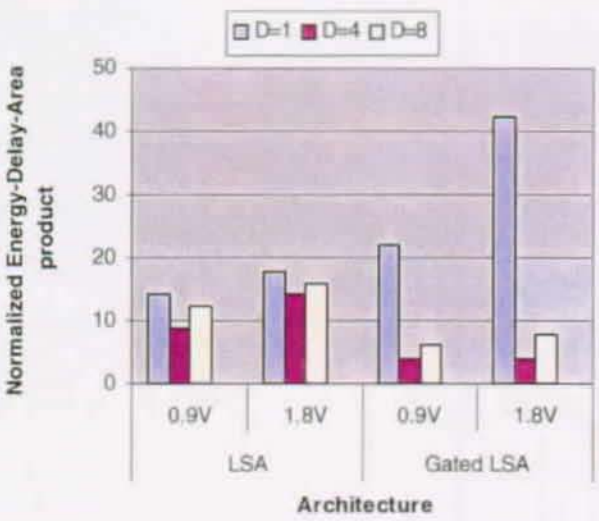


Figure 4-34: Energy-Delay-Area product as function of the digit size of the LSA digit-serial GF(2^{607}) exponentiator

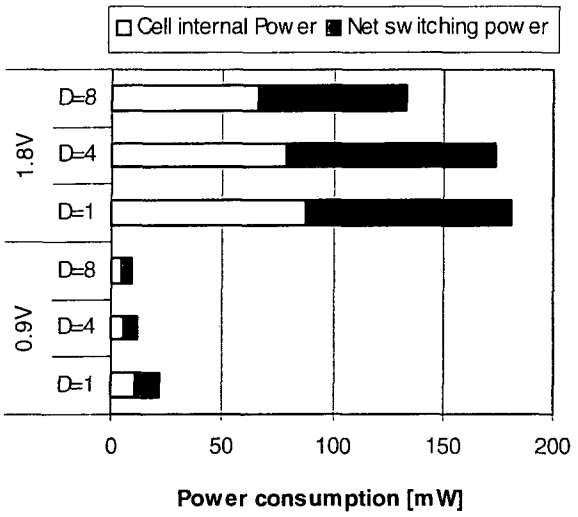


Figure 4-31: Different types of power dissipation components as function of the digit size of the LSA digit-serial GF(2^{607}) exponentiator

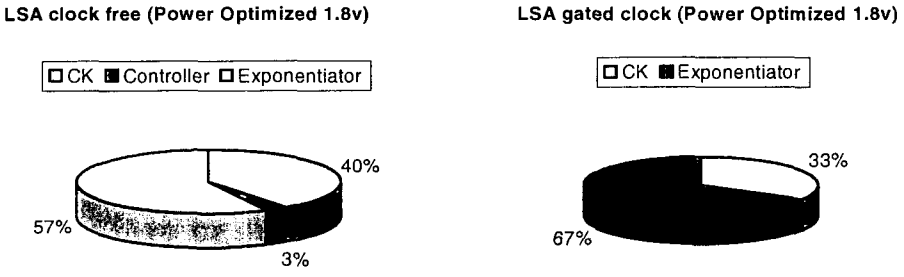


Figure 4-32: Power profile for LSA digit-serial (D=4) exponentiator (1.8v)

If the switching power is dominant then the power consumption can be reduced by factor β^2 when reducing the operating voltage by a factor β Equation 2-1. From Equation 2-2 the delay is increased by factor β assuming that $V_{DD} >> V_t$ Therefore, the power saving is counterbalanced by the increased delay since the energy-delay product is proportional to δ^2 . This is not the case for V_{DD}=1.8V and β =2 since V_{DD} is close to the threshold voltage V_t . Hence the delay increases and the short circuit power is no longer a negligible factor (switching power no longer the dominant factor) as shown in Figure 4-31.

4.6 PHYSICAL DESIGN: LSA EXPONENTIATOR (PLACEMENT AND ROUTING)

Digit-serial LSA exponentiator with D=4 has been designed and synthesized in a 0.18µ m 6LM technology with a 100MHz working frequency. We were able to place and route the design with high density achievable. First result on the placed and routed core gives about 0.994mm^2 chip area including the Die boundary and I/O pins (the overhead of area utilization of all cells is 91.64%). Detailed implementation results are shown at Table 4-8. It shows that about 89% of area is consumed by cells placement and routing. Other 11% is used by the controller. A layout of the core is shown on Figure 4-35 where different components are highlighted.

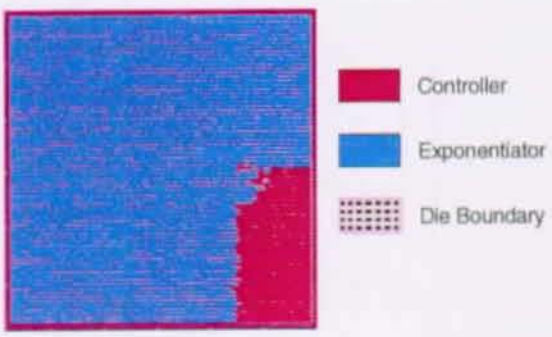


Figure 4-35: Digit-serial (D=4) LSA Exponentiator Layout (0.18µ m,6ML)

The impact on the timing delay is minimal since the hole architecture is pipelined and present a short critical path, nevertheless the timing delay after physical back-annotation is increased to 2.99ns

Unit	Area (mm^2)	Overhead (%)
Exponentiator	0.098	10.8
Controller	0.813	89.2
Total area	0.911	100

Table 4-8: Area overhead: digit-serial (D=4) LSA Exponentiator core implementation

CHAPTER 5

5 SELF-SYNCHRONIZING STREAM CIPHERS (SSSC)

The generation of the keystream can be independent of the plaintext and ciphertext yielding what is termed a synchronous stream cipher or it can depend on the data and its encryption, in which case the stream cipher is said to be self-synchronizing. Most stream cipher designs are for synchronous stream ciphers. In this chapter we propose an engineering oriented approach for the design of self-synchronizing stream cipher SSSC. This approach appears to be useful in practical design. The interesting ideas are more related to the way to combine strong building blocks while we address the problem of designing these blocks themselves. An actual design called Self-Synchronizing Mixture Generator (SSMG) is presented and claimed to be fast, cryptographicaly secure and easily implementable in hardware.

5.1 SSSC DESIGN PARAMETERS

Stream encryption is performed by encrypting the plaintext digit by digit (or bit by bit for binary stream ciphers). Let $X = x_1, x_2, \ldots$, $Y = y_1, y_2, \ldots$ and $Z = z_1, z_2, \ldots$ denotes the binary plaintext, ciphertext and keystream sequences respectively, and let K denote the secret key that is chosen randomly from the set κ of possible keys (key space). The encryption transformation is denoted by $f_e(x_i, z_i)$ and must be invertible with respect to x_i,

$$\begin{aligned} y_i &= f_e(x_i, z_i) \\ x_i &= f_d(y_i, z_i) \end{aligned} \tag{5-26}$$

For the majority of designs the digits are bits. In this case, Equation 5-26 can be reduced to XORing operation or a simple additive combiner. The keystream digits z_i are generated independently of the message stream by a pseudo-random sequence generator (PSG) using the secret key K as described in Section 3.4.2. This is a cryptographic finite state machine whose operation is governed by the two rules:

$$\begin{aligned} S_i &= f_s(S_{i-1}), S_{-0} = IV = f_{init}(K) \\ z_i &= f_o = (S_{i-1}) \end{aligned} \tag{5-27}$$

Where f_s is the state transition function, f_o the output function and f_{init} is the initialisation function. The finite state machine is initialised by loading the initialisation vector IV into the finite state machine. Generally, the initialization procedure is performed by resetting the internal state to a fixed value and substantially iterating the finite state machine a specified number of times. During pseudo-random sequence generation the state transition can be modelled by the function f_s. The security of the stream cipher can be based entirely on the secrecy of (part of) the initial state S_i.

This secret information is generally referred to as the key K can be expressed by a function f_{init}. In this thesis the specification of a PSG consists by definition of the three functions f_s, f_o, f_{init}.

5.1.1 CIPHER FEEDBACK APPROACH

Due to man-made noise, fading and temporary loss of contact, errors occur in bursts and receiver may lost synchronization needed for synchronous encryption as the encryptor and decryptor may have different initial state (continuous state machine). A resynchronization state is required at the time of resynchronization. One solution is to specify the new internal state in terms of information that is only known to the sender and receiver. In this case the sender and receiver calculate the new internal state from the original internal state (or key) and a public parameter p (i.e. the time at the moment of resynchronization,....,etc). The most obvious solution consist of adding some extra bits for message blocks/frames/packet identification.

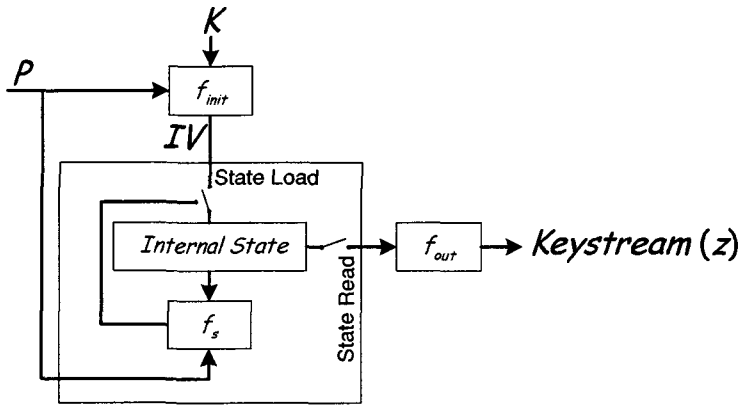


Figure 5-36: Cryptographic state machine: implementation of an SSSC

For SSSC every keystream digit z_i depends not only on the secret (N-bit) key K but also on a public variable that is known to both the sender and the receiver, as shown in Figure 5-36. This mechanism makes it possible to change the session key regularly without introducing new key material. Thus, the cryptographic finite state machine become governed by the two rules:

$$\begin{aligned} S_i &= f_s(S_{i-1}, p), S_{-0} = IV = f_{init}(K, p) \\ z_i &= f_o(S_{i-1}) \end{aligned} \tag{5-28}$$

The most widely adopted approach to self-synchronizing stream encryption is to use a fixed number M of previous ciphertext digits as a public parameter p also can be called *Cipher Feedback* approach analogously to Cipher Feedback (CFB) mode used in block ciphers. One can notice that a block cipher in CFB mode is self-synchronous.

In this case z_i can be calculated by evaluating a publicly known boolean function f_c with $N+M$ input bits. This function is called the *canonical function* of the SSSC and is denoted by:

$$z_i = f_c(y_{i-1}, y_{i-2}, \ldots, y_{i-M}, K) \tag{5-29}$$

By reporting the change to Equation 5-28, we can express the SSSC's CFSM's state transition as follows:

$$\begin{aligned} Si &= f_s(y_{i-1}, y_{i-2}, \ldots, y_{i-M}, K), S_{-0} = IV = f_{init}(K) \\ z_i &= f_o(S_{i-1}) \end{aligned} \quad (5\text{-}30)$$

The initial vector IV can be generated from the secret key in case where the finite state machine is an LFSR based autonomous machine since in general case the initial state is key dependent.

5.1.2 FAULT TOLERANCE

From Equation 5-30 it is clear that decryption is performed correctly if the last M received ciphertext bits have been correctly received. A single bit error y_{i-j} on the channel between encryption and decryption gives rise to an burst error of length j until the bit y_{i-j} is shifted out of the canonical function. Thus when bit error occurs, the next j bits cannot be correctly decrypted at the receiver. So, every bit error in the ciphertext results in an error burst in the deciphered plaintext. Therefore, the SSSCs which use this technique are suitable for application with low bit error rate but for which bit synchronization is required for correct decryption.

5.2 FRAME BASED SSSC (SSMG PRINCIPLE)

In this section we describe our attempt to design an SSSC. The amount of data to be encrypted is subdivided into frames that form the frame space denoted by Φ. The purpose is to encrypt/decrypt each frame φ with a different key (key session). The key is calculated for each frame from the master key using the fingerprint (keyed hash value) of the previous encrypted frame ciphertext.

The solution called Self-Synchronizing Mixture Generator SSMG is a frame based SSSC. The cipher here is a stream cipher and processes one bit or byte at the time (each frame is encrypted bit per bit or byte by byte) and the hash value is computed for the entire frame. Figure 5-37 and 5-24 gives an overview of the encryption/decryption process respectively using the SSMG.

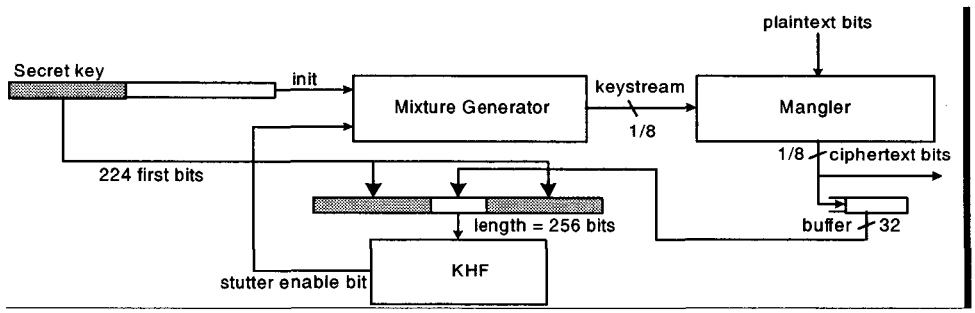


Figure 5-37: SSMG stream cipher encryption

During encryption the *Mixture Generator* (a CFSM) is initialized with the secret key. The first M-bit length frame is encrypted in the *Mangler* (a combiner) using the keystream output of the SSMG. The ciphertext is then buffered up to words of 32-bits length that are appended to the secret key in enveloped method fashion (Section 3.4.5.2) and hashed using a chaining transformations handled by a KHF (Section 3.4.5). The hashing operation can be performed in parallel with the encryp-

tion process. The hash result is then used as an additional parameter to the secret key (synchronization pattern) for Mixture Generator re-keying before processing the encryption of the next frame.

During Decryption the received ciphertext bits are buffered and hashed in the same manner. The Mixture Generator is initialized using the hash results and the secret key before processing the next frame.

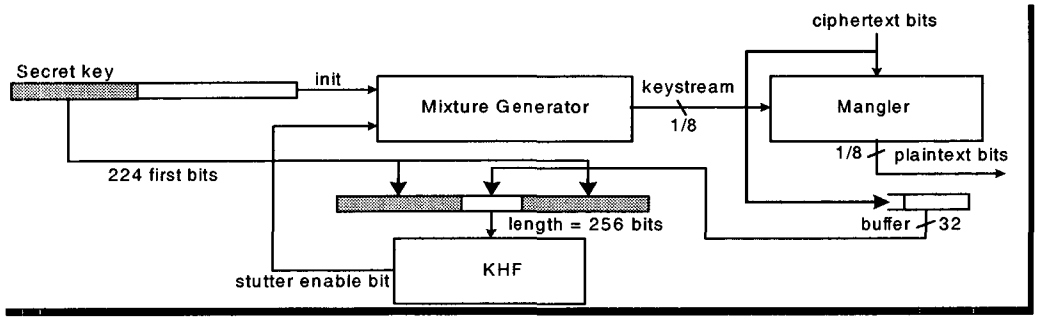


Figure 5-38: SSMG stream cipher decryption

5.2.1 MIXTURE GENERATOR MG

The mixture generator is realized as an autonomous finite-state machine based on the Alternating Stop and Go scheme reported in Figure 3-8a (see Section 3.4.2.4), whose initial state and structure as well depend on the secret key *K*. In what follows, LFSR-1 is called Mixer, LFSR-2 is the Top and LFSR-3 the Bottom. Both the Top and Bottom are clock-controlled LFSRs. The secret key is the Mixer initial state, which controls the clock generator. Each LFSR is based on a recurrence relation over the Galois extension Field $GF(2^m)$ controlled by the field generating-polynomial (see Appendix A.2), i.g., $p(x)=1+x^k+x^m$ for trinomials.

The result on the period and the linear complexity of the Alternating step generator (see Table 3-4) are best compared to these results obtained on the most concurrent generators. The best known attack on the Alternating Stop and Go is a divide-and- conquer attack on the Mixer control register, which takes approximately 2^l steps (assuming a common length l for the three LFSRs). Therefore, LFSRs should be of maximum-length. Those lengths must be pair-wise relatively prime [68]. Moreover, there are certain values of m and k for which the period of the LFSR is the maximum possible (see Section 4.3), namely 2^m-1, which is all the possible states of m bits, excluding the all-zero state.

The pairs m, k that define maximal period LFSRs can be chosen from Table 4-5. Further, the LFSRs lengths are chosen to be $L_{MIXER}>L_{TOP}>L_{BOTTOM}$ in order to optimize the hardware.

5.2.1.1 BIT AND BYTE GRANULARITY

The mixture generator delivers its output in 2 ways as a bit or byte (*granularity*). Figure 5-39 shows a trinomial LFSR performing a byte granularity by folding the bit granularity architecture 8 times and looking ahead for the calculation of the next output bits. For byte granularity the Mixer is stepped or shifted one time wile both the Bottom and Top are shifted 8 times. Thus, the result of requesting 4 bytes for example, is not necessarily the same as the result of requesting 32 bits, because of differences in the way that the LFSRs are stepped.

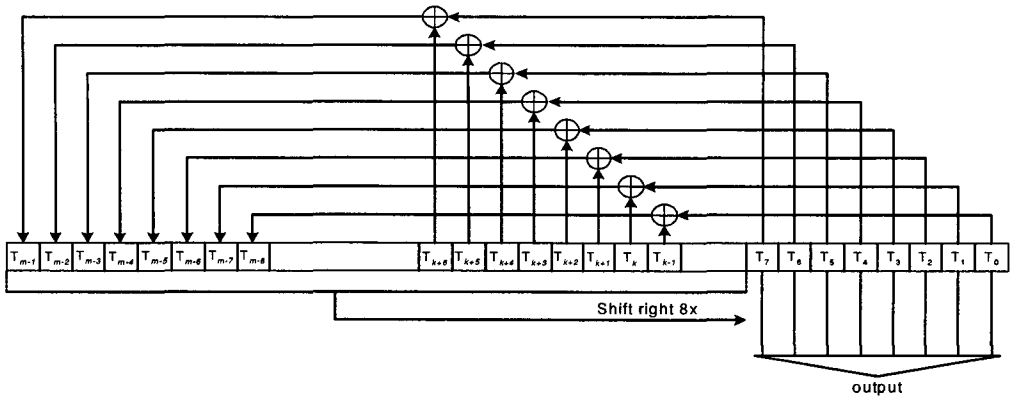


Figure 5-39: Byte Granularity: trinomial LFSR

The different LFSRs steps for the generation of keystream output give rise to different bit-rates when working in bit or byte granularity. Figure 5-40 shows an improved Bottom LFSR architecture able to compute 8 bits output with bit granularity in one rather than 8 clock cycles. The same architecture is applied for the Top LFST in order decrease the total delay (latency) corresponding to the delay of packet processing using bit granularity.

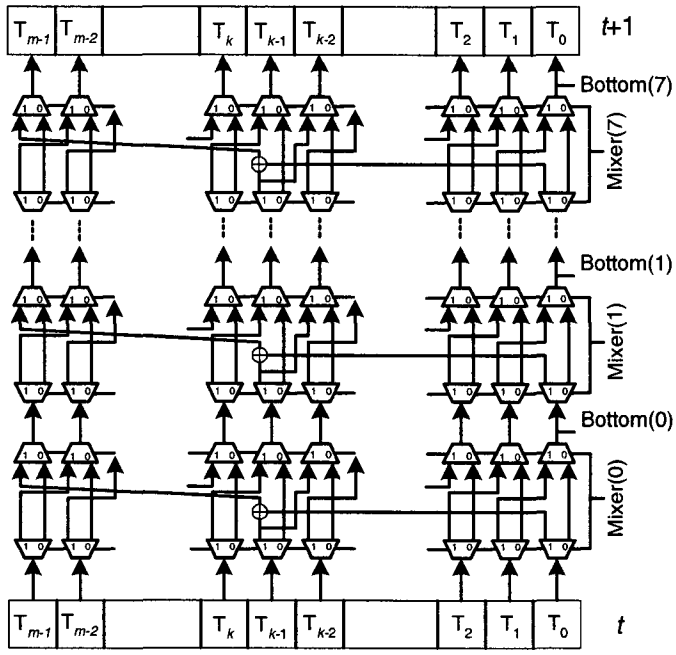


Figure 5-40: Bit Granularity: Speeding up the Bottom LFSR

5.2.1.2 MG STUTTERING

The complexity of the SSMG's keystream output is increased using *stuttering*. The LFSRs of the MG are stepped irregularly in an attempt to break up their linearity while maintaining good statistical properties. Another purpose consists of using the stuttering for SSMG's re-keying between frames processing (clocking the LFSRs in an unpredictable number of times between outputs or re-keying the generator using different internal state generated from the secret key). If the state bits (of the corresponding LFSR) selected by the stutter mask pattern are all zero, then the next skip count bits of the shift register are discarded. The stuttering works differently during *pre-mixing* (in-

itialization) operation than during the encryption operation. During initialization phase the stutter uses a stutter mask given by the CellHash function (the hash result) or *stutter enable bit* in Figure 5-41 and during encryption operation it uses a fix mask value which is generator's state-dependent (bit S1and S2).

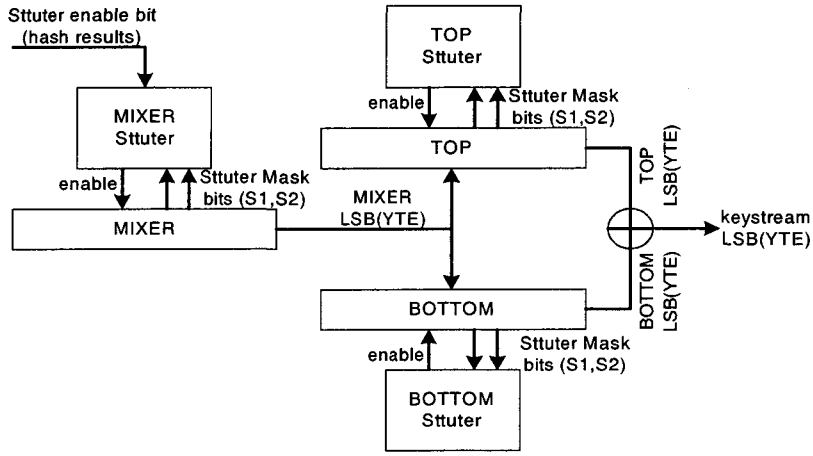


Figure 5-41: Mixture Generator Stuttering scheme

5.2.1.3 MG INITIALIZATION

During initialization, the secret key is loaded into the Mixer before the processing of each subsequent frame. The Top and Bottom are both loaded with a state containing bit 1 at position k, all other bits are 0, where k is the feedback parameter of the trinomial field generating polynomial of the corresponding LFSR. The Mixer is stepped irregularly using the *stutter enable bit* from the KHF. After initialization the 3 LFSRs are stepped using the fixed mask stutter to generate the keystream output.

An output buffer is added to the Mixture Generator in order to increase run-time performance in software implementation and reduce the KHF internal operation (1 iteration for each block of 4 bytes). The mixture generator is able to work in bit or byte granularity, the result in performance and the resulting ciphertext will be different for each case.

5.2.2 MANGLER

The mangling operation is a simple bit-wise XOR between the key stream and the plaintext.

5.2.3 KEYED HASH FUNCTION (KHF)

The KHF is used to obtain a fingerprint of the encrypted frame block. This fingerprint is then used as stutter mask during MG initialization before the next frame to be encrypted. The input to the hash function is a message of the form $M=m_0||m_1...||m_t$, with every m_i a 32 bit ciphertext. According to the requirements for keyed hash function (see Section 3.4.5.1) and the new construction goals defined in Section 3.4.5.2, two sub-keys K_1, K_2 derived from the secret key K are used with each message m_i. The message block m_i is prepended with key K_1 and appended with key K_2 before the hashing operation: envelope method. The hash value H_i is the result of applying the hash function f_h to $K_1||m_i||K_2$, according to Equation 3-4:

$$H_i = f_h(H_{i-1}, K_1 \| m_i \| K_2), \quad \text{for } 0 \le i \le t \tag{5-31}$$

H_0 is equal to IV, a fixed initial value. The hash value of the entire message is given by: KHF(M)=H_t. Here the output function is selected to be the output state buffer. Figure 5-28 shows the structure of the adopted KHF. A hash result is processed on the base of the previous state buffer value and a message that is passed as parameter

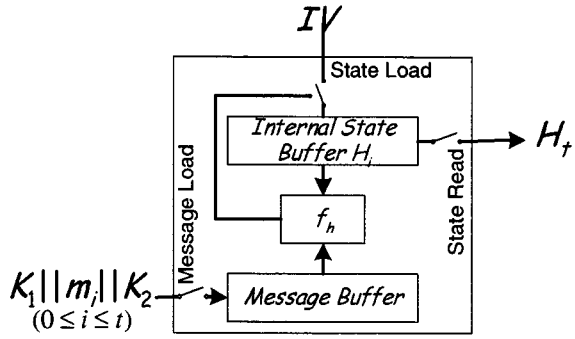


Figure 5-42: KHF structure

5.2.3.1 IMPLEMENTATION OF THE HASH FUNCTION

The function f_h is chosen to perform the authentication of large amounts of data or achieving high speed if implemented in hardware.In [54] Joan Daemen proposes a generic model, which consist of the compression function f_h with a fixed-length input based on an invertible chaining transformation. He presented a concrete proposal named *Cellhash*. With respect to Cellhash the core function is not modified but the functionality is updated to build a KHF according to Figure 5-29. The core of this function is formed by two cellular automata operations and a permutation. The computation of this function is done in 5 steps as described below:

- *Step* 1: $h_j = h_j \oplus (h_{j+1} \vee \overline{h_{j+2}})$ non-linear cellular automata
- *Step* 2: $h_0 = \overline{h_o}$ complementary operation
- *Step* 3: $h_j = h_{j-3} \oplus h_j \oplus h_{j+3}$ linear cellular automata
- *Step* 4: $h_j = h_j \oplus m_{j-1}$ message injection
- *Step* 5: $h_j = h_{10xj}$ bit permutation

All computation are performed modulo 257.

These steps are used for updating the internal state of the KHF state machine at each iteration. In *Step* 1 each bit value H_i is updated according to the bits in its neighbourhood. This is invertible operation if the length of H_i is odd. *Step* 2 consists of complementing 1 bit to eliminate circular symmetry in case all state-bits are 0. The linear CA operation in *Step* 3 is invertible if the length of H_i in not a multiple of 7 or 31. *Step* 4 consists of message injection and in *Step* 5, bits are placed away from their previous neighbours. The length of H_i is then chosen to be the prime 257 to make *Step* 1 and 3 invertible and to prevent a birthday attack (size has to be at least a prime above 128 bits). Through the different steps of the function a good level of confusion and diffusion is achieved, circular symmetric patterns are avoided in H and the message bits are injected and diffused in the result.

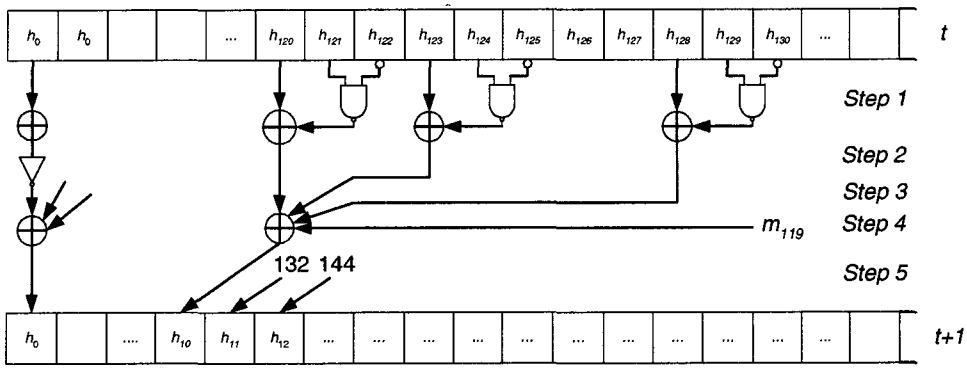


Figure 5-43: Cellhash structure

The state buffer H_i in this case is of length 257 bits. In our implementation the keys K_1 and K_2 are made of the first 224 bits of the secret key K which constraint the limit size of the secret key and hence the Mixer length to the minimum of 224 bits. The initial value IV is always zero and the number of rounds depends of the size of the ciphertext block.

5.2.3.2 DIFFUSION AND CONFUSION

The function f_h has to guarantee diffusion of information. With equal keys, two inputs H_{i-1} and H'_{i-1} that differ in only a few bits must give rise to two outputs H_j and H'_j differing in *substantially* more bits. The hash result must depend on the message bits in an involved and complicated way to achieve a good confusion.

As mentioned in [60] and shown in Figure 5-43, each bit of the state register depends on 9 bits of its predecessor state at the fist iteration. After the second iteration each state bit becomes dependent on 81 bits of the predecessor state. After three iterations the dependence is complete. Further, the key-dependence of the input massage bits maximizes the diffusion of both the message and key bits into the hash results.

Studying the confusion deal with analysing the propagation of differences in the input to intermediate values (differential cryptanalysis). It is shown in [60] that the contribution of *step* 2-5 to the confusion becomes clear only when characteristics over multiple iterations are considered and that because of total diffusion of the compression function f_h, calculations involving bits from internal states separated by several iterations become extremely complicated even if a small number of iterations is considered.

5.2.3.3 COLLISION FREE

For cryptographic purposes the CHF must be collision free. This implies that finding two different messages M_1 and M_2 that have the same hash result must require a computational effort of the order of $2^n/2$ applications of the hash function. In Section 3.4.4, we shows that generating a collision for such a CHF involves either generating a collision for f_h or solving a problem with comparable complexity. However, because of the uncertainty whether generating a collision is in fact a hard problem, it is not claimed here that finding a collision for the proposed hash function is equivalent to any other hard problem than finding a collision for the function.

5.3 SSMG: OPERATIONS OF THE CIPHER

The SSMG stream cipher switches in between two states (modes of operation): *Pre-combining* and Encryption/Decryption

Pre-combining consists of loading the Mixer with the secret key, then clocking the mixture generator until the complete content of the Mixer has been passed through the TOP and BOTTOM LFSRs for the purpose of initialization. This operation is controlled with the stuttering bit of the MIXER. The stuttering bits are the state bits H_f of the KHF corresponding to the keyed hash value of the predecessor encrypted/decrypted frame's finger-print or *digest*. The first time the mixture generator is pre-combined no stuttering takes place for encrypting/decrypting the first frame. When the pre-combining operation takes place, the KHF is reset before encrypting/decrypting each new frame. The stuttering is dependant of the complete previous ciphertext frames. Thus, when starting the encryption of a new frame block, the state of the mixture generator becomes only dependant of the last ciphertext block.

Once the Mixture generator is initialized, the current frame can be processed and authenticated in the same time in a parallel fashion in order to avoid the additional latency of the KHF at the pre-combining process. Thus, reducing the total latency due to the pre-combining operation. During encryption the stuttering takes place on the basis of a fix stutter mask. The message is injected to the KHF by block of 32 bits in order to achieve an optimum number of iterations.

Figure 5-44 shows the inter-frame dependencies during SSMG encryption and decryption.

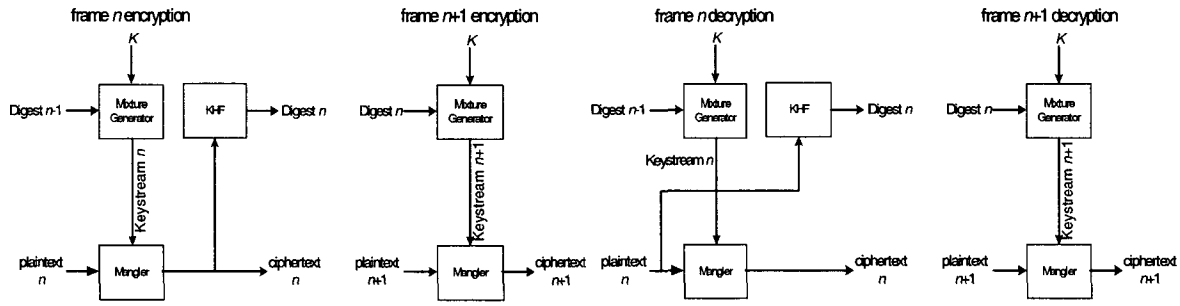


Figure 5-44: SSMG operation: inter-frame dependencies during encryption and decryption

5.4 SSMG: CRYPTOGRAPHIC SECURITY

In some applications the actual internal state of the MG is calculated from a secret key using a publicly known keyed randomization function and a public variable ciphertext. This mechanism makes it possible to change the internal state regularly without introducing new key material. Although this technique is used to improve the security of the system, the additional freedom given to the cryptanalyst can seriously weaken the security. The knowledge of the relation between the secret key and the internal state can be exploited to obtain secret key information. This is can be translated to the fact that the function combining State-transition (KHF) and Keystream Generator (MG) can not approximated by a boolean function which is a necessary condition for an SSSC to be secure according to Maurer [99]. Though, some SSSC criteria are discussed here under the SSMG scheme, mainly repeated occurrences and dependencies between SSMG blocks sizes.

5.4.1 REPEATED OCCURRENCES

A necessary condition for an SSSC to be secure is that both the key space κ and the frame space Φ must be sufficiently large such that the probability is negligible that a length M in Equation 5-30 or t in Equation 5-31 ciphertext pattern is repeated before the secret key is changed. In our case this can be translated to the probability that two frames φ_1 and φ_2 or more contain the same information, which implies the use of the same initial state of the MG. On the other hand, the length of the Mixer L_{MIXER} should be large in order to increase the internal state space. When M (frame size) is large this probability can be very small but because the bit error on the channel propagate over the following deciphered plaintext frame (Section 5.1.2), the frame size should be as small as possible.

In fact, when a cryptanalyst can observe two occurrences of the same length (same frames) ciphertext pattern he is able to compute the XOR (modulo-2 sum) of the corresponding two ciphertext frames (XORing two plaintext bits is equal to the XOR of 2 corresponding ciphertext bits which, gives a way one bit of plaintext information).

This fact is more related to the nature of the information carried in the ciphertext frames (same frames=same ciphertext=same digest=same internal state) or to collision of the f_{ij} reported in Figure 5-43 (collision=same digest=same internal state). Whatever, a collision may occur or frames are repeated (which can reveal some information about the plaintext), since the internal state of the mixture generator is ciphertext's digest dependent and is frequently changed any information on the secret key involve a huge computational effort either on the KHF and the MG due to dependencies of both blocks on the secret key and the nature of one way functions of these blocks.

5.4.2 PERIOD OF THE SSCS AND HASH RESULT SIZE

Although a synchronous stream cipher is insecure unless the period and the linear complexity of the keystream sequence are sufficiently large (security criteria for synchronous stream cipher), this cannot be applied to a traditional SSSC since there exists no sequence in the encryption process that is independent of the plaintext. This is more related to the key space: is the key space as big as the number of frames to be encrypted (plaintext size) principle of one time pad (see Figure 3.4.2)?. This fact is dependent on the size of the hash value, which defines the probability of secret key occurrence in key space.

On one hand, the linear complexity of MG must be increased to avoid repeated occurrences and the use of Berklamb-Massey Algorithm (Section 3.4.2.4). This means that the LFSRs size of the MG must be large enough. On the other hand, the need of large keystream output is limited by the size of the frame and since the MG is re-keyed for each frame, the MG LFSRs sizes may be reduced to an acceptable range. Other than the frame size, the hash result size play an important role in increasing the MG internal state space and reduce the probability of repeated recurrences. Consequently, beside the KHF collisions, there exist a trade-off between the sizes of MG LFSRs, the frame's size the size hash result. As stated before, although the frame size should be as small as possible in order to reduce the burst errors, it is important to notice that the frame's size and the hash result's size are no longer independent.

CHAPTER 6

6 SECURING MPEG-2 STREAM IN DVB

Digital techniques have made rapid progress in audio and video for a number of reasons. A digital signal allows more data to be transmitted within a smaller bandwidth, allows greater consistency, data reliability and stronger encryption of data. MPEG compression is already being used in broadcasting and will become increasingly important in the future with the emergence of Digital Video Broadcasting (DVB) standards. A DVB system is build upon MPEG-2 and uses MPEG-2 transmission. It also defines additional private sections and providing a definition of the physical medium (modulation, coding, etc.) needed to carry the MPEG-2 Transport Stream (TS) [109]. On the other hand, some factors such as atmospheric noise, multi-path propagation, signal fades and transmitter non-linearities may create received bit errors in a DVB system. Since, Forward Error Correction (FEC) layers can detect and correct these errors, up to a reasonable limit, a DVB system seems to be a suitable application for our SSMG.

The fist part (Section 6.1) of this chapter is focused on the performance study of MPEG-2 stream encryption using the AES (Advanced Encryption Standard) and our SSMG. Their impact on real-time streaming is analysed using ciphers performance profiles in the frame of selective encryption. This encryption method consists of encrypting small chosen segments of bits from an MPEG video stream. The second part (Section 6.2) is consecrated to the hardware implementation of our PK-SSMG scheme for securing MPEG-2 TS in DVB application. PK-SSMG is a hybrid cryptosystem including LSA digit-serial exponentiator for key generation and SSMG for stream encryption. Some design considerations have been undertaken in order to support the DVB-SPI interface under resynchronization, i.g., reducing the latency by buffering. Implementation issues and results are detailed.

6.1 MPEG-2 STREAM SOFTWARE ENCRYPTION

Different studies have been made on the way to most efficiently encrypt MPEG-2 data stream. Most of the encryption schemes present either the withdraw that they are not compatible with the MPEG-2 standard [108] such as SECMPEG [107], which requires major changes to MPEG-2 codec. Some of the studies such as Zig-zag permutation algorithm [102] add overhead to the MPEG stream by rendering the compression less effective. The following schemes have been published: Video Encryption Algorithm VEA [104], Pure Permutation Algorithm [105] (this list might not be exhaustive). A comparison is made in [106] for the evaluation of these algorithms with respect to performances (encryption speed) and security level. The authors show that these algorithms are almost vulnerable to some known attacks such as known-plaintext and known-ciphertext attacks (see Section 3.3.1.1).

We can split the MPEG confidentiality schemes into two categories:

- Systems where encryption is processed after MPEG compression; decryption is processed before MPEG decompression. In these systems data is treated as a bit-stream and the entire stream is encrypted. This method doesn't use any special of MPEG-2 structure, adds latency to real time video delivering and involves heavy computations when software solution is adopted but it provides the most secure MPEG bit-stream.
- The second approach consists of encrypting small chosen segments of bits from an MPEG-2 video stream. The original motivation is to reduce the computation time in the MPEG decoding process, without compromising the security too much. Maples and Spanos in [100] performed a real-time software encryption using selective encryption of the I-frames (see section 6.1.2) using DES in CBC mode. In [101][102][103] many selective encryption schemes are proposed using DES to reduce the number of encrypted bits.

The MPEG part of this section is based on the MPEG-2 software package of Felix Wuy and Tsung-Li Wu [101]. Where a selective schemes has been implemented using a DES cipher.

6.1.1 CRYPTOGRAPHIC PRIMITIVES

A number of applications use software encryption in order to provide an acceptable security level at a low cost. An important constraint is that the performance of the application should be influenced as little as possible by the introduction of cryptography. The performance of a cryptographic algorithm is function of the efficiency of the algorithm (security level) and the speed of its execution time. The stream/block ciphers that are used in this section are listed in Table 6-9.

No	Cipher	Key length 0	Key length 1	Key length 2
1	DES[1]	56	56	56
2	3DES[23]	168	168	168
3	MARS[108]	128	192	256
4	RC6[109]	128	192	256
5	Rijndael[110]	128	192	256
6	TwoFish[112]	128	192	256
7	Serpent[111]	128	192	256
8	KASUMI[114]	128	128	128
9	IDEA[113]	128	128	128
0	SEAL[115]	160	160	160
A	RC4[116]	256	256	256
Z	None	-	-	-

Table 6-9: Configuration Values of the Encryption Interface

Almost all these ciphers have been enough cryptanalyzed and their analysis results are reported on the related literature. We aim to compare the speed performance of these algorithms using MPEG-2 selective encryption.

6.1.2 STRUCTURE OF THE MPEG-2 CODEC

Figure 6-45 shows the structure of MPEG video compression. Each group of picture GOP is separated into I (intracoded), P or B (Non-intracoded) frames. The I-frames are separated into

16×16 macro-blocks, which are sub-sampled to four 8×8 luminance blocks (Y blocks) and 8×8 chrominance blocks (C_r and C_b blocks).

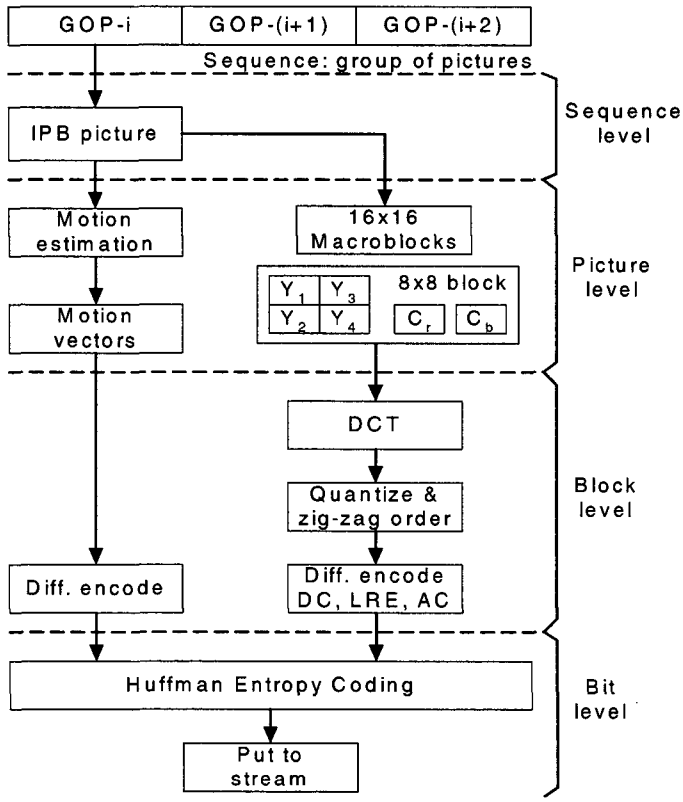


Figure 6-45: Structure of the MPEG-2 codec

The blocks pass then through a Discrete Cosine Transform DCT, which has as effect to concentrate the energy to the upper left corner of the block (lower special frequencies). A quantizing process sets to zero many of the DCT coefficients. The content of each special block is placed in a linear vector following a Zig-Zag scheme starting at the upper left corner. By doing this the low frequencies come always first in the resulting vector.

The P and B frames are defined by the motion estimation of their blocks. P frames are *forward* motion estimated and B frames *forward* and *backward* motion estimated. When a block of P or B frame cannot be obtained from the previous I or P frame (or previous and next P frame for B frames) it is treated as an I frame block. Those blocks are called I-blocks. The resulting macro-block vectors are then compressed using Huffman Entropy Coding (Variable Length Coding *vlc* in the codec package) to obtain the compressed stream.

6.1.3 MPEG-2 STREAM ENCRYPTION

6.1.3.1 MPEG-2 SELECTIVE ENCRYPTION SCHEME

The idea behind the selective encryption scheme is that encrypting only a small part of the MPEG stream could make the stream unusable to users, which have no decryption key. Theoretically, encrypting the I-frames only should render the information of the P and B frames useless. In fact it is not the case. It has been shown in [102] that encrypting only the I-frames is, in some cases, insufficient to make the stream secure. Due to the I-blocks coding in P and B frames some parts of the video will still be viewable.

Our implementation includes the encryption schemes reported in Table 6-10.

Scheme	Description
None	Original sequence – no encryption
If-DCLum	I frame only, DC coefficients of luminance blocs only
If-DCLumChr	I frame only, DC coefficients of luminance and chrominance blocs
IfIb-DCLum	I frame and I blocs of other frames, DC coefficients of luminance blocs
IfIb-DCLumChr	I frame and I blocs, DC coefficients of luminance and chrominance blocs
If-DCACs	I frame, DC and AC (sign) coefficients
IPBf-DCACs	IPB frame, DC and AC (sin) coefficients
IPBf-DCACall	IPB frame, DC and AC coefficients
Total	Naïve encryption

Table 6-10: Selective Encryption Schemes

6.1.3.2 QUALITY OF SELECTIVE ENCRYPTION

In some applications as real-time secure MPEG players the overhead due to naive encryption scheme can be unacceptable. This is an even more sensitive matter when the resolution of the MPEG stream is high (HDTV). Furthermore, in the case of broadcast communications single server multiple clients like video on demand VOD for example, the fact that each client uses a different key implies that as many streams as clients have to be sent over the communication medium. This has for effect the need of large bandwidth. Selective encryption offers:

- A way to limit the bandwidth in broadcast communications.
- A way to limit the encryption overhead in real-time decryption-decoding process.

Selective encryption schemes have also the advantage to be compatible with the MPEG-2 standard. This is true if the cryptosystem is not implemented as part of the encoder/decoder.

6.1.4 ENCRYPTION/DECRYPTION PROCESS

6.1.4.1 CONSTRAINTS

The encryption process as the compression tends to take redundancy out of the resulting data. Therefore encrypting before compression would have as effect to make the compression less effective.

- In the case of MPEG stream, the compression implies loss of information. In the MPEG codec, the information is lost during the quantizing operation.
- At the bit-level each byte is coded in variable length Huffman code. Thus, the identification of the I-blocks in the MPEG stream needs to go trough the MPEG stream bit by bit.

6.1.4.2 DESIGN CHOICES

The encryption and decryption is performed on the bit level. Due to the Hamming variable length encoding of the bytes, all the ciphers have been implemented in OFB mode, which can be easily implemented for variable length bytes.

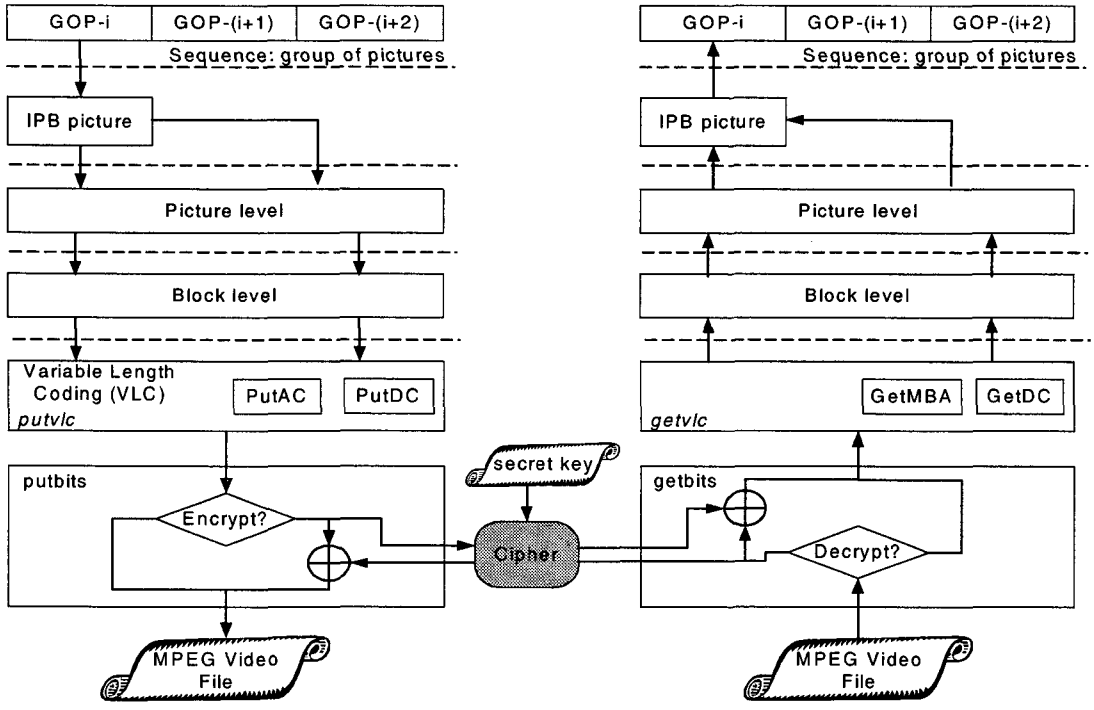


Figure 6-46: MPEG encryption/decryption structure.

A set of functions have been designed or modified from the original version in order to interface the ciphers with the MPEG codec. These functions have been designed in a way that either object oriented ciphers or ANSI-C ciphers can be interfaced.

To define when the encryption/decryption has to take place, flags are set in the *putvlc* or *getvlc* blocks (variable length encoding). The key-stream bits are retrieved from a key-stream buffer, which is refilled when empty. The size of the key stream buffer (OFB mode) varies depending on the cipher block size. Figure 6-46 shows the block diagram of the MPEG2 encryption and decryption.

6.1.4.3 INTERFACE FUNCTIONS

Following functions have been implemented in order to interface the different ciphers:

- `SecInit()`: initializes the interface. Creates the key and cipher instances.
- `GetSecRandomBit()`: Gets one bit of the key stream buffer. When the buffer is empty a new key stream block (buffer size) is generated with the chosen cipher, see Figure 6-47.
- `GetSecRandomNBits()`: N calls of the `GetSecRandomBit()`. Return N key stream bits.
- `SecReInit()`: This function is used in order to offer the possibility to reinitialize the cipher between frames or follow through any other operation, which has to take place before starting a new frame. It is used to reinitialize the SSMG stream cipher.
- `SecClose()`: This function has been create in order to support object oriented C code. It is used to place the object destructors.

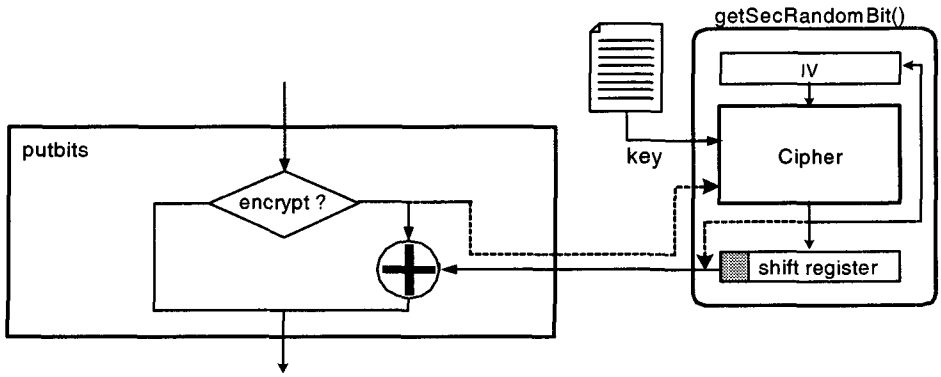


Figure 6-47: Interface for ciphers.

6.1.4.4 SSMG SOFTWARE IMPLEMENTATION

KHF function (CellHash) has been implemented as an ANSI pseudo-object based. It includes one type definition used to create instances of the KHF (including the state of the CellHash and other information) and a set of methods including one creator and one destructor.

To use the CellHash, once has to declare a variable of the `TcellHash` type and pass a pointer to the variable to the `TCellHash_Create()` function. Each method of the CellHash needs the pointer to the stage passed as input parameter.

In addition to the creator and destructor CellHash offers following functions:

- `TCellHash_PutMWORD(TCellHash* pThis, MWORD val)`: This function adds the 32 bit value to the key value (that has been set during creation process), does a hash operation and updates the state buffer value to the new result.
- `TCellHash_GetMWORD(TCellHash* pThis)`: This function gets one `MWORD` of the state buffer. Every time it is called the pointer to the state buffer is incremented. Once all the words of the state buffer have been returned, the return value is set to zero until the next time the CellHash is reinitialized.
- `SSMGERROR TCellHash_Reset(TCellHash* pThis)`: This function resets the state buffer and the state buffer pointer. Next time that `TCellHash_GetMWORD` is called the first word of the state buffer will be returned.

6.1.5 IMPLEMENTATION AND ANALYSIS

This section presents the results of performance measurement to give a comparison of the most used stream and block ciphers.

Table 6-11 shows the percentage of encrypted bits for each selective encryption scheme using a total of 6388262 encoded data-stream bits (800 Kbytes) of a normal sequence. This video sequence shows the traffic in a city (a bus followed by camera). Many different objects move in different directions on a constant moving background.

Other sequences will have different quantities of I blocks and compression rates would therefore gives different results.

Scheme	Encrypted [bits]	Encrypted [%]
If-DCLum	58476	0.9
If-DCLumChr	73240	1.1
If-DCAC	890422	13.9
IfIb-DCLum	142105	2.2
IfIb-DCLumChr	169866	2.7
IfIb-DCAC	1577346	24.7
IPBf-DCLum	142105	2.2
IPBf-DCLumChr	169866	2.7
IPBf-DCAC	5268102	82.5

Table 6-11: Encrypted Bits Per Mode

In order to have an idea of how a selectively encrypted sequence looks like in amount of bits encrypted, we have measured the number of encoded and encrypted bits by frame and processed the mean value per group of picture over the video sequence described above. Results are reported in Appendix D.

6.1.5.1 PERFORMANCE PROFILE OF THE MPEG ENCODER

The performance profile of the encoder has been measured with the Unix *gprof* application. The encoder was running on a 360 MHz, HP-UX station. In Table 6-12 are presented the most time consuming functions of the codec. These functions are shown in their respective parent/children order. All functions with little consumption and of secondary importance have been removed. The *calls* column shows the number of times the routine has been called. The total time column corresponds to the time spent in the routine and it's sub-routines during the whole program operation.

Functions	Calls	Total Time [s]	Time [%]
main	1	221.62	100
putseq	1	221.54	100
Motion_estimation	150	149.31	67.4
Transform (DCT)	150	46.92	21.2
putpict	150	11.79	5.3
PutInra	117355	1.68	0.8
PutNonIntra	58932	3.41	1.5
putbits	5286830	1.18	0.5

Table 6-12: Time Consumption Profile of the MPEG-2 Codec

Analysis shows clearly that in the codec the most time consuming functions are the motion vectors estimation (67.4%) and the Discrete Cosine Transform (21.2%).

6.1.5.2 PERFORMANCE OF THE CIPHERS DURING ENCRYPTION

In the following performance comparison of the proposed ciphers for MPEG codec we show only the complete run time and the encryption run time without giving details about the complete profile of the application. The aim here is to show the impact of the encryption on the runtime performance of the codec (ciphers encryption overhead in comparison to the performance of the MPEG-2 codec). The measures for encryption have been made using *gprof* utility.

Figure 6-48 shows the result obtained using deferent key sizes. The AES ciphers and the SSMG have been measured for 3 different key sizes. The measures have been made in the selective encryption scheme IPBf-DCACall in order to encrypt a maximum number of bits and have an idea of the worst-case scheme

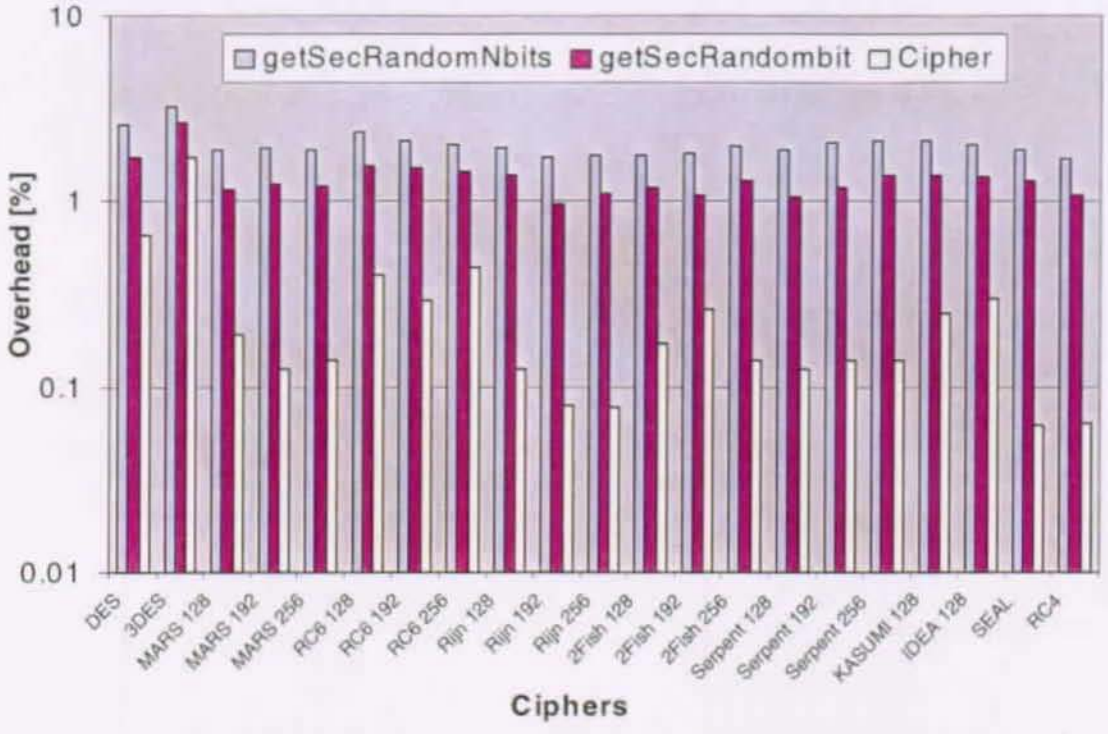


Figure 6-48: Performance of different ciphers during encryption vs MPEG encoding.

The results show that:

- Performance of encryption is very much dependant of the number of times the cipher is called. A cipher able to process bigger blocks will be called less often; it will therefore reduce the call times and might offer better performances
- The block size of the cipher has a important influence on the system performance.
- RC6 and Rijndael seem to be less optimized in their 192 bits key mode than in 128 or 256 bit keys. Rijndael256 and Twofish128 are clearly the two fastest AES algorithms and have a comparable performance.

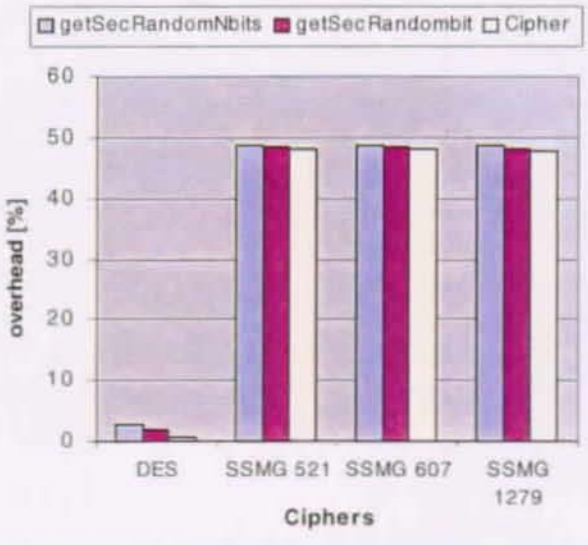


Figure 6-49: SSMG encryption overhead reported to DES

The difference of performance from one cipher to another appears to be of little importance in comparison to the processing time of the MPEG-2 encoding. In fact decryption has more an impact on the MPEG decoder than encryption on the encoder. This is due to the fact that the motion estimation is a time consuming operation during encoding.

At another side, the SSMG offers much worse performance and very high overhead. Figure 6-49 shows the very high overhead of the SSMG compared to DES cipher. The result is dependent of the number of times a cipher is called. The SSMG is called 64 times more than DES cipher and the CellHash has not been optimized as will be described in the next section.

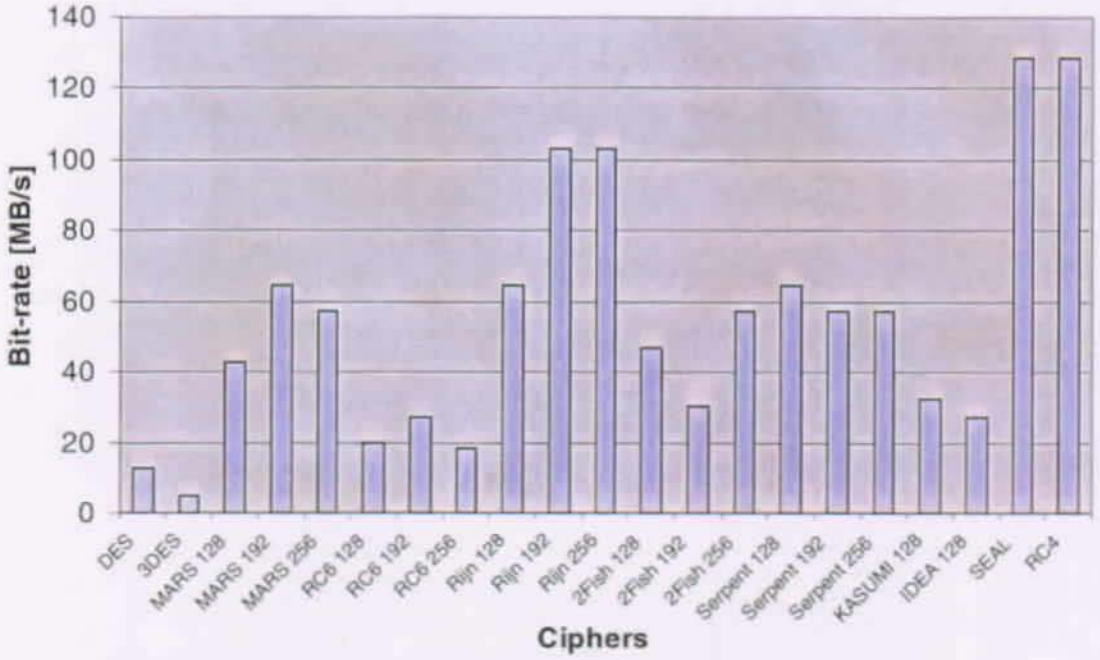


Figure 6-50: Bits Rates for the Different Cipher

Figure 6-50 and Figure 6-51 presents the bit rates we measured for the different ciphers, on the same sequence in IPBf-DCACall selective encryption mode.

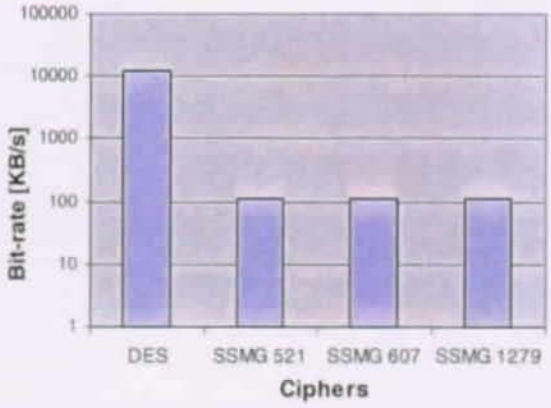


Figure 6-51: SSMG encryption bit-rate reported to DES

6.1.5.3 PERFORMANCE PROFILE OF THE SSMG

Table 6-13 shows the performance profile of the SSMG cipher:

Functions	called	Time [s]	Time [%]
getSecRandomBit	5286830	51,66	100,0%
Combiner_GeKSbit	5286830	51,22	99,1%
CellHash_PutMWORD	165213	49,52	95,9%
own time	-	23,14	44,8%
GetBit	381972456	16,57	32,1%
SetBit	169838964	9,60	18,6%
TMixGen_Output	5286830	1,09	2,1%
TMixGen_PreCombine	150	0,07	0,1%

Table 6-13: Performance profile of the SSMG with key length 607

The numbers indicate very clearly the bad performance due to the CellHash function, which 96% time consuming operation (see Figure 6-52). This is due to excessive use of *getBit* and *setBit* functions and probably a lack of optimization of the `CellHash_PutMWORD()` function. The CellHash has by now only been implemented to work, not to be outstanding. This function could probably greatly be improved by the use of 257 bits rotations.

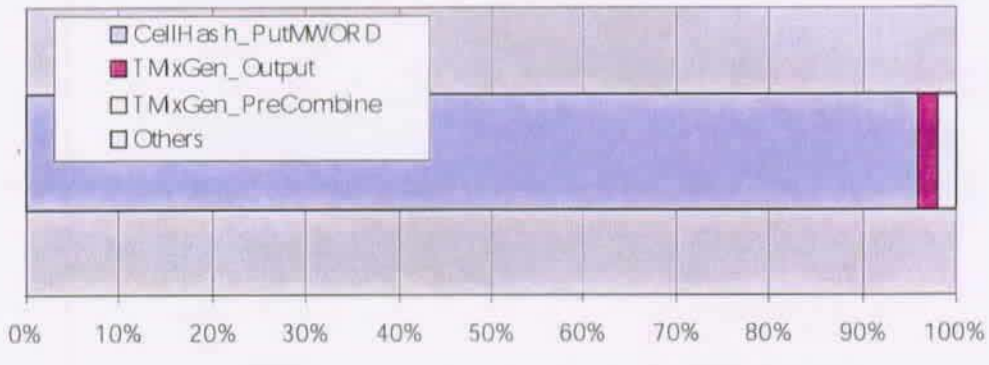


Figure 6-52: Impact of Cellhash on SSMG performance

6.1.5.4 DECRYPTION AND REAL-TIME PLAYBACK

In OFB mode encryption and decryption are symmetrical operations and therefore consume the same CPU time for the same amount of bits to encrypt and decrypt. This is true for all the ciphers but the SSMG. Since the MPEG-2 decoding process is much faster than the encoding process, the decryption has more influence on the decryption than the encryption.

Though *gprof* seems to not support our decoder implementation the Unix function time has been used to measure the processing time of the secured MPEG decoder. The sequence mentioned in above, under Section 6.1.5, has been repeated 10 times in order to obtain a more accurate measure. This corresponds to a 1500 frame sequence. Result shows that in IPBF-DCACall selective encryp-

tion mode, the encryption represents, for most of the cipher, less than 1% of the decoding time. The resulting frame rate of all ciphers but SSMG is 75 fps for a sequence at 30 fps. Real-time playback is therefore insured. SSMG frame rate is about 55 fps.

Though we could not use *gprof* for the MPEG player it is difficult to say what is really measured. The result has to be considered with care!. Each measure has been made 5 times with the mean value. These measurements have a large diffusion of ±40 ms. Therefore, this result can't be used to compare the performance of the ciphers. This comparison is already done in Section 6.1.5.2. The only result we could give is the difference between encrypted and not encrypted playback of the video.

6.1.6 MAIN RESULTS

6.1.6.1 FASTEST CIPHERS

The measure we have made here give a good idea of the performance of the ciphers already enumerated. It shows clearly that RC4 is the fastest stream cipher and Rijndael presents the best performance among the AES ciphers.

6.1.6.2 MEASURING ONE SEQUENCE

The measurements have been made only for one sequence. Different sequence would give different results due to the fact that the number of I, P or B frames can be different from one sequence to another and the number of I-blocks depends of the complexity of the movements and changes between the frames of a sequence. In fact the results can be extended to include to the other encryption schemes or sequences from the bits/sec rate that different ciphers are able to process (if the number of bits to be encrypted is known).

6.1.6.3 ENCRYPTION OVERHEAD

The results show that encryption represents a small part of the encoding-encryption process. The selective encryption offers the possibility to reduce dramatically the number of bits, but in some of the modes parts of the sequence are still visible.

In decryption-decoding process using IPBf-DCACall selective encryption mode, the processing time for the cipher represents less than 1% of the complete processing time. Further, discussions for the different modes can be found in [100].

6.1.6.4 INTERFACING THE CIPHERS

The interface between encoder and cipher has to be designed carefully in order to achieve the highest performance. The number of time a cipher is called during a sequence should be brought down as much as possible (work with big blocks). The OFB mode offers quite good results for MPEG stream encryption (variable length encoded stream).

6.1.6.5 WEAKNESSES OF THE ENCRYPTION SCHEME

The overhead due to encryption will vary depending on the video sequence. The strongest variations will happen in the IfIb (I-frames an I-blocks) encryption schemes. This is due to the fact that some complicated streams might have much more I-blocks than other steady or simple video sequences. The selective encryption scheme should be designed for the worst case.

6.2 PART2: MPEG-2 STREAM HARDWARE ENCRYPTION

6.2.1 PROGRAM STREAM AND TRANSPORT STREAM

Further to the MPEG-2 compression scheme the standard includes program and transports streams definitions [108]. The program and transport stream define the way an MPEG-2 stream will be packetized and multiplexed with other streams (audio, other video and more) for proper transport through networks (transport stream) or storage on digital storage media (program stream).

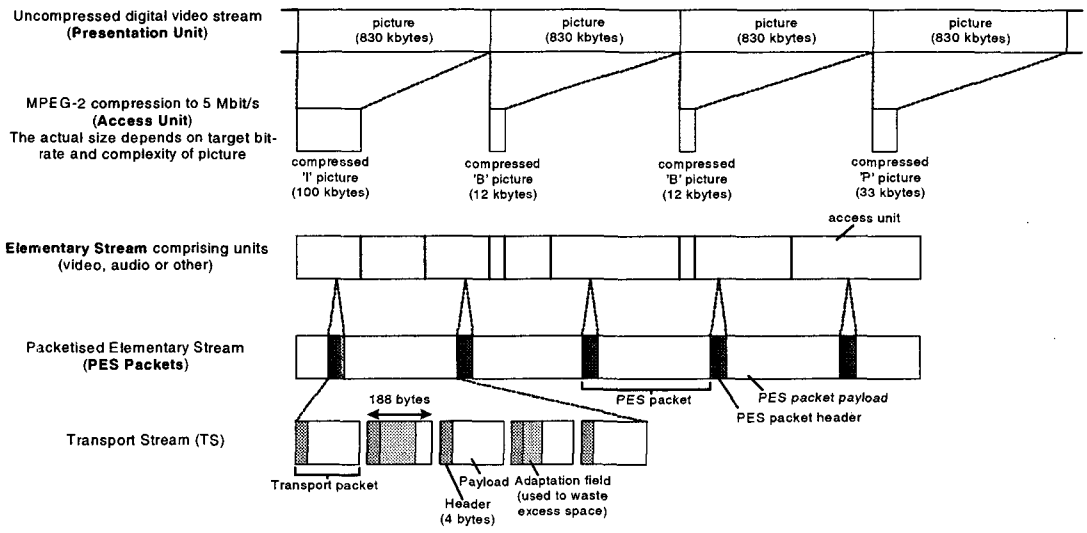


Figure 6-53: MPEG-2 from Presentation unit to Transport stream

Figure 6-53 illustrates the different streams composing the MPEG-2 systems layer [108]. The output of the MPEG-2 video/audio encoder is the *elementary stream*. This elementary stream is usually organized into *access unit*. An access unit is a frame or picture in the case of a video stream or an audio frame, in the case of an audio elementary stream. This elementary stream is now mapped onto a *packetized elementary stream* (PES), that consists of PES packets. As shown in Figure 6-53, each PES packet has a PES header followed by a variable size payload. This payload can be an exact access unit (which is a frame or a picture in the context of video) of the elementary stream. The PES packet is then converted into transport stream (TS) packets, also consisting of a header and a payload. TS packets, as can be seen from the figure, are of fixed length. MPEG-2 defines this length to be 188 bytes. This length was chosen with ATM and ATM Adaptation Layer (AAL-1) as possible transport protocol levels in mind. A TS packet maps exactly into payload of four ATM cells. An ATM cell has 48 bytes of payload, but one byte of the payload is used for the AAL-1 protocol (i.e., 4×47=188). Please refer to[108] for more detailed information MPEG-2 system standard.

6.2.2 MPEG-2 TRANSMISSION USING DVB STANDARDS

6.2.2.1 TRANSMISSION OF THE MPEG-TS

MPEG-2 Transport Stream (MPEG-2) is so called, is not, in itself a transport layer protocol and no mechanism is provided to ensure the reliable delivery of the transported data. MPEG-2 relies on underlying layers to identify the transport packets and indicate in the transport packet header,

when a transport packet has been erroneously transmitted. European Standards DVB (Digital Video Broadcasting) have been developed in order to ensures that under normal transmission condition, the received MPEG2-TS flow is quasi error free.

6.2.2.2 DVB BEARER NETWORKS

DVB standards allows a DVB signal to be carried over a range of bearer networks. Various standards have evolved which define transmission over particular types of link:

- DVB-S (Satellite) ETS 300 421 (Digital Satellite Transmission Systems).
- DVB-T (Terrestrial) ETS 300 744 (Digital Terrestrial Transmission Systems).
- Interfaces to Plesiochronous Digital Hierarchy (PDH) networks (prETS 300 813).
- Interfaces to Synchronous Digital Hierarchy (SDH) networks (prETS 300 814).
- Interfaces to Asynchronous Transfer Mode (ATM) networks (prETS 300 815).
- Interfaces for CATV/SMATV Headends and similar Professional Equipment (EN50083-9)

Digital TV transmission is currently making inroads into all types of transmission path. At the beginning, digital programs were transmitted via satellite (DVB-S), then via cable using standard DVB-C and now terrestrial digital TV is taking shape. The objectives of DVB can be briefly summarised as follows:

- More efficient utilisation of the physically limited transmission media (bandwidths) and, as a consequence, a wider spectrum of information and opinions.
- Transmission standard allowing the use of available channels for information and data other than audio and video data with equal privilege.
- TV standard combining different existing analog standards.
- TV standard providing both upward and downward compatibility for HDTV.
- Reliable picture quality improved against previous analog standard

In DVB-S, 16 bytes of Reed Solomon (RS) coding are added to each 188 byte transport packet to provide Forward Error Correction (FEC) using a RS(204,188,8) code. For the satellite transmission, the resultant bit stream is then interleaved and convolutional coding is applied. The digital bit stream is then modulated using Quadrature Phase Shift Keying QPSK modulation. In DVB-T, each MPEG-2 MPTS multiplex carries a number of streams which in combination deliver the required services. The information is transmitted using COFDM or Quadrature Phase Modulation (QPSK) (COFDM uses either 1705 carriers (usually known as '2k'), or 6817 carriers ('8k')). Reed-Solomon (RS) coding at 8% overhead.

6.2.2.3 FEC AND CHANNEL CODING

Digital video, when used in networked multimedia applications, suffers from data losses/errors. This is a serious problem in the case of wireless networks. There are several ways to recover from these losses or errors. Recovery mechanisms based on re-transmission of the data may not be suitable in many cases because of the real-time nature of the applications and the absence of reverse channel for feedback. Real-time communication of digital video, as in the case of video conferencing, benefit from forward error correction/recovery techniques. Forward error correction (FEC) codes and frequent synchronizing codewords are used in which sufficient extra bits, known as redundancy, are added to the data to allow the decoder to perform corrections in real time.

The FEC used in modern systems is usually based on the Reed-Solomon (R-S) codes. A full discussion of these is outside the scope of this thesis. Briefly, R-S codes add redundancy to the data to make a code-word such that when each symbol is used as a term in a minimum of two simulta-

neous equations, the sum (or syndrome) is always zero if there is no error. This zero condition is obtained irrespective of the data and makes checking easy. In Transport streams, the packets are always 188 bytes long. The addition of 16 bytes of R-S redundancy produces a standard FEC codeword of 204 bytes. In the event that the syndrome is non-zero, solving the simultaneous equations will result in two values needed for error correction: the location of the error and the nature of the error. However, if the size of the error exceeds half the amount of redundancy added, the error cannot be corrected. Unfortunately, in typical transmission channels, the signal quality is statistical. This means that while single bits may be in error due to noise, on occasion a large number of bits, known as a burst, can be corrupted together. This corruption might be due to lightning or interference from electrical equipment.

It is not economic to protect every code word against such bursts because they do not occur often enough. The solution is to use channel coding which is a technique known as interleaving. When interleaving is used, the source data are FEC coded, but prior to transmission, then the sequential order of the data stream is scrambled or reordered using a RAM in order to disperse the MPEG-2 packet data throughout time. On reception, the data are put back to their original order, or deinterleaved, by using a second RAM. The result of the interleaving process is that a burst of errors in the channel after deinterleaving becomes a large number of single-symbol errors, which are more readily correctable.

When a burst error reaches the maximum correctable size, the system is vulnerable to random bit errors that make code-words uncorrectable. The use of an *inner* code applied after interleave and corrected before deinterleave can prevent random errors from entering the deinterleave memory. When this approach is used with a block interleave structure, the result is a product code. Moreover, interleave can also be convolutional (*Trellis* encoding), in which the data array is sheared by applying a different delay to each row. Convolutional, or cross interleave, has the advantage that less memory is needed to interleave and deinterleave.

There are other factors to be considered: efficient modulation and multiplexing schemes are needed for maximum utilization of bandwidth and tolerable rate of signal error. Thus, FEC when used with QPSK modulation uses two forms of error correction. In order to correct for burst errors at the receiver, it will be necessary to implement a two level FEC strategy of an outer Reed Solomon (204:188) Code for an inner FEC Convolutional code. This strategy will enable the realization of a *convolutional* coding with *Viterbi* algorithm code for inner FEC decoding at the receiver.

Inner decoding for correcting burst errors is implemented as a Viterbi decoder, while Reed Solomon decoding is undertaken for outer decoding in order to complete the channel decoding stage. After the convolutional error correction code has been removed and used as needed, a second error form of error correction is used called the *Reed-Solomon code.* This correction results in 188 bytes out for every 204 bytes coming in with the remainder used as parity bits to help correct any remaining errors. Additionally, the FEC scheme also uses interleaving of the data stream to prevent noise bursts from interrupting the flow of data.

6.2.3 DVB ENCRYPTION SYSTEM

A general overview of the DVB system architecture is shown in Figure 6-54. The function of each block is as follows:

- *source coding and multiplexing:* This provides the MPEG-2 compression tools to compress the digital video and audio signals to the rate required by the data stream which, in turn, depends on the specified quality of the media signal. It helps to reduce the bandwidth needed.

- *channel adaptation:* This block helps to facilitate the transmission of the MPEG-TS over the specific delivery medium. A particular modulation and coding schemes are adopted to ensure that the signal is matched to the channel environment. Depending on the channel characteristics, either a more robust or a more spectrum-efficient modulation scheme may be optimum [119].
- *Transmission media:* The transmission media can be based on delivery by satellite, cable, SMATV or terrestrial transmitters (UHF or microwave). Each one has its own physical properties which have been considered in the development of the DVB systems. **Obviously, the DVB Project does not specify particular requirements for the transmission medium, but the transmission channel characteristics have duly been taken into account to ensure that reception of the MPEG-TS is virtually free of errors. The concept utilized by DVB is based on *quasi-error-free* QEF reception of the MPEG2-TS signal. This means that less than one uncorrected error-even-per-hour should be presented to the MPEG decoder.**
- *Demodulation*: implemented in the receiver chain in order to acquire the baseband MPEG2-TS signal and to remove the RF carrier signal.
- *Decoding:* MPEG2-TS source decoding.

The encryption module is then incorporated at the transmitter chain between the source coding and channel adaptation. Decryption is performed on the stream between demodulator and decoder.

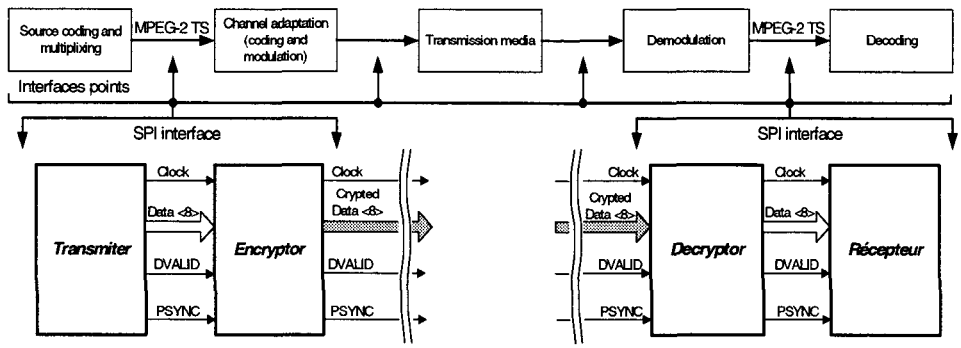


Figure 6-54: DVB Encryption System using SPI

6.2.3.1 BASEBAND MPEG-2 INTERFACES

Baseband MPEG-2 interfaces are used to communicate MPEG data packets amongst collocated MPEG equipment. These interfaces may be categorized in different ways: some transmit serial bit streams, while others transmit parallel byte streams. An important differentiating factor in categorizing these interfaces is the relationship between the raw clock rate and the rate at which MPEG-2 data is transmitted over the interface. Two types of relationships exist between these rates:

- Packet synchronous (PS): The raw clock rate is directly proportional to the rate at which MPEG-2 data is transmitted; such interfaces contain no padding.
- Packet asynchronous (PA): The interface's raw clock rate is fixed, while MPEG-2 data is transmitted over the interface at some maximum allowable capacity. The capacity is determined by the raw clock rate and any overhead required by the interface.

The PA interface may be thought of as a fixed-size pipe through which data may be pumped at any rate up to the maximum capacity of the pipe. The PS interface is more like a customized pipe,

the capacity of which has been matched to the data rate. According to PS and AS, three different interfaces have been standardized ASI, SSI and SPI.

6.2.3.1.1ASYNCHRONOUS SERIAL INTERFACE (ASI)

The DVB-ASI interface has become popular for use with infrastructure equipment in such facilities as cable headends or uplink sites. DVB-ASI is a fixed-frequency serial interface with a clock rate of 270 Mbps that transmits MPEG-2 data in packet asynchronous fashion.

6.2.3.1.2SYNCHRONOUS SERIAL INTERFACE (SSI)

This is DVB's less popular synchronous serial interface. Like the other DVB interfaces, it is a point-to-point uni-directional link. The clock frequency varies depending on the data rate required by the MPEG-2 transport stream, with a maximum upper limit of 105 MHz. The DVB-SSI is a PS interface with the clock rate locked to the transport rate so that no stuffing is needed. In fact, if no error correction bytes are included, the data rate equals the raw clock rate. The SSI interface allows transport of 188-byte packets, or 204-byte packets with 16 bytes reserved for RS coding bytes or dummy bytes. Automatic packet type detection is performed using the sync byte's period of occurrence and inversion of the MPEG-2 sync byte to 0xB8 when valid.

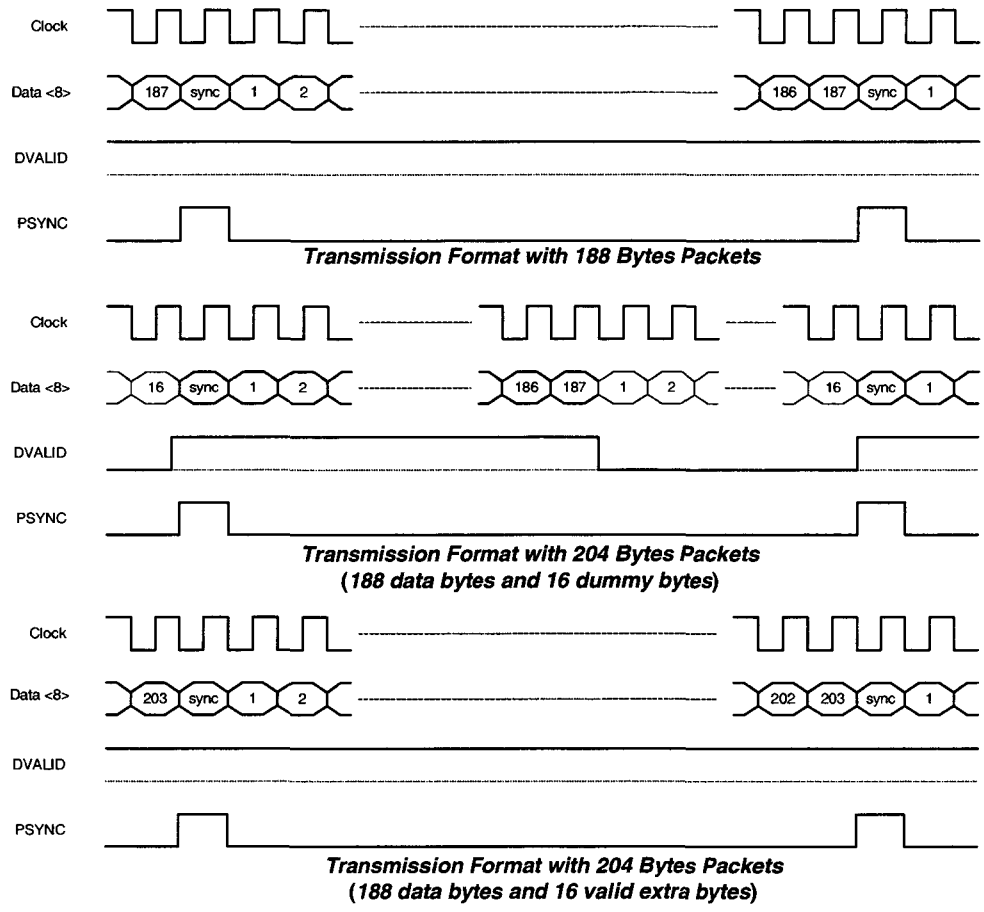


Figure 5-55: MPEG-2 TS packet formats for SPI

6.2.3.1.3SYNCHRONOUS PARALLEL INTERFACE (SPI)

The SPI interface is a fairly popular point-to-point unidirectional parallel interface for connecting infrastructure equipment at the headend or within an uplink site. Since the transport packet rate is locked to the clock rate, the interface is PS; it transmits byte wide with a clock, packet *sync*, and data-*valid* signal as shown in Figure 5-55. The data-valid signal is used to differentiate between bytes that are part of the 188/204 byte packet versus dummy bytes that follow a 188-byte packet.

The interface use the MPEG2-TS Packet structure (188 bytes) or the RS-coded packet structure (204 bytes). The 204 bytes coming in with the remainder used as parity bits (addition of 16 bytes of R-S redundancy to produce a standard FEC code-word) to help correct any remaining errors. For the Synchronous Parallel Interface (SPI) and the Synchronous Serial Interface (SSI), the 204-byte format may be used either for the transmission of 188-byte MPEG2-TS packets with 16 dummy bytes, or for the transmission of 204-byte RS-coded packets.

6.2.3.2 CRYPTOSYSTEM STRUCTURE

Figure 6-56 shows the overall system architecture of the PK-SSMG. The architecture consist of a datapath with single data bus connecting all the different instances associated with a central control processing unit for resources sharing, while minimising the clock period and interconnections which also makes it possible to take account of low power consumption constraints. The datapath includes a hardware operators comprising the exponentiator unit and the SSMG.

The combination of the exponentiator with SSMG makes the cryptosystem operating in three different modes, exponentiation mode, an initialization phase and a combining or streaming phase. The exponentiation phase is done once when setting up the secret key. The later operating modes are best viewed as two connected parts, unified by the concept of a mixture generator and the haching unit.

The exponentiation mode consist of Key generation operation. The *Public* operator generates the *Open Key* Q and the *Closed Key* K from the *Public Key* E and *Random Key* R for encryption and the Private operator generates the same Closed Key K from Private Key D and the Open Key Q for decryption. These operations are described under Section 3.4.3.4. The secret key K is then stored in the Key Memory. The initialization phase consist of resetting the mixture generator, loading the Mixer with the Closed Key K and initializing the MG according to scheme described in Section 5.3 for both encryption and decryption operation. Combining consist of encrypting and decrypting the frames according to scheme described in Section 5.3.

All input/output data pass through the input/output FIFOs including the SPI data and the user control and data signals. These interfaces operates at the link speed (13.5 MHz) while system's internal frequency is set to 100 MHz.

6.2.3.2.1 PK-SSMG INTERFACE

The interface comprises the IO pin configuration, the SPI IO data and a user interface for test and forward key mode. The system configuration parameters are set prior to the combining phase. These includes the encryption/decryption mode, mixture generator configuration, the key size and the bypass mode. During initialization and combining mode the granularity may be changed on the fly. Its configuration bit is stored in the frame decoder for use at the start of each new frame.

The user interface can be used for external key load or supplying the system with the exponent and the base of the exponentiator. Three bits *usr_data_ctrl*, *source*1 and *source*2 are used for the configuration of source/destination of user input data as shown in Table 5-14. The *usr_data_ctrl* bit is

used to configure the user interface's input/output data port while source1 and source2 bits specify the destination source.

usr_data_ctrl	source1	source2	Destination
1	0	0	Exponentiator (exponent)
1	0	1	Exponentiator (X)
1	1	0	Exponentiator (base)
1	1	1	key Memory

Table 6-14: User interface control bits: data input destination

In output mode, the user may access the exponentiator, the KHF and the MG for test purpose. The user control bits values in output mode are reported in Table 6-15.

usr_data_ctrl	source1	source2	output data from	Port destination
0	0	0	Output FIFO	*data_out*
0	0	1	MG	*data_ctrl*
0	1	0	Exponentiator	*data_ctrl*
0	1	1	KHF	*data_ctrl*

Table 6-15: User interface control bits: data output

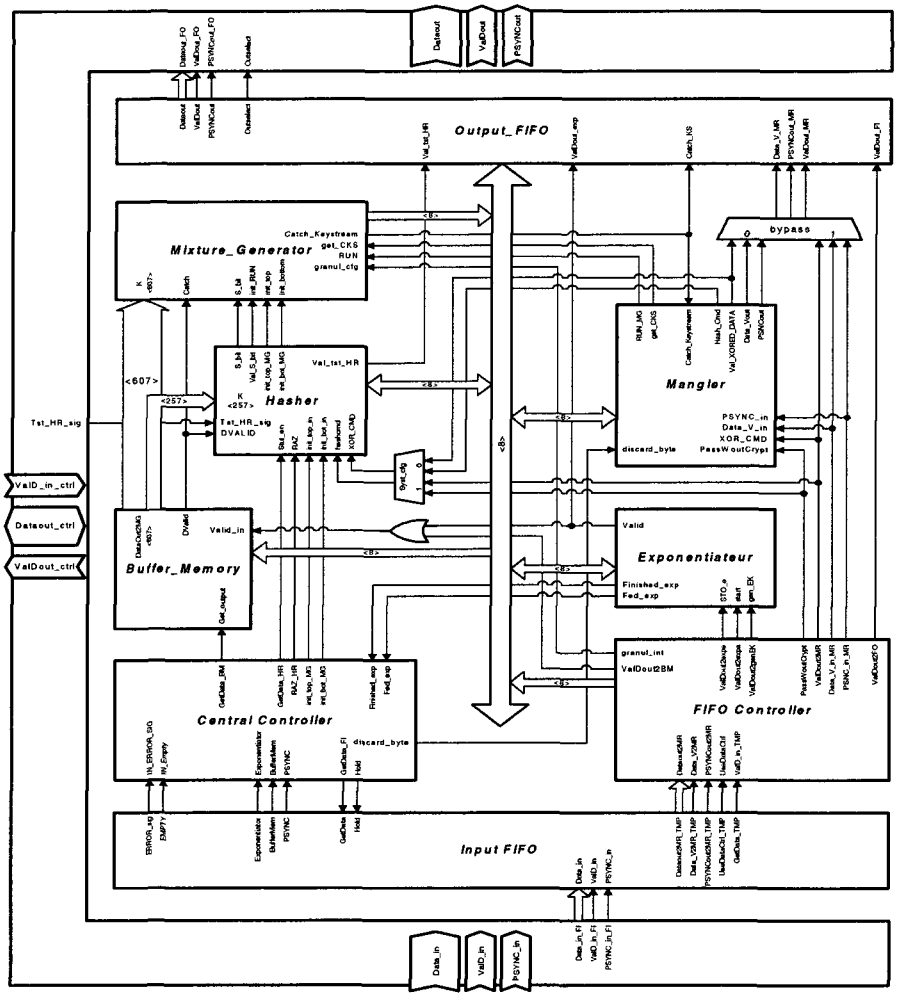
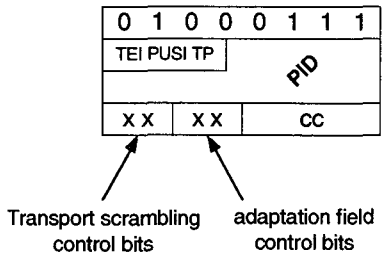


Figure 6-56: PK-SSMG datapath

6.2.3.2.2 MPEG-TS PARSING (FRAME DECODING)

The MPEG2-TS Packet structure (188 bytes) or the RS-coded packet structure (204 bytes) begins with a 4-byte header. Of the various fields it contains, as shown in Figure 6-57, two are of particular interests:

- *Adaptation field control*: 2-bit field ad_1 and ad_2 indicate whether the TS packet contains an adaptation field or not and whether an adaptation field id followed by a Payload or not. If an adaptation field is present then the next byte contain the length size of this field.
- *Transport scrambling control*: 2-bit field sc_0 and sc_1 indicate whether the TS packet is scrambled or not. The values of the TS *scrambling control* bits are parsed, prior to the decryption process.



sc_1, sc_2	description
00	bypass
01	bit granularity
10	byte granularity

ad_1, ad_2	description
00	no adaptation field, payload only
01	adaptation field only
10	adaptation field followed by payload

Figure 6-57: TS packet header

The MPEG2-TS parsing is done in the frame decoding unit. Packet's payloads with sc_0 and sc_1 bits equal to 01 are encrypted using bit granularity. The configuration 10 is used for byte granularity. When these control bits are equal to 00, the payload data of the corresponding packet are bypassed. The adaptation field and packet header bytes are bypassed to the output FIFO.

6.2.3.2.3 MIXTURE GENERATOR

The MG is programmable in security with respect to the LFSRs lengths. Further, the LFSRs lengths are chosen to be $L_{MIXER}>L_{TOP}>L_{BOTTOM}$ in order to optimize the hardware. The field size m and parameter k for which an irreducible trinomial of degree m in Finite Field exists are reported in Table 6-16. These pairs define maximal period LFSRs and high MG linear complexity.

MG configuration (L_{MIXER}, L_{TOP} L_{BOTTOM})	MIXER	TOP	BOTTOM
(127,89,87)	127/97	89/51	87/13
(521,127,89)	521/158	127/97	89/51
(607,521,127)	607/273	521/158	127/97
(1279,521,127)	1279/418	521/158	127/97

Table 6-16: MG configuration parameters

6.2.3.2.4 EXPONENTIATOR

The digit-serial LSA exponentiator reported in Figure 4-28 is implemented with D=4. The architecture has been modified to be programmable with respect to the key size (L_{MIXER}). This is done by simply adding some multiplexers to choose the right intermediate signals in feedback loop of. The IO data of the exponentiator are formatted at the input and output stages (interfaces between the input FIFO, the exponentiator and the bus). Since the user input data are buffered at the

link speed and the exponentiator is clocked at the internal clock frequency, the exponentiator input data are not immediately available. Thus, during the feeding operation the exponentiator is halted and its internal state is freezed when the input data is not available. Furthermore, the vector X involved in the computation of the exponentiator (Section 3.4.3.4) is generated internally using the control signal of the exponentiator.

6.2.3.2.5INPUT/OUTPUT FIFOS

The DVB-SPI frequency and the modes of operation of the mixture generator and its granularity described in Section 5.2, impose some buffering of the input and output data in order to reduce the latency and provide a constant IO bitrate (13.5 MB/s). This buffering is implemented using FIFOs operating at the link speed and whose size may be matters at issue. Speeding up the frame processing while maintaining the required IO bitrate is one solution. In fact, there exist a trade-off between the MG's period (size) and the size of the FIFOs. Using long period for increasing security yields to oversize the MG, increases the latency and thus, necessitate large FIFOs. Increasing the system's internal frequency yielding to operate at two system frequencies may be exploited to preserve the MG security properties. Table 6-17 and 6-18 shows a worst cases analysis of SSMG latency operating at 100 MHz in bit and byte granularity respectively using two frame's formats.

MG configuration (L_{MIXER}, L_{TOP} L_{BOTTOM})	MG initialization (cycles)	Frame processing (cycles)	Total delay @100MHz (ns)	>/<	input Frame time @13.5MHz (ns)
(127,89,87)	89+87=176	188/204	3640/3800	<	13912/15096
(521,127,89)	216	188/204	4040/4200	<	13912/15096
(607,521,127)	648	188/204	8360/8520	<	13912/15096
(1279,521,127)	648	188/204	8360/8520	<	13912/15096

Table 6-17: SSMG Latency: byte granularity

Thus, when operating at 100 MHz, the time required for processing one frame is less than the input frame time. This is the case when the MG operates in byte but bit granularity (Table 6-18). The frame processing latency is increased due to the slowness of MG. In this case we spend much time in processing the frames, delay is accumulated and the IO data cannot be efficiently buffered.

MG configuration (L_{MIXER}, L_{TOP} L_{BOTTOM})	MG initialization (cycles)	Frame processing (cycles)	Total delay @100MHz (ns)	>/<	input Frame time @13.5MHz (ns)
(127,89,87)	89+87=176	1504/1632	16800/18080	>	13912/15096
(521,127,89)	216	1504/1632	17200/18480	>	13912/15096
(607,521,127)	648	1504/1632	21520/22800	>	13912/15096
(1279,521,127)	648	1504/1632	21520/22800	>	13912/15096

Table 6-18: SSMG Latency: byte granularity

We can remedy this issue using the LFSR architecture shown in Figure 5-40. Thus, speeding up the MG when operating in bit granularity with keystream output bit rate equal to byte granularity mode.

The worst case corresponds to 648 clock cycles required for the initialization of the MG in (1279,521,127) configuration and operating at 100 MHZ internal frequency. During this time 88 IO data bytes must be buffered at 13.5 MHz. This is the minimum size of the input/output FIFOs. Moreover, the FIFOs register based design uses a pointer-based scheme. When the FIFO is accessed, only the pointer stored in a single bit shift register moves, not the data. Compared to other register based FIFO designs, where significant power can be consumed in shifting data between registers, this scheme minimizes power by minimizing data switching.

E	P	V	Destination
0	PSYNC	VALID	Mangler
1	0	0	Exponentiator (exponent)
1	0	1	Exponentiator (X)
1	1	0	Exponentiator (base)
1	1	1	key Memory

Table 6-19: Input FIFO: data flags

The FIFOs contains blocks of 11-bit data. Among these are 8-bit input data from the SPI, the valid bit and PSYNC bit. The two later bits are used with an additional bit as flags. The forwarding FIFOs data may be used to supply an external secret key to the key memory or other exponentiator data. Table 6-19 shows the value of these flags and the corresponding destination of the data.

6.2.3.2.6 BUS MULTIPLEXING

In order to reduce the capacitance of point-to-point buses due to locality of reference (communication only with physically adjacent functional units see Section 2.3.3.2), a single shared bus is used. Different tasks are sequenced using finite control state machine and data are exchanged on the shared bus. Table 6-20 shows sources and destinations units sharing the bus.

Operation	Source	Destination
Exponent acquisition	input FIFO	Exponentiator
expo. base acquisition	input FIFO	Exponentiator
Exponentiation (result)	Exponentiator	output FIFO
key forward load	input FIFO	key Memory
MG initialization	key Memory	MG
SSMG test	KHF	output FIFO
Combining process	input FIFO MG Mangler	Mangler Mangler output FIFO and KHF

Table 6-20: Bus Sharing: source and destination

The Concurrent signals during combining phase are processed using time multiplexing bus. Three cycles are required to process one byte of data frame (Figure 6-58) yielding to increase the latency. Using the specification of input/output FIFOS we are able to buffer the input/output frames before receiving a complete frame (no overflow), see Table 5-21. In the worst case, 110 IO data bytes must be buffered.

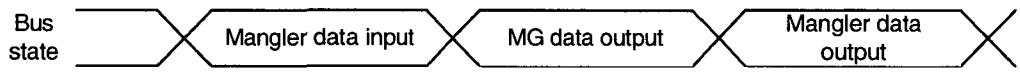


Figure 6-58: Bus multiplexing during combining phase

MG configuration (L_{MIXER}, L_{TOP}, L_{BOTTOM})	MG initialization (cycles)	Frame processing (cycles)	Total delay @100MHz (ns)	>/<	input Frame time @13.5MHz (ns)
(127,89,87)	16+89+87=192	564/580	7560/7720	<	13912/15096
(521,127,89)	282	564/580	8460/8620	<	13912/15096
(607,521,127)	724	564/580	12880/13040	<	13912/15096
(1279,521,127)	808	564/580	13720/13880	<	13912/15096

Table 6-21: SSMG Latency with bus sharing: bit/byte granularity

6.2.3.2.7 MANGLER

In the case of adaptation and packet header fields the output bytes from the input FIFO are not encrypted. These bytes are forwarded to the output FIFO. The Mangler request the keystream from the MG only when the data is a part of the payload. The MG is disabled in case where the data is bypassed and a control signal is received from the MG, which indicates that the bus is disposed for the Mangler data output.

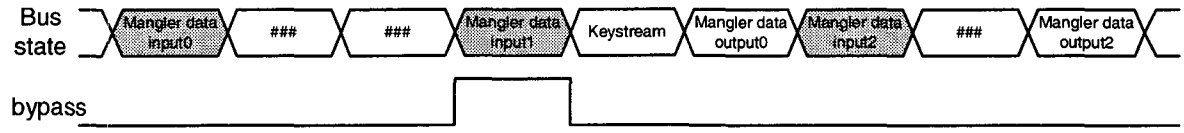


Figure 6-59: Bus multiplexing: data bypass and encryption

The Mangler request the keystream after receiving one byte data valid. The keystream is received two clock cycles later due to the latency of the MG. Thus, the mangler data input is buffered up to 2 bytes. Figure 6-59 shows the bus multiplexing scheme during combining phase when the Mangler data input is bypassed or encrypted.

6.2.3.3 PK-SSMG CONTROLLER

The controller is a finite state machine that controls the cryptosystem instances and mange the shared bus.Figure 6-60 gives an overview over the state diagram governing the exponentiation and the MG initialization process. The combining process including the mangler and the MG is controlled by the frame decoder.

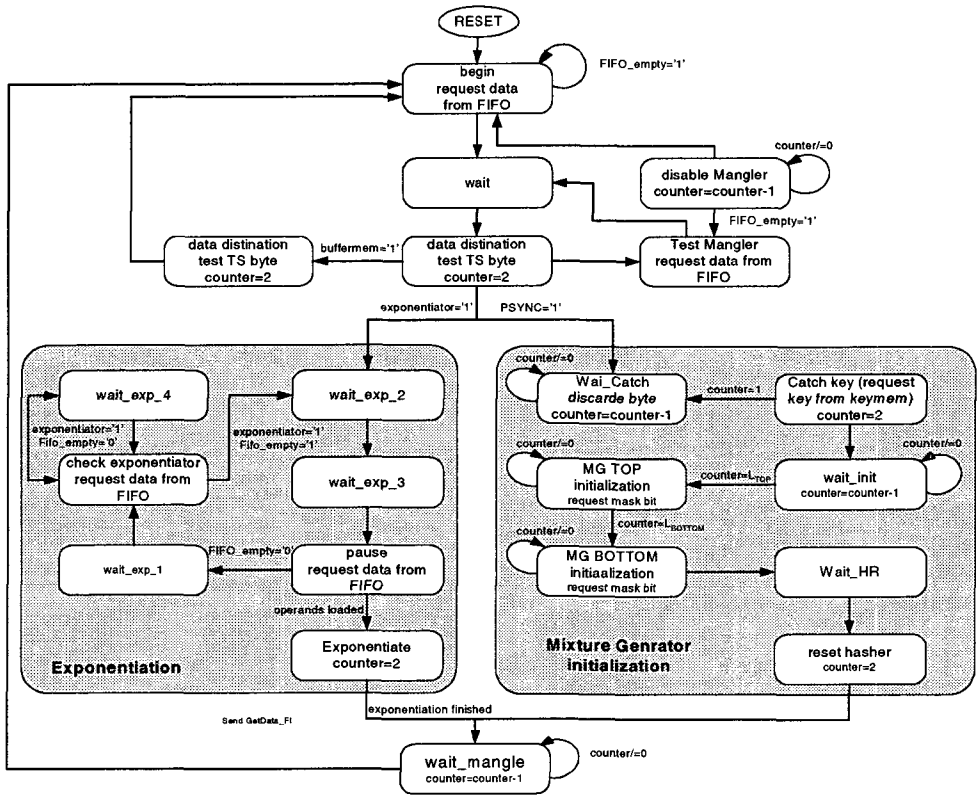


Figure 6-60: PK-SSMG finite state machine (controller)

At the start we request a byte from the input FIFO. It is processed according to its destination (if the FIFO is not empty): key memory or exponentiator. If the data is the first byte of a new frame, the Mixture Generator is initialized. because of the system operates at two different frequencies, certain block such as the exponentiator are frozen (waiting for the next data bytes), when loading the data operands while the FIFO is empty. Also, some wait states are included in order to overcome the inherent latency of certain processing blocks which may be due to the data formatting.

6.2.3.3.1 THE BURST MODE

During MG initialization phase, the incoming data are accumulated in the input FIFO. After initialization the input FIFO is guttered till empty at internal frequency rate; the data is then read each 3 clock cycles. In normal mode. When the input FIFO is empty, the read process becomes dependent upon the incoming data from SPI. The delivery rate (processing time) is then reduced to one processing each 7-8 cycles since the link speed is 13.5MHz. The burst mode is defined as the interval of time spent in reading the FIFO's data after MG initialization and before normal mode as shown in Figure 6-61. During burst mode an amount of data bytes is available in the input FIFO, which correspond to the bytes received from the SPI during MG initialization phase.

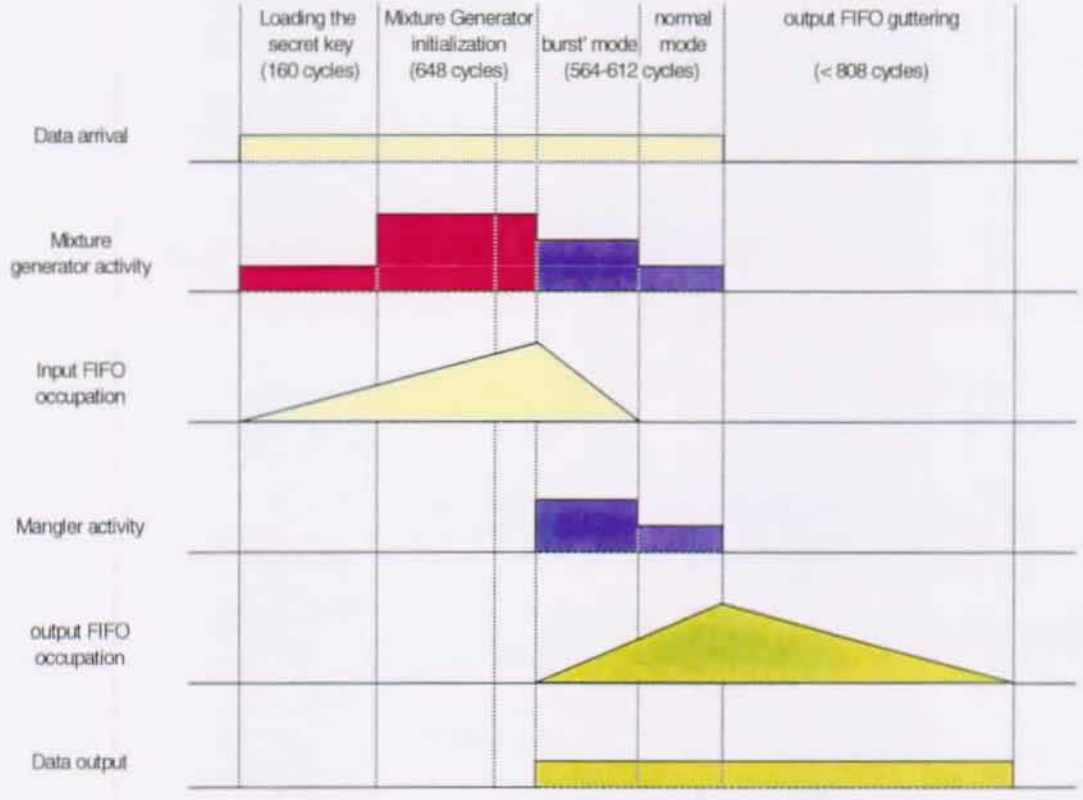


Figure 6-61: Burst mode

6.2.3.3.2 EXPONENTIATION PHASE

The bytes destined to the exponentiator (when Exponentiator ='1') are formatted are the input/output stage. Thus two clock cycles per byte are required when loading input operands during burst mode. In normal mode (input FIFO empty) the exponentiation is frozen waiting for SPI data. Ex-

ponentiation phase starts when the input operands are completely loaded. During the this phase the MG, hash module and mangler are disabled till end of computation.

6.2.3.3.3 MG INITIALIZATION AND COMBINING PHASES

When the received byte is the first byte of a new frame (PSYNC='1') it is discarded and the MG is initialized. It is requested afterwards when MG is initialized. During this phase the input FIFO is filled with the SPI data bytes. These bytes are processed during burst mode. When switching between the fast cycle (burst mode) and normal mode, the incoming data rate from the input FIFO is reduced to one processing each 7-8 cycles. The fact that link and internal clock frequencies are not synchronized at the end of burst mode, a test is made (test mode state in Figure 6-60) to detect the end of burst mode by requesting a data from the input FIFO. During decryption at the end of burst mode, the bus can be shared between the SPI data and the keystream or the output of the previous data byte. In this case the requested keystream output and the incoming Mangler output data must be sufficiently shifted in time such that they can be written on the bus. Another state is added to the controller (disable Mangler) in order to disable the data request from the input FIFO as shown in Figure 6-62.

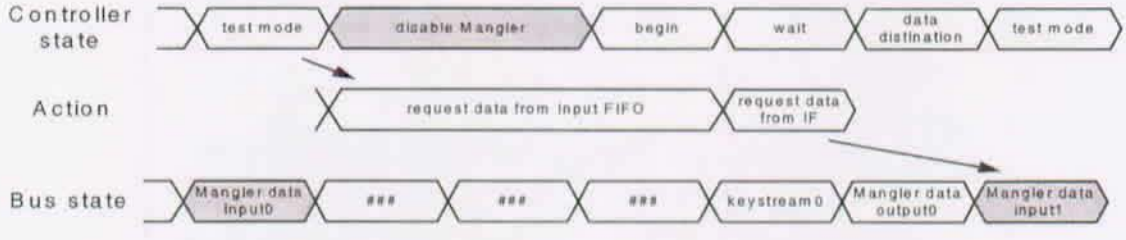


Figure 6-62: Disabling data request from the input FIFO

6.2.4 PERFORMANCE MEASUREMENT AND RESULTS

The cryptosystem was implemented at gate level using the same technology $1.8\mu m$ operating at 1.8v. It was tested at all possible key lengths, granularity and packet sizes and was found to be fully functional in all configurations.

	Clock free architecture, K = (127, 521, 607,			Clock free architecture, K = (127, 607)			Clock gating architecture, K = (127,607)		
Area [Kgates]	total	127.50		total	73.36		total	62.25	
	Exponentiator	72.80	57.1 %	Exponentiator	34.40	46.9 %	Exponentiator	27.26	43.8 %
	Mixture Generator	25.24	19.8 %	Mixture Generator	16.14	22.0 %	Mixture Generator	14.94	24.0 %
	key Memory	9.43	7.4 %	key Memory	4.11	5.6 %	key Memory	5.85	9.4 %
	output FIFO	6.76	5.3 %	output FIFO	6.75	9.2 %	input FIFO	5.60	9.0 %
	input FIFO	6.50	5.1 %	input FIFO	6.38	8.7 %	output FIFO	4.86	7.8 %
	CELLHASH	6.25	4.9 %	CELLHASH	4.99	6.8 %	CELLHASH	3.24	5.2 %
	Others	0.51	0.4 %	Others	0.59	0.8 %	Others	0.50	0.8 %
Power Consumption [mW]	total	216.13 mW		total	117.30 mW		total	50.97 mW	
	Exponentiator	126.00	58.3 %	Exponentiator	53.49	45.6 %	Exponentiator	41.84	82.1 %
	CK	40.85	18.9 %	CK	37.54	32.0 %	CK	3.62	7.1 %
	key Memory	16.43	7.6 %	key Memory	7.16	6.1 %	key Memory	0.00	0.0 %
	Mixture Generator	10.81	5.0 %	Mixture Generator	6.80	5.8 %	Mixture Generator	1.99	3.9 %
	output FIFO	10.81	5.0 %	output FIFO	5.28	4.5 %	output FIFO	0.76	1.5 %
	CELLHASH	10.37	4.8 %	CELLHASH	4.69	4.0 %	CELLHASH	0.66	1.3 %
	input FIFO	0.00	0.0 %	input FIFO	0.94	0.8 %	Input FIFO	0.76	1.5 %
	CKB	0.00	0.0 %	CKB	0.00	0.0 %	CKB	0.66	1.3 %
	Others	0.86	0.4 %	Others	1.41	1.2 %	Others	0.66	1.3 %

Table 6-22: Area and power consumption of each instance in the PK-SSMG cryptosystem

Two different architectures were implemented. The first one is security scalable with four different key lengths 127,512,607 and 1279 bits. In the second architecture the key can be chosen to be 127 bits length as low security level or 607 bits as high security level. The power consumption and gates area are estimated. Table 6-22 shows the amount of power consumption and area occupation of each instance of the cryptosystem for both implementations.

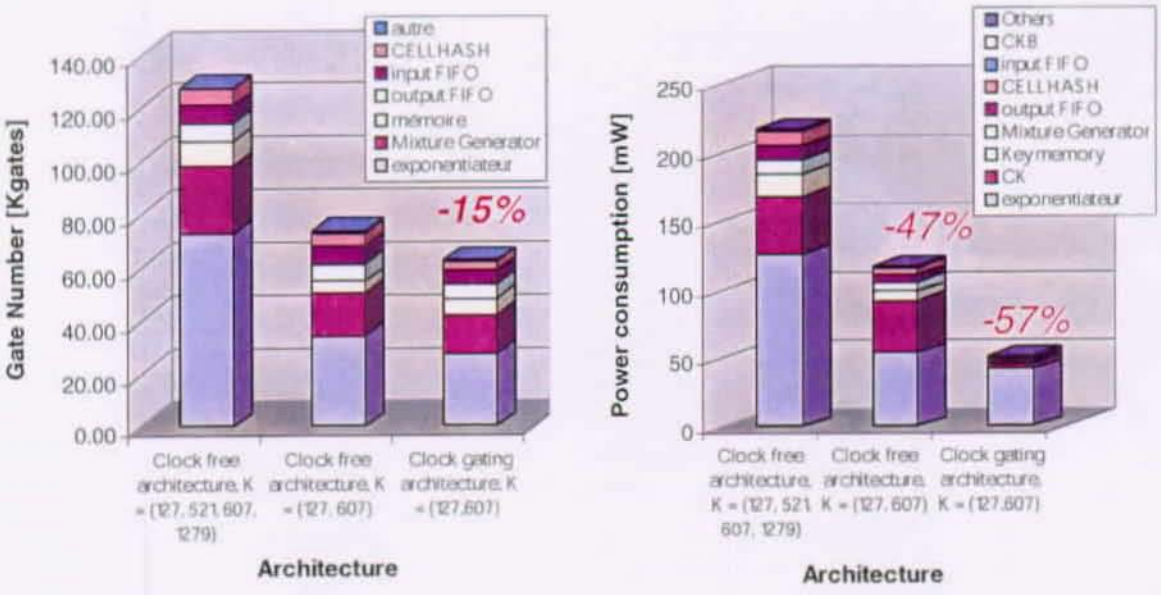


Figure 6-63: Area and power profiles of the cryptosystem

The power is estimated using gate level simulation on a pseudo random input. The test comprises 127 bit exponentiation and the encryption of 4 frames. It is shown that the exponentiator consume the greatest power and area shares (more than 40%). As shown in Figure 6-63, gating the clock helps to reduce significantly (more than 50%) the power consumed by different instances. Also, a gain of 15% is noticed on the area when gating the clock. Hardware implementation is 2 orders of magnitude more area efficient when scaling the key length from 1279 down to 607 bits.

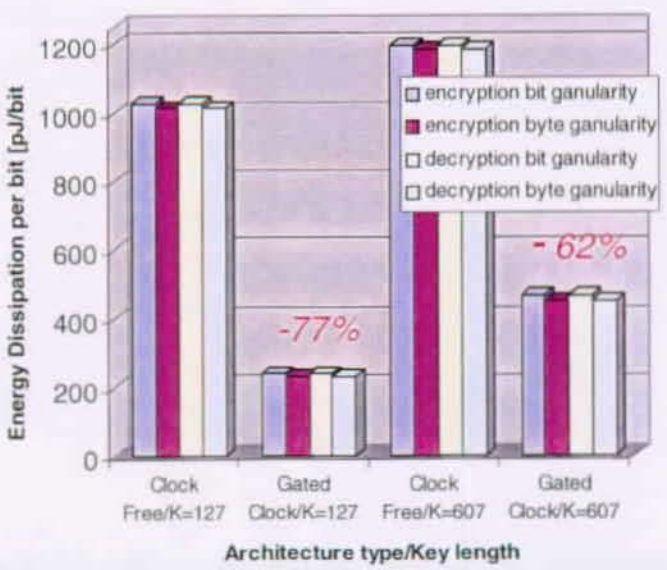


Figure 6-64: Energy consumption per bit of different implementations

The energy scalability of the cryptosystem is seen in Figure 6-49 which shows the effects of reducing the key length (disabling unused datapaths) and gating the clock to exploit a reduction in the amount of computation as the key length is varied from 607 bits down to 127 bits, at a fixed data stream rate of 13.5MBytes/s. The results reported in Figure 6-64 are estimated using gate level simulation for the encryption and decryption of 20 pseudo random frames using bit and byte granularity. The granularity has little effect on the power consumption as only little decrease (about 1.3%) in the switching activity is noticed at the same clock frequency. The same energy is consumed during encryption and decryption process as the system is completely symmetrical (the same resources are shared during these processes).

Further, using 3 times low level of security helps to reduce the amount of energy consumed per bit by 17% only. However, gating the clock reduces significantly the amount of energy. The optimal solution dissipates between 200pJ/bit (@13.5 Mbytes/s) using low level of security and 400 pJ/bit at the same throughput using high level of security.

6.2.4.1 PHYSICAL DESIGN

The PK-SSMG core has been designed and synthesized in a 0.18um 6LM technology (technology with high integration density) with 100 MHz internal system frequency. First result of the placed and routed core including all components (exponentiator, SSMG) gives an area of less than $3mm^2$ ($2,76mm^2$ including the Die boundary and I/O pins). Detailed implementation numbers of the core are shown in Table 6-23. A layout of the PK-SSMG is shown in Figure 6-65. where the different components are highlighted. One can see the major impact of the exponentiator and the Mixture Generator on the whole area mainly due to the fact that we did not use full custom routing of such highly regular block. However, their impact on the timing delay is minimal since the hole architecture of such block is pipelined and presents a short critical paths.

Unit	Area (mm^2)	Overhead (%)
Exponentiator	1.23	46.9
Mixture Generator	0.58	22
Output FIFO	0.24	9.2
Input FIFO	0.23	8.7
KHF	0.18	6.8
Key buffer	0.15	5.6
Controller	0.005	0.33
Mangler	0.004	0.27
Frame Decoder	0.003	0.2
Total Cells Area	2.62	100

Table 6-23: PK-SSMG Core VLSI implementation (0.18um,6ML)

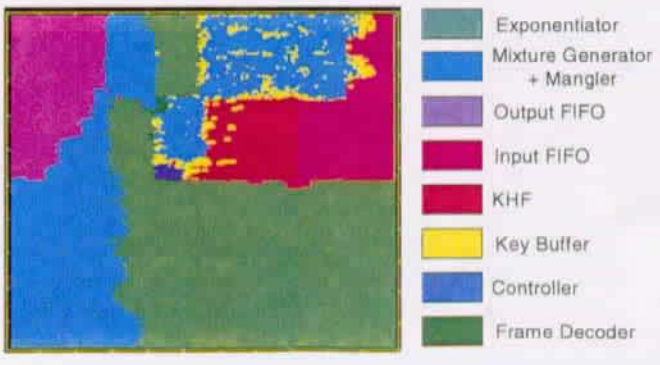


Figure 6-65: PK-SSMG Layout

CHAPTER 7

7 EFFICIENT IMPLEMENTATION OF RIJNDAEL BLOCK CIPHER

In this chapter we present an analysis of RIJNDAEL hardware architectures from the viewpoint of its realization in a FPGA. What follows is an investigation of the Rijndael block cipher to determine the nature of its underlying components. A general characteristics of the algorithm is analyzed, then the cipher structure is defined. An efficient iterative looping architecture is developed and implemented. Implementations are done in ASIC and FPGA where, SBOXes are implemented using LUT-free design. ASIC implementation, allows a faster analysis of the required area and performing a power analysis, which is not possible within an FPGA. In the other hand FPGA implementation is optimized for high-throughput. As a result, the implementation exhibit higher throughput Per Slices (TPS) factor when compared to the most known FPGAs implementations.

7.1 CHOICE OF AN ARCHITECTURE

The choice of a hardware architecture suitable for the implementation of block ciphers is influenced mainly by the required performances that can be defined by the throughput, power consumption and area. Throughput can be limited by the mode of operation and area by the cost and limit of maximum available FPGA slices.

7.1.1 MODE OF OPERATION

Symmetric-key block ciphers as described in Section 3.4.1.3 can be used in several operating modes depending on the application and the current security standards. From the point of view of hardware implementations, these modes can be classified into two major categories:

- Non-feedback (NFB) modes, such as ECB and counter mode.
- Feedback (FB) modes, such as CBC, CFB and OFB.

There are variety of suitable implementation of the architecture reported in Figure 3-4. The choice is dictated by the throughput and limited by the operating mode, see Section 7.1.3 and Section 7.1.4

7.1.2 HARDWARE PARAMETERS

Due to the nature of block ciphers algorithms, the hardware architectures most suited for their implementation depends on the hardware characteristics and parameters including speed, area and

the mutual dependence among these parameters. The most used characteristic for performance evaluation of the block cipher is the throughput that is calculated as:

$$T = (B_l \times N_b)/L \tag{7-32}$$

where, B_l is the block size, N_b is the number of blocks processed simultaneously and L is the latency or cycles required per encrypted block. Typically, the encryption and decryption throughputs are equal, and therefore only one parameter is reported.

AS FPGA resources are limited, a suitable metric for the measure of the FPGA hardware resources cost associated with an implementation's resultant throughput is the Throughput-Per-Slice (TPS), defined in [124] as:

$$TPSE = (EncryptionRate)/(numberofCLBSlacesUsed) \tag{7-33}$$

Therefore, the optimal implementation will display the highest throughput and have the largest TPS. Note that the TPS metric behaves inversely to the classical time-area product used as performance parameter metric for ASIC implementation.

7.1.3 CIPHER ARCHITECTURES FOR FEEDBACK MODES

7.1.3.1 BASIC ITERATIVE LOOPING

Iterative looping is an effective method for minimizing the hardware required. This basic hardware architecture used to implement an encryption unit of a typical secret-key block cipher is shown in Figure 7-66a. One round of the cipher is implemented as a combinational logic, and supplemented with a single register and a multiplexer. In the first clock cycle, input block of data is fed to the circuit through the multiplexer, and stored in the register. In each subsequent clock cycle, one round of the cipher is evaluated, the result is fed back to the circuit through the multiplexer, and stored in the register. The two characteristic features of this architecture are:

- Only one block of data is encrypted at a time.
- The number of clock cycles necessary to encrypt a single block of data is equal to the number of cipher rounds r.

The throughput and latency of the basic iterative architecture, throughput bi and latency bi, are given by,

$$T = (B_l \times f)/L \tag{7-34}$$

where, the latency L is equal to r in this case.

7.1.3.2 PARTIAL AND FULL LOOP UNROLLING

The partial loop unrolling architecture is depicted in Figure 7-66b. The only difference compared to the basic iterative one is that the combinational part of the circuit implements $k < r$ rounds of the cipher, instead of a single round. This architecture allows for the unrolling of multiple round. However, while this approach reduces the number of clock cycles needed to perform a block encryption, it maximizes the required hardware and the worst case register-to-register delay for the system, resulting in an extremely slow system clock.

Architecture with full loop unrolling is shown in Figure 7-66c. The input multiplexer and the feedback loop are no longer necessary, leading relatively to a small increase in the cipher speed and

decrease in the circuit are compared to the partial loop unrolling with the same number of the rounds unrolled.

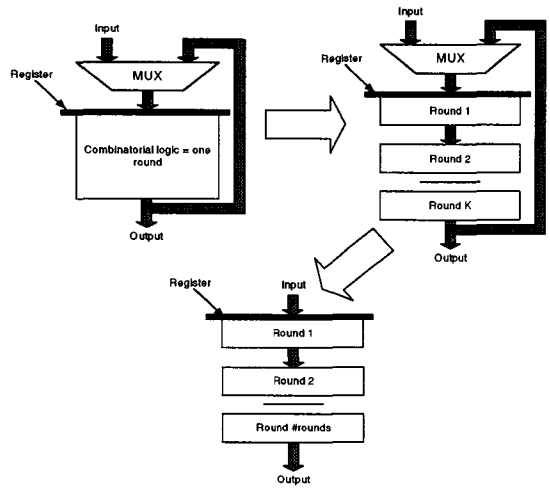


Figure 7-66: Alternative architectures of the encryption/decryption unit of a block cipher suitable for FB mode: a) basic iterative looping, b) partial loop unrolling, c) full loop unrolling

From Equation 7-34, it is shown that bigger the latency L is, smaller is the achieved throughput T using loop unrolling architecture. Basic iterative looping achieves the smallest throughput since $L=r$. Partial and full loop unrolling helps to reduce the number of clock cycles necessary to encrypt a single block of data by a factor k. At the same time the minimum clock period increases by a factor slightly smaller then k, leading to an overall relatively small increase in the encryption throughput. However, the area increases significantly as multiple round are implemented.

7.1.4 CIPHER ARCHITECTURES FOR NON-FEEDBACK MODE

Six alternative architectures that can be used for fast implementations of secret-key block ciphers operating in NFB modes are shown in Figure 6-6. These architectures can be derived into two major categories, with members of first group shown in Figure 6-6abc and members of second group shown in Figure 6-6def. Each group consists of several architectures arranged in such way that:

- The next architecture in the same category requires more area, but gives higher throughput than the previous one.
- There exist a simple transformation that lead to the subsequent architecture in each sequence.

Each group of architectures start from the basic iterative approach, discussed above, that can be chosen as an optimum architecture for the NFB mode.

7.1.4.1 PARTIAL/FULL PIPELINING

A partially pipelining architecture offers the advantage of high throughput rates by increasing the number of blocks of data that are being simultaneously operated upon. The presence of registers inside the combinational logic on the boundaries between any two subsequent cipher rounds allows the processing of k blocks of data at the same time. Each of these block is stored in a different register at the end of a clock cycle. Thus, multiple streams of data are processed and the throughput is increased by the hardware resources is maximized as shown in Figure 6-6bc.

7.1.4.2 PARTIAL/FULL PIPELINING WITH SUB-PIPELINING

A technique that can be used in order to reduce the delay between pipeline stages when the round function of the pipelined architecture is complex. Adding sub-pipeline stages as shown in allows the sub-division of the round function into smaller functional blocks. This sub-division can be performed on the round function for the iterative loop unrolling and duplicated for each stage on the pipelined architecture as shown in Figure 6-6def. The throughput and area of the circuit increase proportionally to the number of round stages *k*, This have the.... of increased latency by a factor equal to the number of sub-divisions.

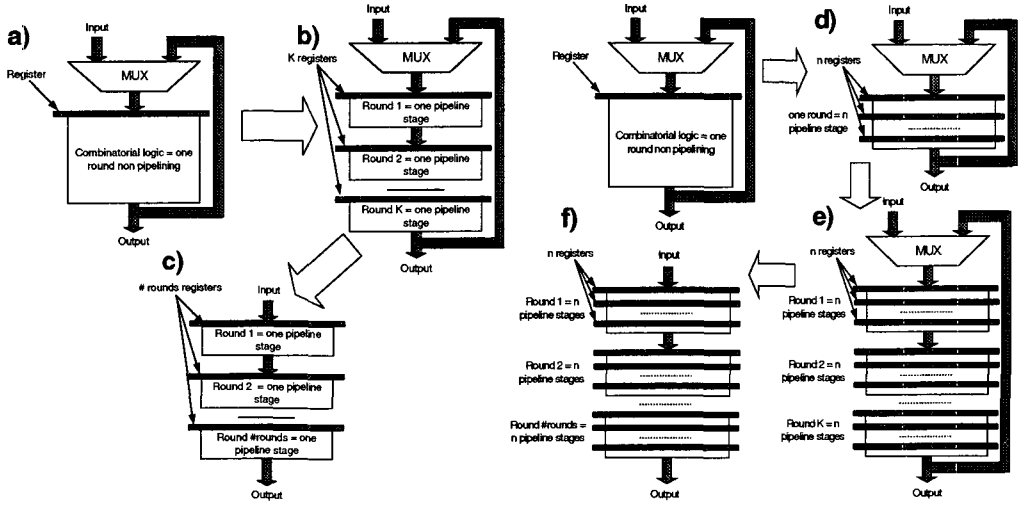


Figure 7-67: Alternative architectures of the encryption/decryption unit of a block cipher suitable for NFB mode: a) basic iterative looping, b) partial outer-round pipelining, c) full outer-round pipelining, d) inner-round pipelining, e) partial mixed inner- and outer-round pipelining, f) full mixed inner- and outer-round pipelining

7.2 STRUCTURE AND PARAMETERS OF RIJNDAEL BLOCK CIPHER

Created in 1999 by Vincent Rijmen and Joan Daemen [112]. The Rijndael round function is based on 3 underlying processes as shown in Figure 7-68:

- Linear transformation using rotations and constant multiplications.
- Non-linear transformation based on SBOXEs
- Key injection using a XORing operation

Non-linearity is provided through the use of SBOXes, which includes operations such as inversion and multiplication over $GF(2^8)$. Other operation such as rotation and combination of linear and non linear transformation guaranty a high diffusion required to build up a block cipher.

The key length K_l and the block size B_l are independent and vary in the range 128,192 and 256 bits. The number of rounds r required for the encryption of one block of data is defined by both K_l and B_l as shown in Table 7-1

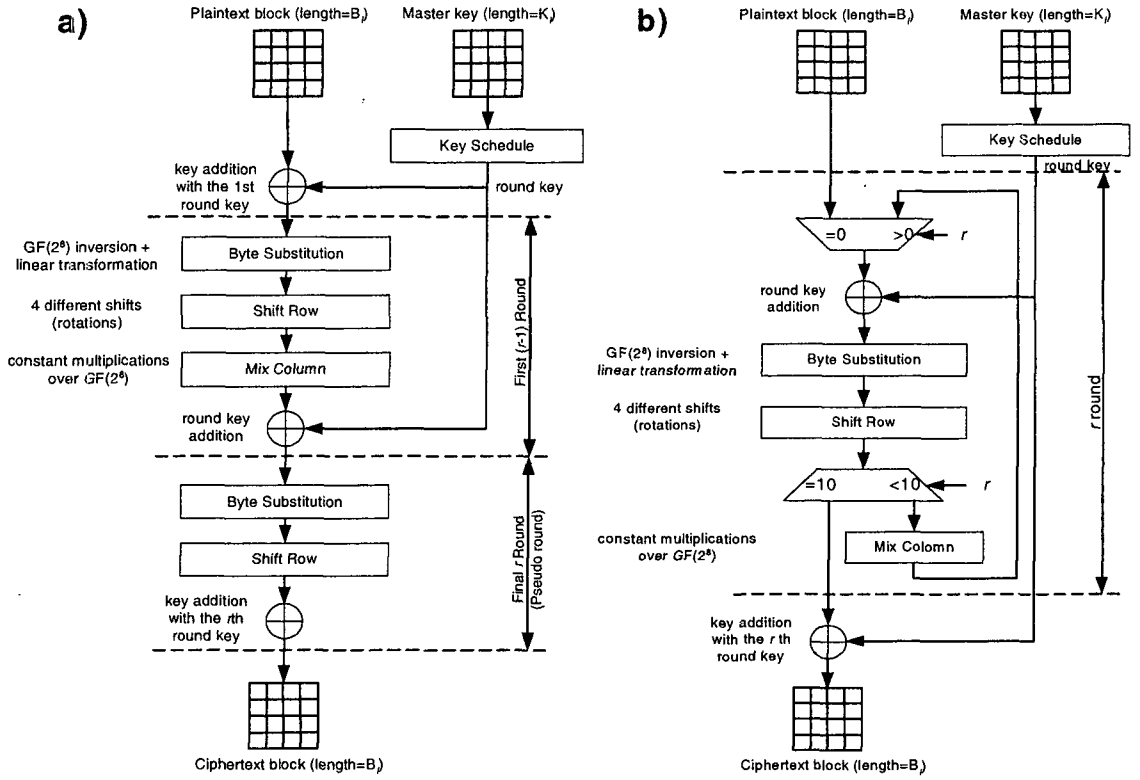


Figure 7-68: Rijndael round function structure: a) general structure, b) optimized structure

	B_f=128 bits	B_f=192 bits	B_f=256 bits
K_f=128 bits	10	12	14
K_f=192 bits	12	12	14
K_f=256 bits	14	14	14

Table 7-1: number of round r as a function of key and block length

In Section 7.1.3.2, it is shown that loop unrolling enables increasing the circuit speed in FB operating mode. Nevertheless this increase is relatively small and incurs a large area penalty. In NFB operating mode, full pipelining offers better characteristic since, it helps to reduce the latency, while operating at the same minimum clock period. The throughput is thus increased with the number of implemented round stages k as shown in Figure 7-69. However, inserting registers inside of a cipher round significantly increases the area with respect to k compared to loop unrolling scheme. As a result, the throughput to area ratio increases until the number of internal pipeline stages reaches its maximum.

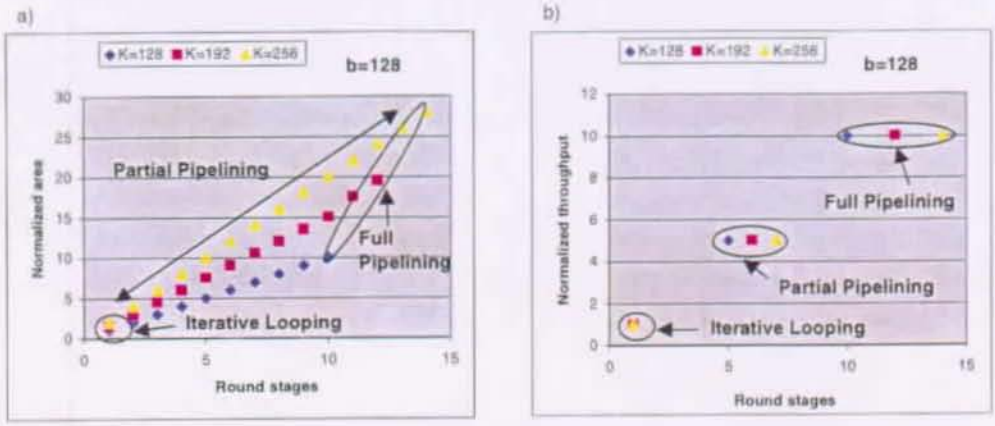


Figure 7-69: Throughput & area vs round stages

In what follows an area efficient Rijndael round function is implemented. The motivation is to minimize the number of logic required mainly by the SBOXes. The targeted architecture is the basic is the basic iterative looping since it is optimal architecture for both FB and NFB modes.

7.3 RIJNDAEL ITERATIVE LOOP UNROLLING

The round function describes exactly one round of the cipher, including encryption and decryption. The round can be run repeatedly to perform entire encryption or decryption process. The design includes a key schedule optimized implementation thus, all subkeys are computed on-chip.

7.3.1 INTERFACE

External interface is fixed to be of general purpose interface, which easily operate in a microprocessor system, and allows performing encryption with the maximum speed. It consists of a 32 bits data/key bus along with 2 control signals as shown in Figure 7-70. These control signals are used to enable encryption and decryption modes as well as the data and key loading into the circuit.

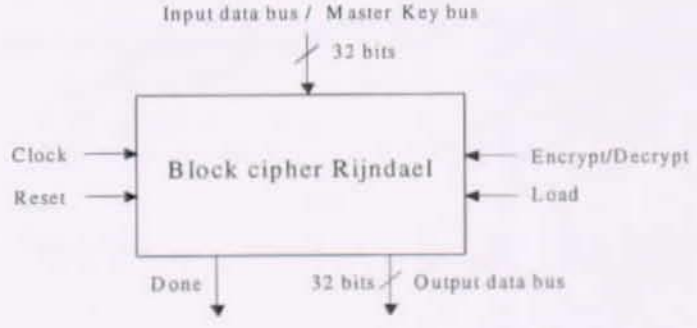


Figure 7-70: External interface

The circuit is composed of the key schedule and the round function as shown in Figure 7-71. At each iteration the round function uses a round key of 128 bits generated by the keyschedule.

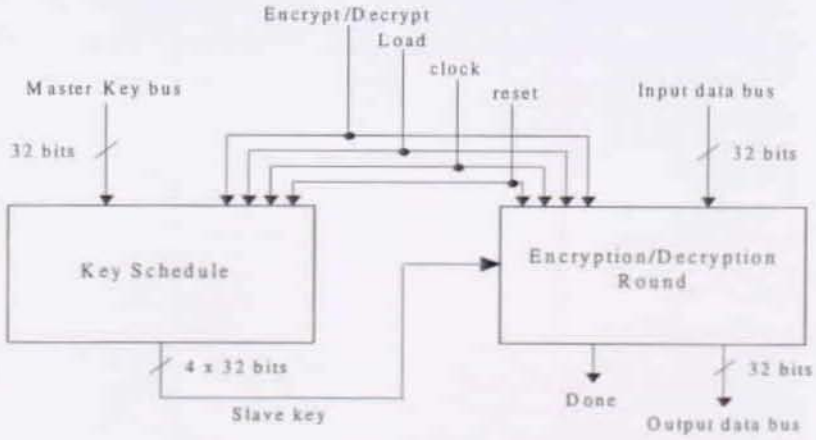


Figure 7-71: On-chip subkeys calculation

7.3.2 KEY SCHEDULE

For each round key four 32-bit subkeys are computed on chip using the bit-level structure depicted in Figure 7-72. The input key is the master key for the first round and the previous computed round keys for the later rounds. Beside the input/output stages, the expansion unit consists of byte substitution (SBOX), rotation and constant multiplication with subkeys. These components are configured to perform both encryption and decryption.

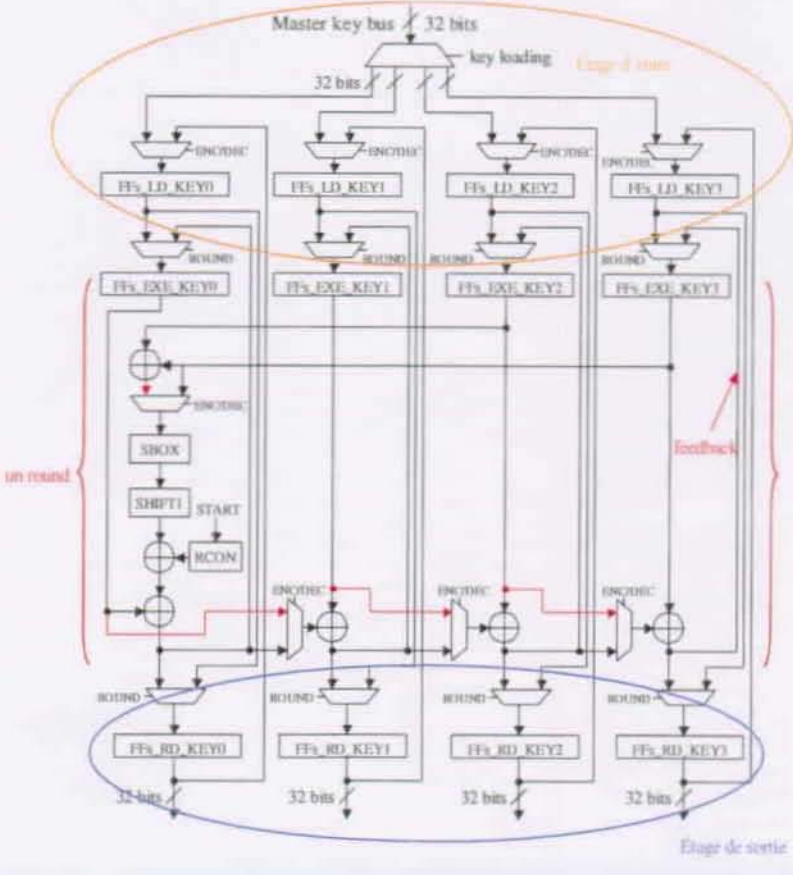


Figure 7-72: Key-schedule Block bit-level Structural Architecture

The architecture can be extended to 192 or 256 bits key length using additional branches to the right side of structure. The bit-level constant multiplications required for round keys generation during decryption, are implemented using the `MIX` components in the `FFs_RD_KEY0`,..., `FFs_RD_KEY3` modules configured as shown in Figure 7-73.

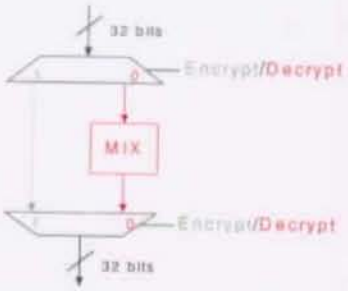


Figure 7-73: Output stage of the Key-schedule for final operation

7.3.3 DATA FLOW

The bit-level structure of Rijndael round function for encryption and decryption is shown in Figure 7-74. It can be extended to variable message lengths. Here, only a version with 128-bit message length is discussed. The datapath consists of input and output stages and four 32-bit wide branches with processing components. Input and output data are XORed with the same four subkeys `KEY0`,...,`KEY3` before and after the processing. The 32-bit data in each branch are processed using the same components operations: byte substitutions (SBOXes), rotations and constant multiplications configured to perform the required operations for both encryption and decryption.

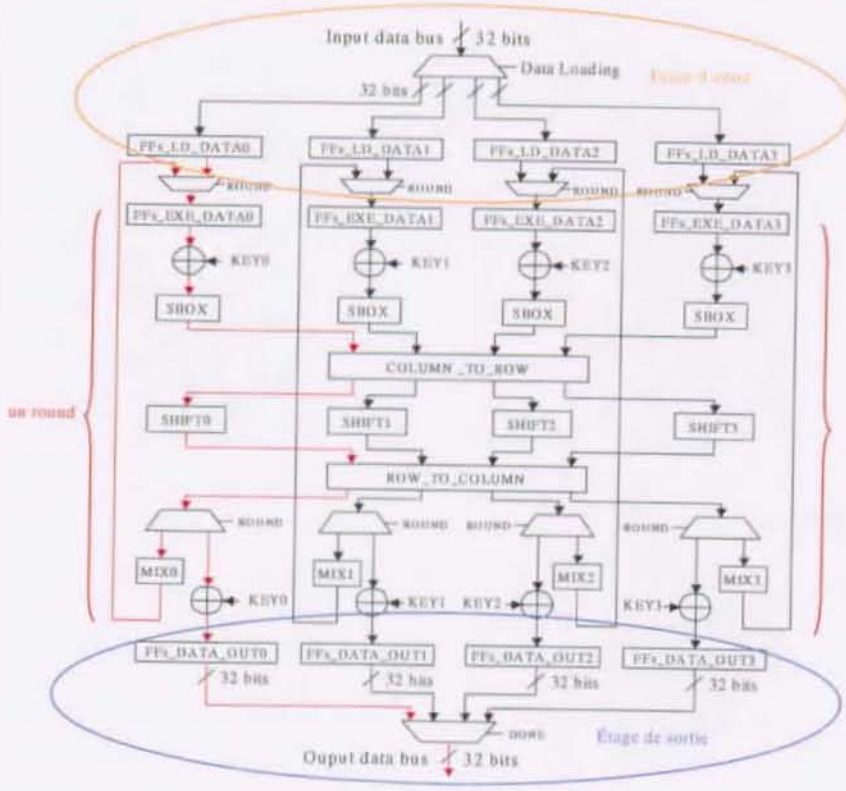


Figure 7-74: Bit-level Structural architecture of the round function

7.3.3.1 SHIFT ROW FUNCTION

Shift components are bits and bytes rotation functions. SHIFT0 is transparent component and SHIFT2 is 2 bytes right rotation. The structure of SHIFT1 and SHIFT3 components is reported in Figure 7-75. These are bit rotations as detailed in the figure.

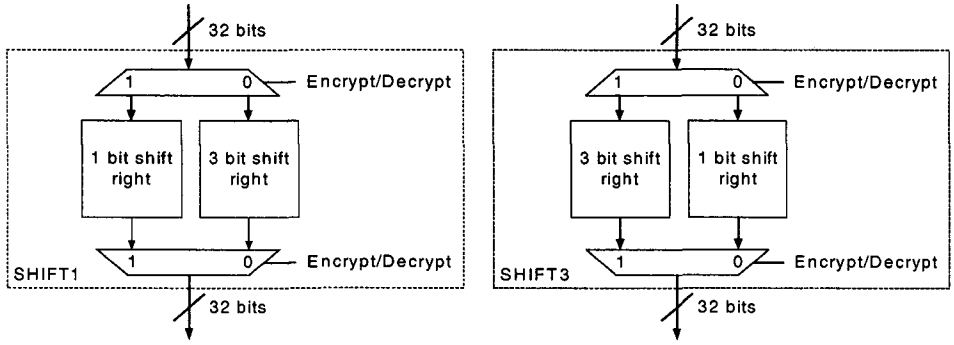


Figure 7-75: SHIFT1 and SHIFT3 components

7.3.3.2 MIX COLUMN FUNCTION

MIX components are applied on 32-bit words and they represent following matrix multiplication (encryption and decryption matrix),

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix}, \quad \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \bullet \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

These matrix multiplication are constant multiplications that are performed at `MIX` components in Figure 7-74. The underlaying structure of a `MIX` component is depicted in Figure 7-76. Multiplications are realized by bit-parallel $GF(2^8)$ multipliers and are hardwared [75] and predisposed as shown in Figure 7-76.

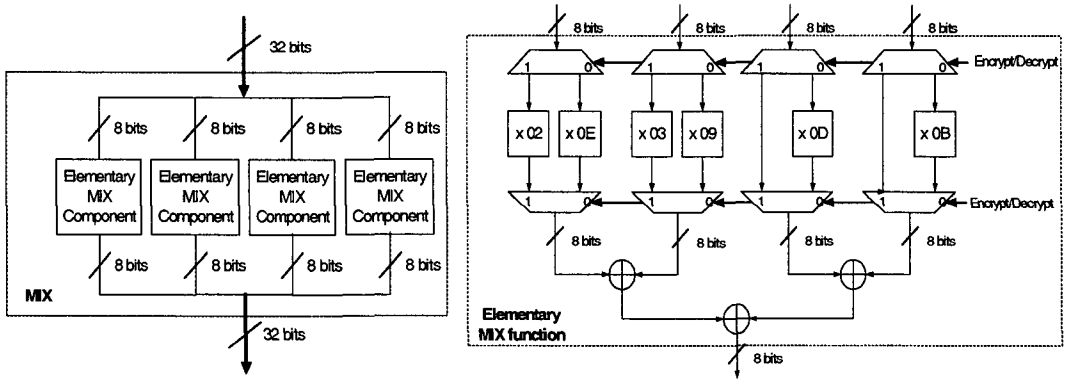


Figure 7-76: MIX and Elementary MIX component structure

7.4 RIJNDAEL SBOXES

When implementing the Rijndael algorithm, it was first determined that the SBOXes were the dominant element of the round function in terms of the required logic blocks. Each round requires 16 copies of the SBOX, implementing a combination of $GF(2^8)$ inversion with linear transformation as shown in Figure 7-77, each of which is an 8-bit to 8-bit requiring significant hardware resources. However, the remaining components of the round function including byte substitutions, rotations and constant $GF(2^8)$ multiplications, are simple in structure requiring few hardware resources.

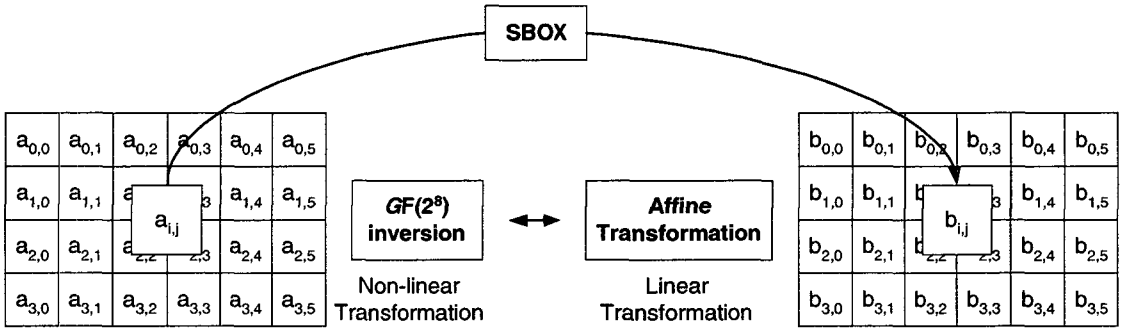


Figure 7-77: Rijndael SBOX

Inversion over $GF(2^m)$ is generally costly in term of gate count and delay. Bit-serial inverter are often used in order to reduce the hardware resources. Two solutions can be considered in order to cope with the inherent latency of bit-serial architecture,

- Using Look-Up-Table (LUT) to store the inversion results. This requires two types of LUT based SBOXes one for encryption and the other for decryption (RAM blocks organized in 8X256)
- Using bit-parallel inverters.

The usual solutions uses LUTs. Here each SBOX is an 8-bit to 8-bit LUT, requiring significant memory space. Figure 7-78 shows the amount of RAMS blocks required as function of the implemented round stages in iterative/partial loop unrolling and partial pipelined architectures.

Since there is a space-time trade-off, bit-parallel architectures are faster, which makes them attractive for constructing the SBOXes. However, the gate consumption of such architectures is to high. Meanwhile, bit-parallel inversion is possible with moderate gate count using composite fields.

In composite field, $GF(2^8)$ is decomposed into $GF[(2^4)^2]$ and the representation of $GF(2^8)$ is looked at as an extension of the field $GF(2^4)$. Therefore, the computation the parallel inversion over $GF(2^8)$ is implemented based on direct inversion in $GF(2^4)$ using the algorithm of Morri-Kasahara.

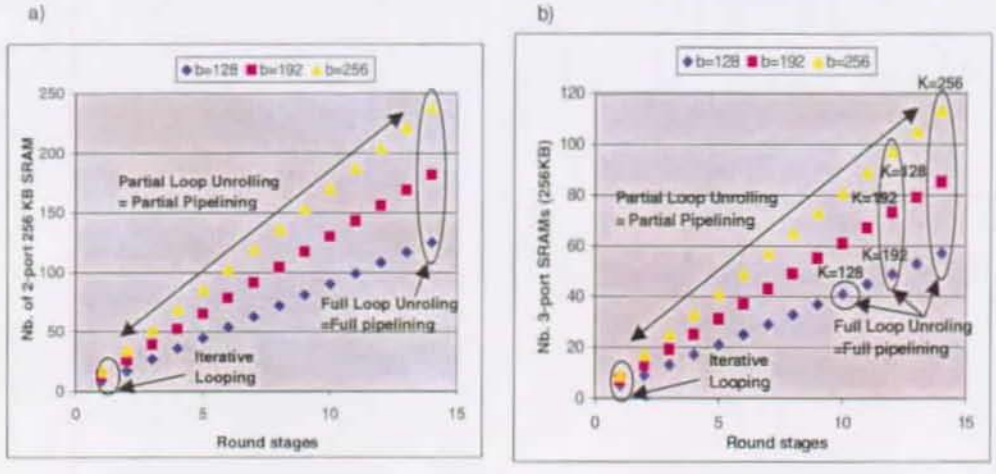


Figure 7-78: Memory requirement for Rijndael SBOXes vs round stages

7.4.1 ALGORITHM OF MORII-KASAHARA

The algorithm is described in [120] and reintroduced in [121]. It makes use of extension field of degree two.

Let us consider an element $A(x) \in GF(2^{m/2})^2)$ in canonical representation: $A(x)=a_l+a_hx$, where $a_l,a_h \in GF(2^{m/2})$. There exist always an irreducible field polynomial of the form $R(x)=x^2+x+r_0$, where $r_0 \in GF(2^{m/2})$ [122]. If the inverse of $A(x)$ is denoted as $B(x) = A^{-1}(x)= b_l+b_hx$, the equation:

$$A(x) \cdot B(x) = [(a_l b_l + a_h b_h w^{14}) + (a_l b_h + a_h b_l + a_h b_h)x] \bmod R(x) = 1 \qquad (7\text{-}35)$$

must be satisfied, which is equivalent to a set of two linear equations in b_l, b_h over $GF(2^{m/2})$ whose solution is:

$$b_l = \frac{a_l + a_h}{a_l(a_l + a_h) + w^{14} a_h^2}$$

$$b_h = \frac{a_h}{a_l(a_l + a_h) + w^{14} a_h^2} \qquad (7\text{-}36)$$

Hence, an inversion of an element in $GF(2^m)$ can be accomplished by 1 inversion, 3 general multiplications, 2 additions, 1 constant multiplication and 1 squaring, where all operations are over $GF(2^{m/2})$. The main advantage of this algorithm is that the inversion now is performed in the sub-field, which is supposed to be considerably easier that in the field $GF(2^m)$.

7.4.2 $GF(2^8)$ AS AN EXTENSION OF $GF(2^4)$

We are in particular interested in the standard representation of $GF(2^8)$ with the primitive polynomial $P(x) = 1+x+x^3+x^4+x^8$, which is used in Rijndael algorithm to perform finite field arithmetic [112] and the standard representation of $GF(2^4)$ with the primitive polynomial $Q(y) =1+y+y^4$. The field $GF(2^8)$ can also be represented as an extension of its sub-field $GF(2^4)$. We shall refer to this representation as the composite representation of $GF(2^8)$. In this case an element $A(x) \in GF(2^8)$ is represented either as a polynomial of degree 1, $A(x) =a_l+a_hx$ where, $a_l,a_h \in GF(2^4)$. Ad-

dition in $GF(2^8)$ is component-wise in $GF(2^4)$. Multiplication is polynomial multiplication modulo a primitive polynomial over $GF(2^4)$ of degree 1.

With the standard representation of $GF(2^8)$ with the primitive polynomial $P(x) = 1+x+x^3+x^4+x^8$ and the standard representation of $GF(2^4)$ with the irreducible polynomial $Q(y) = 1+y+y^4$, mentioned above, we have $GF(2^8)$ as an extension of $GF(2^4)$ with the irreducible polynomial $R(z) = w^{14}+z+z^2$, where w is a root of $P(y)$.

7.4.3 TRANSITION BETWEEN REPRESENTATIONS OF $GF(2^8)$

In the standard representation of $GF(2^8)$ each element $A(x)$ is an 8-bit binary number $A = (a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7)$. In the composite representation, the same element is a pair $(a_h a_l)$, where $a_h a_l \in GF(2^4)$. Note that $a_l=(a_{l0},a_{l1},a_{l2},a_{l3})$ and $a_h=(a_{h0},a_{h1},a_{h2},a_{h3})$ in the standard representation of $GF(2^4)$, and therefore $A=(a_{l0},a_{l1},a_{l2},a_{l3},a_{h0},a_{h1},a_{h2},a_{h3})$ in the composite representation.

The map from the standard representation $(a_0,...,a_7)$ into the composite form $(a_{l0},...,a_{h3})$ can be realized using the algorithm reported in [123], which finds linear *isomorphic* mappings between different field representations by means of matrix representation T. The inverse of T, will perform the mapping in the other direction. Here is the idea behind:

Let α be a root of $P(x)$ and $(1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7)$ is the standard base with which the elements of $GF(2^8)$ are represented. Each element of $GF(2^8)$ is thus represented as a binary of 8 vectors, denoting a linear combination of the base elements.

In order to construct the isomorphic mapping, we are looking for l base elements represented with respect to $GF[(2^4)^2]$, to which the l base elements from $(1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7)$ are to be mapped. The "one" element is mapped to the "one" element. The primitive base element α must be mapped to a primitive element β^t, the base element α^2 must be mapped to β^{2t}, and so on:

$$T\alpha^i = \beta^{it},\ i=0,1,...,7 \tag{7-37}$$

Thus the mapping is based on determining the exponent t with the following condition that assure that the mapping is homomorphic with respect to multiplication and addition:

$$P(\beta') = 0 \pmod{Q(y), R(z)} \tag{7-38}$$

There will be exactly k primitive element which fulfil this condition, namely β^t and its k-1 conjugates β^{t2^j}, $j=1,2,...,k-1$.

In our case:

$$P(\beta) = \beta w^{14} + w^5 \neq 0 \rightarrow P(\beta^i) \neq 0 \text{ for } i = 2,4,8,16,32,64,128$$
$$P(\beta^3) = 1 + w^2 + \beta(w + w^2) \neq 0 \rightarrow P(\beta^i) \neq 0 \text{ for } i = 6,12,24,48,96,192$$
$$P(\beta^5) = 0$$

Thus, α^i is mapped to $\beta^{5 \cdot 2^i}$ for $1 \leq i \leq 7$ and the transformation matrix is

:

$$\begin{pmatrix} a_{l_0} \\ a_{l_1} \\ a_{l_2} \\ a_{l_3} \\ a_{h_0} \\ a_{h_1} \\ a_{h_2} \\ a_{h_3} \end{pmatrix} = T \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{array}{c} \begin{matrix} \beta 0 & \beta 5 & \beta 10 & \beta 15 & \beta 20 & \beta 25 & \beta 30 & \beta 35 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \bullet \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \rightarrow 18\ XOR\ gates$$

This transformation can be performed with no more than 18 XOR Gates. The reverse mapping is performed using the inverse of T, which is the matrix T^{-1}:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = T^{-1} \bullet \begin{pmatrix} a_{l_0} \\ a_{l_1} \\ a_{l_2} \\ a_{l_3} \\ a_{h_0} \\ a_{h_1} \\ a_{h_2} \\ a_{h_3} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \bullet \begin{pmatrix} a_{l_0} \\ a_{l_1} \\ a_{l_2} \\ a_{l_3} \\ a_{h_0} \\ a_{h_1} \\ a_{h_2} \\ a_{h_3} \end{pmatrix} \rightarrow 24\ XOR\ gates$$

7.4.4 AN INVERTER OVER $GF(2^8)$

We consider the field $GF(2^8)$ generated by the primitive polynomial $P(x)= 1+x+ x^3+ x^4+ x^8$. For the application of Morii-Kasahara algorithm the decomposition of $GF(2^8)$ into $GF[(2^4)^2]$ is considered. Let $Q(y)=1+y+y^4$ be the primitive polynomial generating $GF(2^4)$ with $Q(w)=0$ and $R(z)=w^{14}+z+z^2$ the primitive polynomial generating the composite field. $R(x)$ is the best possible irreductible polynomial since multiplication with w^{14} requires one XOR gate see Appendix B (it is determined through an exhaustive search [123]).

For computing the inverse of an element in $GF(2^8)$ according to Equation 7-36 in hardware, the following operations, providing arithmetic in $GF(2^4)$, must be realized (see Appendix B):

- 1 inversion: 10 AND gates and 15 XOR gates
- 3 general multiplication: 48 AND gates and 62 XOR gates
- 2 additions: 8 XOR gates
- 1 constant multiplication: 1 XOR gate
- 1 squaring: 2 XOR gates

In this case multiplication in the fields $GF(2^4)$ can be performed as described in Appendix B. The corresponding multiplier is often referred to as Mastrovito multiplier. Summation of the partial complexities shows that bit-parallel inversion is possible with not more than 71 XOR and58 AND gates

7.4.5 RIJNDAEL SBOX CONFIGURATION

Each SBOX consists of four elementary SBOXes, whose outputs are configured to form a 32 bit output data. The elementary SBOX consists of an $GF(2^4)$ inversion circuit combined with an affine transformation or its inverse configured according to encryption or decryption process as shown in Figure 7-79.

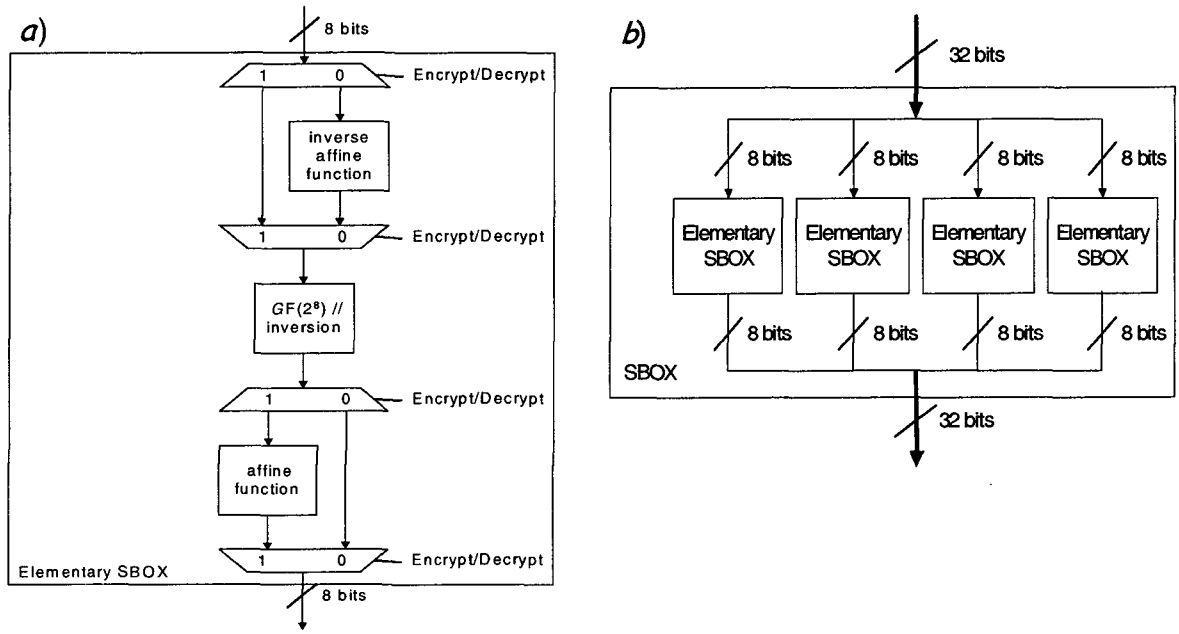


Figure 7-79: SBOX: a) structure of the elementary SBOX function, b) Rijndael SBOX component

7.5 IMPLEMENTATION RESULTS

In this section, implementation results of On-The-Fly SBOX architecture (`SBOX-OF`) regarding power consumption, area and speed of the gate level circuit are discussed. Comparison is made with the iterative looping architecture using LUTs components (`SBOX-LUT`). ASIC implementation uses the $0.18\mu m$ technology operating at 0.9v and 1.8v.

7.5.1 ASIC IMPLEMENTATION

Figure 7-80 shows the area of each component that constitutes the round function and key schedule unit for both architecture, where `RF` denotes the round function and `KS` the key schedule. One can see that the `MIX` and round function SBOXes are the predominant components. Consequently, the area is significantly reduced (about 45%) when using on-circuit $GF[(2^4)^2]$ parallel inversion.

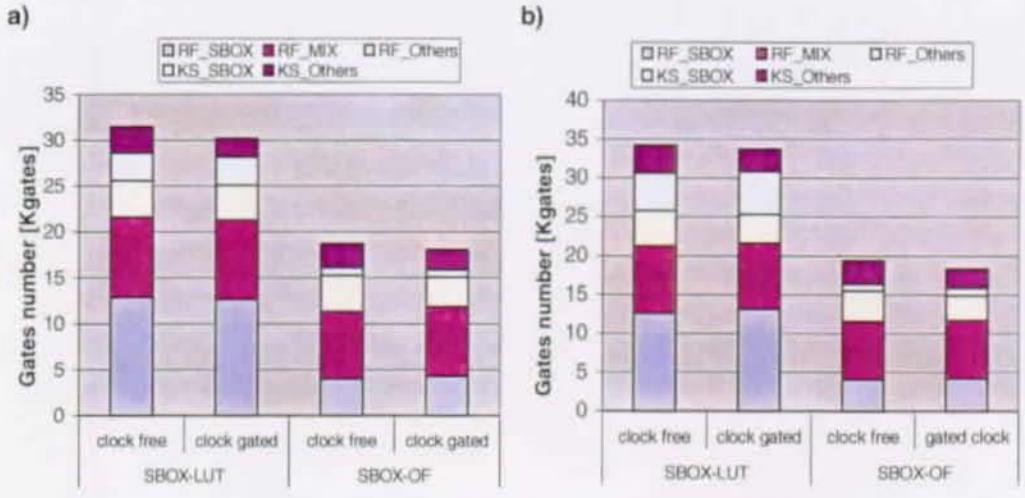


Figure 7-80: Gate count of both SBOX-LUT and SBOX-IN architectures: a) circuit operating at 0.18v, b) circuit operating at 0.9v

Although, the on-the-fly SBOX architecture minimizes the required hardware, the worst case register-to-register is increased by the combinational logic supplemented with a $GF[(2^4)^2]$ parallel inversion circuits. As shown in Figure 7-81, the induced penalty is about 14% when operating at 1.8v and 40% when operating at 0.9v. Gating the clock helps to increase slightly (2%) the throughput of the SBOX-LUT architecture with 4% decrease in the circuit area when operating at 1.8v. However, little decrease (4%) is noticed on the throughput of the SBOX-OF architecture when gating the clock due to the increase of the critical path.

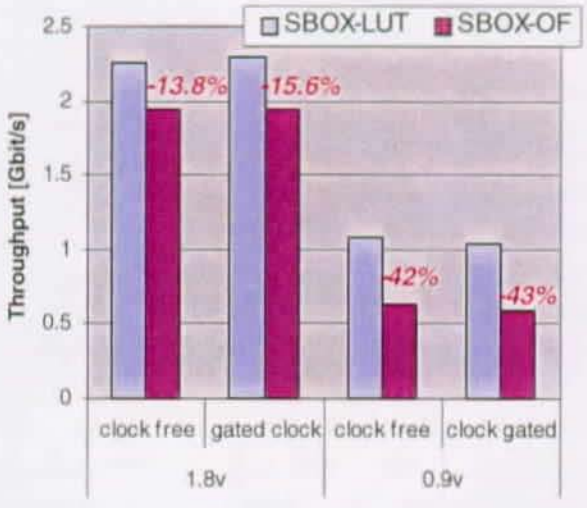


Figure 7-81: Throughput of both SBOX_LUT and SBOX-OF architectures in iterative looping configuration

Figure 7-66 shows the power consumption profiles of the above mentioned architectures. The power is estimated using gate level simulation on 1KBytes pseudo-random input for both encryption and decryption operations. It is shown that encryption and decryption are not fully symmetrical operations. In fact, before decryption r rounds are executed by the key schedule on the master key in order to recover the last round key as a master key for decryption. Beside this key setup ac-

tivity, the multiplication factors and constant operands are not the same for decryption, which adds some switching activity and increase the total dynamic power.

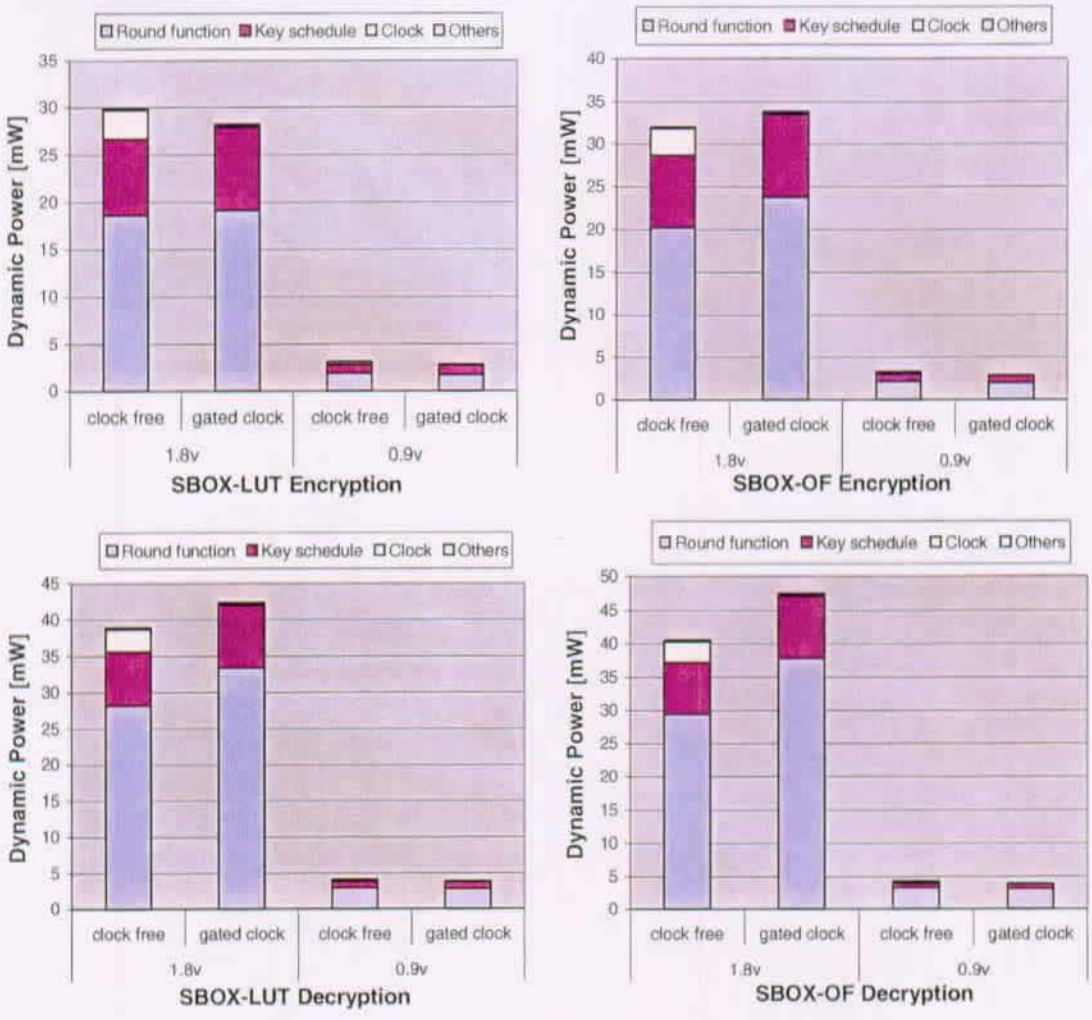


Figure 7-82: Dynamic Power Consumption

Moreover, the SBOX-OF architecture exhibits higher switching activity than SBOX-LUT and dissipates slightly more power (about 4-7%) due to the switching activity of the SBOXes. Gating the clock helps to reduce the amount of clock switching activity. However, the logic inferred by the clock gating contributes to increase the overall switching activity of the circuits, as it is often switched and not well suited when operating at 1.8v. Furthermore, while reducing the supply voltage helps to reduce considerably the consumed power ($\geq 90\%$), it is well desirable to analyze the Energy-Delay product in this case, since the critical path delay is increased. This product is computed and reported in Figure 7-83. It shows clearly that the SBOX-LUT outperforms our architecture and its best energy-optimized circuit must operate at 0.9v, while SBOX-OF architecture doesn't benefit from the supply voltage reduction.

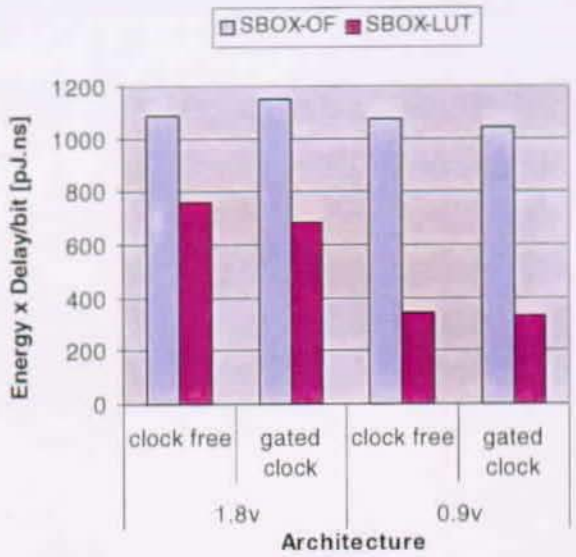


Figure 7-83: Energy-Delay Product during encryption

7.5.2 FPGA IMPLEMENTATION

The SBOX-OF architecture was implemented using XILINX Virtex FPGA XCV800. The XCV800 is a 0.22μm CMOS process, which delivers high-performance, high-capacity programmable logics operating at 2.5V supply voltage. It exploits a 5 layer-metal to perform an efficient routing required between configurable units. It has a 512 usable I/O pins and 114K bits of embedded RAM divided among 24 RAM blocks that are separate from the main body of the FPGA and acts as 56X84 Configurable Logic Blocks (CLBs). It can also provide a maximum of 301K bits of RAM independent of the embedded RAM. This amount of memory can be useful for testing the algorithm ones implemented into the device.

The round keys for both encryption and decryption were computed on the FPGA and not loaded from external key bus and stored in internal registers as the case in implementations reported in [124] and [125]. Simulation was performed for functional correctness using test vectors provided in Rijndael package [112]. The placed and routed design uses 4607 slices from the 9408 available with up to 40% of device utilization and present a maximum delay of 16.18ns limiting the system clock rate up to 60MHz. The implementation results are reported in Table 7-2. Comparison is made with implementations reported in [124], [125] and [126] using TPS and throughput metrics. It should be noted that these implementations are performed in the same Xilinx Virtex family using larger device XCV1000, which presents higher-capacity of programmable logic blocks.

Architecture	Slices	Clock Frequency [MHz]	Latency	Throughput [Mbit/s]	TPS
SBOX-OF	4607	59.94	11	697.5	151400
[Paar][a]	3528	25.3	11	294.2	83387
[Paar][b]	3488	24.9	11	290.1	83159
[GMU]	2507	35.58	11	414	165137
[USC]	4312	30.34	11	353	81864

Table 7-2: FPGA Performance evaluation of Rijndael LU-1

a. speed optimized Rijndael LU1 architecture
b. area optimized Rijndael LU1 architecture

In spite of implementing the key schedule, the proposed architecture has roughly two time higher throughput with comparable number of slices, used by the device to place and route the design, yielding to much better TPS when compared to implementations reported in [124]. The [126] design includes the key scheduling units and presents slightly better area count but lower throughput. Architecture in [125] presents higher TPS but area count includes only encryption and decryption unit and memory of internal round keys, since key schedule unit is not fully implemented.

CHAPTER 8

8 SUMMARY AND CONCLUSIONS

In this thesis we explore the state of the art in designing low-energy hardware implementing cryptographic primitives. Mainly three primitives have been investigated. the first one concern arithmetic in large Galois extension field $GF(2^m)$ mainly exponentiation, with m a large Mersenne prime to be used in public key schemes. The second primitive is Self-Synchronizing Stream Cipher SSSC and its construction from a clock-controlled LFSRs stream cipher. The resultant scheme is called Self-Synchronizing Mixture Generator SSMG and has been designed using the concept of packet fingerprint, as synchronization patterns. These patterns are generated with keyed cryptographic hash function that is well suited for hardware implementation. SSMG has been implemented in both software and hardware. Software implementation has been compared with the AES software implementation. Performance profiles have been derived in the frame of MPEG-2 selective encryption. A practical hardware cryptosystem has been implemented using the best reported design of $GF(2^m)$ exponentiator and the SSMG packet based encryption. It has been validated using DVB-SPI standard. The last part of this work deals with block ciphers and presents an efficient hardware implementation of the Rijndael algorithm, the standard that is pointed by the NIST to be a replacement for DES.

8.1 MAIN CONTRIBUTIONS

8.1.1 ARITHMETIC IN LARGE $GF(2^m)$

First and foremost, two low-energy, digit-serial architectures, suitable for large prime $GF(2^m)$ multiplication are presented. The MSR architecture is area efficient LFSR-based for trinomial polynomial field-generator and the LSA architecture is bit-level pipelined, linear systolic array architecture, which is programmable with respect to the primitive polynomial $p(x)$. Digit-serial technique when applied to the MSR architecture can be exploited efficiently, in order to decrease the critical path (total delay), when buffering the architecture and helps to reduce the switching activity for the LSA architecture at the expense of increased area. Higher gain in energy-delay product is obtained (over 90%) when digit-size is large. A trade-off can be made between the area, energy consumption and speed. No significant gain on the energy-delay product is obtained when reducing voltage supply since the delay and power counterbalance each other, when the switching power dominates. Gating the clock when possible achieves a great saving in power consumption and area and has no significant effect on the circuit speed.

A new area efficient ($\sim 17m$) exponentiator has been designed based on linear systolic array (LSA) multiplier. The architecture implements exponentiation over large $GF(2^m)$ according to square-and-multiply algorithm. These operations are overlapped in order to reduce the latency down to $2(m+1)m$ per exponentiation. The architecture relies on Standard Basis, bit-level pipelined

multiplier that is easily expendable to any field order. Also, the exponentiator is programmable with respect to the primitive polynomial $p(x)$.

In order to reduce the energy, we extended the LSA bit serial exponentiator to a generalized digit-serial architecture obtained by folding the bit-serial multiplier. The resulting circuit is array type at the digit-level using parallel multiplication algorithm inside of each digit cells. Consequently, the latency is reduced to $2(m+1)d$ where d is the number of digits and the energy delay products are significantly reduced (at the expanse of increasing area) when increasing the digit size D. The cost function defined as the energy-delay-area product versus the digit-size D has an optimum for D=4 with 55% saving of energy-delay product and more than 94% when gating the clock.

8.1.2 SELF-SYNCHRONIZING MIXTURE GENERATOR

On the other hand, while public key schemes are useful for key establishment and digital signature, they are not fast enough to encrypt bulk of data as stream and block ciphers do. Also, most of stream ciphers schemes are fast enough when implemented in hardware but lack resynchronization mechanisms to recover from canal synchronization loss SL. Adding some synchronization patterns increases the amount of information to be transmitted to the receiver and increases the bandwidth. SSSCs are well suited in this case. These schemes use public parameters such as time at the moment of resynchronization or ciphertext/plaintext patterns to recover from SL. The proposed scheme in this work (SSMG) uses keyed packet identification or packet digest as public parameter. It is computed with a KHF using a part of the encryption secret key. Though SSMG recovers from SL, it adds some workload in between packets/frames processing and increases the latency since a key setup time is required before encryption/decryption of each new frame. This is the time required to reinitilize the MG (an LFSRs based PRBG). Choosing large frame size appears to be helpful for reducing the initialization frequencies and decrease the total latency. Increasing the frame size helps also to decrease the repeated occurrences and ensure the security of the SSMG. However, within this scheme, it is shown that bit-error on the channel propagates over the following deciphered plaintext frame and thus, the frame size should be as small as possible. Security criteria of the SSMG is derived from the secret key and hash value sizes, which define the probability of secret key occurrence in the key space. Further, the hash result size is limited by the frame size and collision-free degree of the KHF. It is concluded that beside KHF collisions, there exist a trade-off between the MG LFSRs length, the frame size and the size of the hash result.

8.1.3 SECURING MPEG-2 BITSTREAM

SSMG has been implemented in software and used for real time MPEG-2 selective encryption that consist of encrypting selective amount of MPEG-2 bit stream. Performance profiles have been extracted and compared with the 5 AES finalists and other encryption stream/block ciphers. It is shown that Rijndael has the best performance among the AES ciphers and that SSMG is not very efficient in software implementations due to the bit level computations of the KHF.

A programmable PK-SSMG with respect to the key size has been implemented in hardware to secure MPEG-2 bitstream in DVB system. MPEG-2 transport stream in DVB system is frame based application with low bit-error (QEF). Further, incorporated FEC layers can detect and correct these errors, up to a reasonable limit. The cryptosystem has been interfaced with DVB-SPI and input/output data are buffered to overcome the latency incurred by the MG initialization. Moreover, some design consideration have been undertaken to reduce number of interconnections and the I/O buffer size, i.g., operating at higher frequency and avoiding locality of references using time multiplexing bus, or operating with bit/bytes granularity. Implementation results shows the energy scalability of cryptosystem with respect to the key size at a fixed data stream rate of 13.5 MBytes/s.

The granularity (encryption bit/byte) has little effect on the power consumption as only little decrease (about 1.3%) in the switching activity is noticed at the same clock frequency. The same energy is consumed during the encryption and decryption process as the system is completely symmetrical (the same resources are shared during these processes). However, gating the clock reduces significantly the amount of energy while using 3 times low-level of security helps to reduce the amount of energy consumed per bit by 17% only. The optimal solution dissipates between 200 pJ/bit (@13.5Mbytes/s, 100 MHz internal frequency) using low level of security and 400 pJ/bit at the same throughput using high level of security with ~5 times larger key length.

8.1.4 EFFICIENT IMPLEMENTATION OF RIJNDAEL BLOCK CIPHER

On the other hand, while multiplicative inverses over Galois field are well suited and used operation in cryptographic block ciphers, typically for m=8, it appears to be very difficult to find good structures for fast and low cost parallel $GF(2^m)$ inversion when $m \geq 8$. It is difficult to even find the Boolean functions defining the inverse. Nevertheless, an efficient architecture has been designed based on computations in composite field $GF[(2^8)^2]$ that helps to reduce both area and time complexities. The resultant bit-parallel inversion has been used to design an area efficient, iterative looping architecture, implementing Rijndael algorithm. First it is shown that the predominant components are the SBOXes, whose operations are simple affine (linear) operation and $GF(2^8)$ inversion as non-linear transformation. These SBOXes necessitate large amount of hardware when implemented as LUT. Thus, both round function and key schedule have been implemented using on-chip computation of $GF[(2^8)^2]$ inversion. Implementation results shows that the proposed architecture have twice smaller area than LUT based SBOXes architecture, but dissipates about 4-7% more power and present slightly lower throughput, about 14%. The real benefit is for FPGA implementation, which exhibits higher TPS factor when compared to most known Rijndael FPGA implementations.

8.2 FURTHER WORK

Practical solutions exist for designing low-energy, VLSI architectures, implementing cryptographic primitives. It is possible to reduce the hardware complexity and increase the performance of such primitives. Appropriate design methods must span the range from algorithm to hardware implementation. Power-reduce can be achieved by technology-improvements, voltages-scaling and design decision (partial parallelism, pipelining, ...). However, it is demonstrated along this work that, reducing the supply voltage is not often the best solution for low-energy design as it degrades the circuit performances when the supply voltage V_{DD} approaches the threshold voltage V_t, it is shown also that, gating the clock can be benefit for architectures whose part of logic is not often switched and area can be traded for higher throughput. Furthermore, other than classical performance evaluation and optimization mainly power and speed, it is possible at the same time to increase the security level at the cost of area. However, while efficient application of these solutions is already demonstrated, major open questions remain both for a firm theoretical foundation and the proper cryptanalysis of SSMG.

REFERENCES

[1] FIPS 46, "*Data Encryption Standard*", Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January, 1977.

[2] R.D. Silverman, "*A Cost-Based Security Analysis of Symetric and Asymetric Key Lengths*", PricewaterhouseCoopers Cryptographic Centre of Excellence (CCE), Febrary 2000.

[3] C.B. Roellgen, "*The Polymorphic Cipher*", Technical report about large keyspace cryptography.

[4] R. A. Rutenbar, "*(When) will FPGAs kill ASICS*", 38th Design Automation Conference, Las Vegas, June 18-22, 2001.

[5] D.G. Chinnery, K. Keutzer, "*Closing the Gap between ASIC and Custom: An ASIC Perspective*", Proceedings of the 37th Design Automation Conference, Los Angeles CA, pp. 637-642, June, 2000.

[6] E. S. Sinencio, A. G. Andreou, "*Low-Voltage, Low-Power Integrated Circuits and Systems*", IEEE Press, 1998.

[7] [6] R. Kanan, "*Low-Power Digital Circuit Design in GaAs MESFET Technology*", PhD thesis, EPFL, Departement of Electrical Engineering, Lausanne, Swiss, 1998.

[8] A. P. Chandrakasan, R. Brodersen, "*Low-Power CMOS Design*", IEEE Press, 1998.

[9] A. P. Chandrakasan and R. W. Brodersen "*Minimizing Power in Digital CMOS Circuits*", Proceedings of the IEEE, Vol. 83, No. 4, April 1995.

[10] Y. Taur, Y. J. Mii, D.J. Frank, H.S. Wong, D.A. Buchanan, S.J. Wind, S. A. Rishton, G.A. Sai-Halasz and E.J. Nowak, "*CMOS scaling into the 21st century: 0.1 mm and beyond*", IBM Journal of Research and Development, vol. 39, no. ½, Jan/Mar 1995, pp. 245-260.

[11] A. Bellaouar, M. Elmasry, "*Low-Power Digital VLSI design: Circuits and Systems*", Boston, Massachusetts, Kluwer Academic Publishers.

[12] L. Wei, K. Roy, V. K. De, "*Low Voltage Low Power CMOS Design Techniques for Deep Submicron ICs*", International VLSI Design, pp. 24-29, January, 2000.

[13] Jacquet Droz, "*CoolRISC 816 8-bit Microprocessor Core*", CSEM, IC Design, cr816v.1/docv3.1, Neuchatel, Switzerland, 1997.

[14] Jason Cong, Lei He, Kei-Yong Khoo, Cheng-Kok Koh, and Zhigang Pan, "I*nterconnect Design for Deep Submicron ICs*", Proc. ACM/IEEE Int'l Conf. on Computer-Aided Design (Embedded Tutorial), pp.478-485, November, 1997.

[15] Y. Ji-ren, I. Karlsson, C. Svensson, "*A true single phase clock dynamic CMOS circuit technique*", IEEE Journal of solid-State Circuits, Vol. SC-22, pp. 899-901, 1987.

[16] E. Brunvand, S. Nowick, and K. Yun, "*Practical Advances in Asynchronous Circuits and Systems*", International Conference on Computer Design (ICCD 97), pp 662-668, 1997.

[17] H.J.M Veendrick, "*Short-Circuit Dissipation of Static CMOS circuitry and its Impact on the Design of Buffer Circuits*", IEEE Journal of Solid-State Circuits, vol. SC-19, no. 4, pp. 468-473, Aug 1984.

[18] E. Macii, M. Pedram, F. Somenzi, "*High-Level Power Modeling, Estimation, and Optimization*", 34th Design Automation Conference, California, June 9-13, 1997.

[19] K. Keutzer, P. Vandekbergen, "*The Impact of CAD on the Design of Low Power Digital Circuits*", IEEE Symposium on Low Power Electronics, Oct. 10-12, San Diego, 1994.

[20] W. Diffie, M. E. Hellman, "*Exhaustive cryptanalysis of the NBS Data Encryption Standard*", Computer 10(1977), pp.74-84.

[21] M.J. Wiener, "*Efficient DES Key Search*", presented at the Rump session of Crypto '93. Reprinted in Practical Cryptography for Data Internetworks, W. Stallings, editor, IEEE Computer Society Press, pp. 31-79 (1996).

[22] M.J. Wiener, "*Efficient DES Key Search-An Update*", RSA Laboratories' CryptoBytes, Vol. 3, No. 2, pp. 6-8.
[23] W. Tuchman, "*Hellman Presents No Shortcut Solutions to DES*", IEEE Spectrum, v. 16 n. 7, July 1979, pp. 40-41.
[24] [VC] G. Vernam, "*Cipher Printing Telegraph Systems*", Transactions AIEE, Vol 45, pp 295-301.
[25] L. D. Andersen, "*Latin squares and their generalizations*", Ph.D. thesis, University of Reading, 1979.
[26] [DS] T. Ritter, "*Substitution Cipher with Pseudo-Random Shuffling: The Dynamic Substitution Combiner*", Cryptologia. 14(4), pp 289-303.
[27] T. Ritter, "*The Efficient Generation of Cryptographic Confusion Sequences*", Cryptologia, 15(2), pp.81-139, 1991.
[28] D.Wheeler, "*Problems with Chaotic Cryptosystems*" , Cryptologia, No.13, pp.243-250, 1989.
[29] D. Mitchell, "*Nonlinear Key Generators*", Cryptologia, No.14, pp.350-354, 1990.
[30] S. Wolfram, "*Random Sequence Generation by Cellular Automata*", Advances in Applied Mathematics, Vol.7, pp:123-169, 1986.
[31] L. Blum, M. Blum, M. Shub, "*A simple Unpredictable Pseudo-Random Number Generator*", SIAM Journal on Computing, Vol.15, pp.364-383, 1986.
[32] G. Marsaglia, L. Tsay, "*Matrices and the Structure of Random Number Sequences*", Linear Algebra and its applications, Vol.67, pp. 147-156, 1985.
[33] R. Rueppel, "*Analysis and design of stream ciphers*", Springer-Verlag 1986.
[34] S. Golomb, (original publication 1967), "*Shift Register Sequences*", Revised Edition. Aegean Park Press: Laguna Hills, CA, 1982.
[35] H. Beker, F. Piper, "*Cipher Systems: the protection of communication*", Wiley, 1982.
[36] E. Berlekamp, "*Algebraic Coding Theory*", Aegean Park Press: Laguna Hills, CA, 1984.
[37] J. Massey, "*Shift-Register Synthesis and BCH Decoding*", IEEE Transactions on Information Theory, vol IT-15, no. 1, januari 1969.
[38] G. Rose, "*A Stream Ciphers based on Linear Feedback Pver GF(2^n)*", ACISP'98 Proceedings Lectures Notes on Computer Science, Springer-Verlag, 1988.
[39] J.L. Massay, R.A. Rueppel, "*Linear Ciphers and Random Sequences Generators with multiples Clocks*", Advances in Cryptology Proceedings of EUROCRYPT'85, Springer-Verlag, Berlin, 1985.
[40] R.A. Rueppel, "*When Shift Registers Clock Themeselves*", Advances in Cryptology EUROCRYPT'87 Proceedings, Springer-Verlag, Berlin, 1987.
[41] S. M. Jennings, "*A Special Class of Binary Sequences*", PhD dissertation, University of London, 1980.
[42] C.G. Günther, "*A Generator of Pseudo-Random Sequences with Clock Controlled Linear Feedback Shift Register*", EUROCRYPT'87 Proceedings Lectures Notes on Computer Science, Vol. 304, Springer-Verlag, 1988.
[43] P.W.Shor, "*Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms On a Quantum Computer*", SIAM Journal on Computing, Vol. 26, No. 5, pp. 1484-1509, 1997.
[44] O. Schirokauer, D. Weber, T. Denny, "*Discrete Logarithms: the effectiveness of index Calculus Methods*", in Algorithmic Number Theory, Henri Cohen, ed. Berlin: Springer-Verlag,pp.337-331, 1996.
[45] D.P. Reiff, "*Discrete Logarithm in Finite Field*", Master thesis dissertation, University of Colorado at Boulder & Danver, 1996.
[46] D.M. Gordon, K.S. McCurley, "*Massively Parallel Computation of Discrete Logarithm*", Advances in Cryptology - Crypto'92, Lectures Notes on Computer Science, Vol. 740, pp.312-323, Springer-Verlag, New York,1993.
[47] J. Buchmann, D. Weber, "*Discrete Logarithms: Recent Progress*", in Coding Theory, cryptography and related areas, Technical Report No. TI-12/98, 26.11.1998.

[48] [3] A. M. Odlyzko, "*Discrete Logarithms in Finite Fields and their cryptographic significance*", EUROCRYPT, 1984, pp. 224-314.

[49] [19] A. M. Odlyzko, "*Discrete logarithms: The past and the future*", Designs-Codes and Cryptography 19(2/3), pp. 129-147,2000.

[50] R.L.Rivest, A.Shamir, L.M.Adleman, "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*", Communications of the ACM, Feb 1978.

[51] [Pre] B. Preneel, "*Analysis and Design of Cryptographic Hash Functions*", PhD thesis, Katholieke University Leuven, January 1993.

[52] [Bak] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, "*Cryptographic Hash Functions: A survey*", Technical Report 95-09, Department of Computer Science, University of Wollongong, July 1995.

[53] [Wol] S. Wolfram, "*Random Sequence Generation by Cellular Automata*", Advances in Applied Mathematics, 7:123-169, 1986.

[54] [Cel] J. Daemen, R. Govaerts, J. Vandewalle, "*A Framework for the design of One-Way Hash Functions including Cryptanalysis of Damgard's One-Way Function Based on Cellular Automaton* ", Advances in Cryptology, Proceedings of ASIACRYPT '91, pages 82-98, 1991.

[55] [Wol] S. Harari, "*Non-Linear, Non-Commutative Functions for Data Integrity*", Advances in Cryptology, Proceesings of EUROCRYPT '84, pages 25-32, 1985.

[56] R. L. Rivest, "*The MD5 message-digest*", Request for Commens (RFC) 1321, Internet Activities Board, Internet Privacy Task Force, april 1992.

[57] National Institute of Standards and Technology (NIST), FIPS PUB 180-1, "*Secure Hash Standard*", Federal Information, Processing Standards Publication 180-1, April 17, 1997.

[58] B. Preneel, A. Bosselaers, H. Dobbertin, "*The Ctyptographic Hash Function RIPEMD-160*", CryptoBytes, Vol. 3, No. 2, pp. 9-14, 1997.

[59] ISO/IEC 9797:1993, "*Information technology-Data cryptographic techniques-Data integrity mechanisms using a cryptographic check function employing a block cipher algorithm*".

[60] J. Daemen, R. Govaerts, J. Vandewalle, "*A hardware Design Model for Cryptographic Algorithms*", Computer Security - ESORICS'92, Proceedings 2nd European Symposium on Research in Computer Security, LNCS 648, Y. Deswarte, G. Eizenberg and J.-J. Quisquater, Eds., Springer-Verlag, 1992, pp. 419-434.

[61] I.B. Damgard, "*A Design Principle for Hash Functions*", In Advances in Cryptology, Proceedings of CRYPTO'89, pages 416-427, Springer-Verlag, 1999.

[62] P. Camion, J. Patarin, "*The Knapsack Hash Function proposed at Crypto'89 can be broken*", Advances in Cryptology, Eurocrypt'91, Proceedingd, Springer-Verlag, 1991.

[63] Bert Den Boer, Internal Report RIPE.

[64] Joan Daemen, "*Cipher and Hash Function Design, Strategies based on linear and differential cryptanalysis*", Doctoral dissertation, PhD thesis, Katholieke Universiteit Leuven, 1995.

[65] B. Preneel, P.C. van Oorshot, "*MDx-MAC and Building Fast MACs from Hash Functions*", Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995.

[66] M. Bellare, J. Kilian and P. Rogaway, "*The security of the cipher block chaining message authentication code*", in Advances in Cryptology - Crypto 94 Proceedings, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed, Springer-Verlag, 1994.

[67] B. Preneel, P.C. van Oorshot, "*On the Security of two MAC Algorithms*", Advances in Cryptology, EUROCRYPT'96, volume 1070 of Lecture Notes in Computer Science, pages 19-32. Springer-Verlag, 12-16 May, 1996.

[68] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, "*Handbook of Applied Cryptography*", CRC Press LLC, 1997.

[69] B. Schneier, "*Applied Cryptography*", Second Edition, Wiley, 1996.

[70] S. Lin, D. J. Castello, "*Error control coding, Fundamentals and applications*", Premtice-Hall, New Jersey, 1983.

[71] F.J. Mac Williams, N.J.A. Sloane, "*The theory of error correcting codes*", North-Holand, 1977.
[72] I. Stewart, "*Galois theory*", Chapman & Hall, London, 1973.
[73] E.R. Berlekamp, "*Bit-serial Reed-Solomon encoders*", IEEE Trans. Information Theory, vol. 28. pp. 120-126, Nov. 1982.
[74] J.L. Massey, J.K. Omura, "*Computational method and apparatus for finite field arithmetic*", U.S. Patent Aplication, submitted 1981.
[75] E. D. Mastrovito, "*VLSI Architectures for Computations in Galois Fields*", PhD thesis, Linköping University, Departement of Electrical Engineering, Linköping, Sweden, 1991.
[76] S. K. Jain, L. Song, K. K. Parhi, "*Optimum Primitive Polynomials for Low-Area Low-Power Finite Field Semi-Systolic Multipliers*", ECE Department, University of Minnesota, Minneapolis.
[77] D. E. Knuth, "*The Art of Computer Programming: Seminumerical Algorithms*", Volume2, Addison-Wesley, Second edition, 1981.
[78] S. K. Jain, L. Song, K. K. Parhi, "*Efficient Semi-Systolic Architectures for Finite Field Arithmetic*", IEEE Trans. on VLSI Systems, vol. 6, pp. 101-113, March 1998.
[79] I. S. Hsu, T. K. Truong, L. J. Deutsch, and I. S. Reed, "*A Comparison of VLSI Architecture of Finite Field Multipliers Using Dual, Normal, or standard Bases*", IEEE Transactions on Computers, vol. 37, no. 6, pp. 735-739, June 1988.
[80] Christof Paar, Nikolaus Lange, "*A Comparive VLSI Synthesis of Finite Field Multipliers*", Proceedings of the 3rd International Symposium on Communication Theory & Applications, 10-14 July 1995, Lake District, UK.
[81] A. G. Wassal, M. A. Hassan and M. I. Elmasry, "*Low-Power Design of finite Field Multipliers for wireless Applications*", Proceedings of the Great Lakes Symposium on VLSI '98.
[82] L. Song, K. K. Parhi, "*Low-Energy Digit-Serial/Parallel Finite Field Multipliers*", Journal of VLSI Signal Processing Systems, Vol. 19, N0. 2, Issue No. 2, pp. 149-166, June 1998.
[83] M. R. Schroeder, "*Number Theory in Science and Communication*", Volume2, Springer-Verlag, 1986.
[84] T. ElGamal, "*A public key Cryptosystem and a signature scheme based on discrete logarithms*", IEEE Transactions on Information Theory, vol IT-31(4), pp 469-472, July 1985.
[85] W. Diffie, M. E. Hellman, "*New Directions in Cryptography*", IEEE Transactions on Information Theory, 22:644-654, November 1976.
[86] William M. Rake, "*The RPK Public-Key Cryptographic System*", Technical Summary, 1993-1996.
[87] M. Kovac, N. Ranganathan, "*ACE: A VLSI Chip for Galois Field Efficient GF(2^m) Based Exponentiation*", IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 43, No. 4, pp. 189-297, April 1996.
[88] B. B. Zhou, "*A New Bit-Serial Systolic Multiplier Over GF(2^m)*", IEEE Transactions on Computers, Vol. 37, No. 6, pp. 749-751, June 1988.
[89] Chin-Liang Wang, "*Bit-Level Systolic Array for Fast Exponentiation in GF(2^m)*", IEEE Transactions on Computers, vol. 43, no. 7, pp. 838-841, July 1994.
[90] M. R. Schroeder, "*Number Theory in Science and Communication*", Vol. 2, Springer-Verlag, 1986.
[91] Ö. Egecioglu, ç. K. Koç, "*Exponentiation using Canonical Recoding*", Theorical Computer Science, 129(2): 407-417, 1994.
[92] H. Wu, A. Hasan, "*Efficient Exponentiation of a Primitive Root in GF(2^m)*", IEEE Transactions on Computers, Vol. 46, No. 2, pp. 162-172, February 1997.
[93] ç. K. Koç, T. Acar, "*Fast Software Exponentiation in GF(2^k)*", Proceedings, 13th Symposium on Computer Arithmetic,
[94] C. C. Wang et al., "*VLSI architecture for computing multiplications and inverses in GF(2^m)*", IEEE Transactions on Computers, Vol. C-34, No. 8, pp. 709-717, August 1985.
[95] P. A. Scott, S. E. Tarvares and L. E. Peppard, "*A fast multiplier for GF(2^m)*", IEEE J. Select. Areas Commun., Vol. SAC-4, Jan 1986.

[96] P. A. Scott, S. J. Simmons, S. E. Travers, L. E. Peppard, "*Architecture for exponentiation in GF(2^m)*", IEEE Transactions on Computers, Vol. 6, No. 3, pp. 578-586, April 1988.

[97] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "*Optimal normal bases in GF(p)*", Discrete Applied Mathematics, pp. 149-161, 1988/1989.

[98] S. T. J. Fenn, M. Benaissa, D. Taylor, "*GF(2^m) Multiplication and Division Over the dual Basis*", IEEE Transactions on Computers, Vol. 45, No. 3, pp. 319-327, March 1996.

[99] U. M. Maurer, "*New Approaches to the design of Self-Synchronizing Stream Ciphers*", Advances in Cryptology, EUROCRYPT'91, Lecture Notes in Computer Science, Springer-Verlag, vol. 547, pp. 458-471, 1991.

[100] T.B. Maples, G.A. Spanos, "*Performance study of a selective encryption scheme for the security of networked real-time video*", In Proc. 4th Int'l Conference on Computer and Communication, Las Vegas, NV, 1995.

[101] S. Felix Wu, T.L. Wu, "*Run-time Performance Evaluation for a Secure MPEG System Supporting Both Selective Watermarking and Encryption*", Internatonal Conference on Image Science, Systems, and Technology, CISST'97, Feb, 1997.

[102] Iskander Agi, Li Gong, "*An Empirical Study of Secure MPEG Video Transmissions*", ISOC-SNDSS, 1996.

[103] Y. Li, Z. Chen, S.M. Tan, R.H. Campbell, "*Security Enhanced MPEG Player*", NSA grant 1-5-20369 MDA 904-94C-6113.

[104] Lei Tang, "*Methods for Encrypting and Decrypting MPEG Video Data Efficiently*", Prpceedings of ACM Multimedia 96, Boston, MA, November 1996.

[105] Changgui Shi, Bharat Bhargava, "*A Fast MPEG Video Encryption Algorithm*", Electronic Proceedings, ACM Multimedia 98.

[106] Lintian Qiao, Klara Nahrstedt, "*Comparison of MPEG Encryption Algorithms*", Elsevier Science, Jan, 1998.

[107] J. Meyer, F. Gadegast, "*Security mechanisms for Multimedia-data With the Example MPEG-1 Video*", Project description SECMPEG.

[108] ISO/IEC 13818-1, "*Information technology-Generic coding of moving pictures and associated audio information: Systems*", MPEG-2 International Standard 1996.

[109] P.A. Sarginson, "*MPEG-2: Overview of the systems layer*", Research and Development Report, BBC RD, 1996.

[110] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S.M. Matyas Jr., L. O'Connor, M. Peyravian, D. Safford, N. Zunic, "*MARS - a candidate cipher for AES (complete version)*" , IBM, 1999

[111] R.L.Rivest, M.J.B. Robshaw, R. Sidney, Y.L. Yin "*The RC6 Bloc Cipher*" MIT & RSA Labs, 1999.

[112] J. Daemen, V. Rijmen, "*AES Proposal: Rijndael*", Proton World Int.l / ESAT-COSIC, 1999

[113] R. Anderson E. Biham, L. Knudsen, "*Serpent: A proposal for the AES*", Cambrige University England, Technion Haifa Israel, University of Bergen Norway.

[114] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, "*Twofish: A 128-Bit Block Cipher*", AES submission, 1998.

[115] X. Lai, J.L. Massey, "*The IDEA Bloc Cipher*", Ascom Systec Ltd. (Switzerland), 2000.

[116] ETSI/SAGE, "*Specification of the 3GPP Confidentiality and Integrity Algorithms - Document 2:KASUMI Specification*", ver 1.0, December, 1999

[117] Philip Rogaway, Don Coppersmith, "*A Software-Optimized Encryption Algorithm*", Depart. Of computer science, University of California, IBM T.J. Watson Reaserch center.

[118] "*The Algorithm (Stream cipher) RC4 (ARC4)*", **cipher designed by Ron Rivest,** web page: http://www.achtung.com/crypto/rc4.html.

[119] J. Sesena, "*The DVB satellite, cable and SMATV systems: why the technical choices were made*", EBU Technical Review, No. 266, Winter 1995.

[120] Morri and, M. Kasahara, "*Efficient construction of gate circuit for computing multiplicative inverses over GF(2^m)*", Transactions of the IEICE, vol. E 72, pp. 37-42, January 1989.

[121] C. Paar, "*Some remarks on Efficient Inversion in Finite Field*", IEEE International Symposium on Information Theory, p. 58, September 17-22, 1995.

[122] R. Lidl and H. Hiederreiter, "*Finite Fields*", vol. 20 of Encyclopedia of Mathematics and its applications. Reading, Massachusetts: Addison-Wesley, 1983.

[123] C. Paar, "*Efficient VLSI Architectures for Bit-Parallel Computation In Galois Field*", PhD desertation, Institute for Experimental Mathematics, University of Essen, Germany, 1994.

[124] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, "*An FPGA-Based Performance Evaluation of the AES Block Cipher Condidate Algorithm Finalists*", The Third Advance Encryption Standard (AES3) Condidate Conference, New York, USA, April 13-14, 2000.

[125] K. Gaj, P. Chodowiec, "*Comparison of the hardware performance of the AES condidates using reconfigurable hardware*", The Third Advance Encryption Standard (AES3) Condidate Conference, New York, USA, April 13-14, 2000.

[126] A. Dandalis, V.K. Prasanna, J.D. Rolim, "*A comparative Study of Performance of AES Final Condidates using FPGAs*", The Third Advance Encryption Standard (AES3) Condidate Conference, New York, USA, April 13-14, 2000.

APPENDIX A

A CRYPTOGRAPHY (BASICS AND LFSRS THEORY)

A.1 DEFINITIONS

A.1.1 MAXIMAL LENGTH

A linear feedback shift register (LFSR) sequence of 2^n-1 steps (assuming a bit-wide shift register of n bits). This means that every binary value the register can hold, except zero, will occur on some step, and then not occur again until all other values have been produced. A maximal-length LFSR can be considered a binary counter in which the count values have been shuffled or enciphered. While the sequence from a normal binary counter is perfectly balanced, the sequence from a maximal-length LFSR is almost perfectly balanced.

A.1.2 LINEAR COMPLEXITY

Definition1: Let $s^m = s_1 s_2 \cdots s_m$ be a binary sequence. The linear complexity $L(s^m)$ of this sequence is the size n of the smallest LFSR that can produce this sequence.

Each binary sequence generated using a finite state machine over a finite field has a finite linear complexity. The linear complexity must be increased to avoid the use of Berklamb-Massey Algorithm. Another notion is the linear complexity profile, which is the measure of the linear complexity of binary sequences at the output of an LFSR.

A.2 LFSR THEORY

The key distribution problem for One-Time Pad suggests that one might use an algorithm to generate the random sequence needed as the key (transfer of only a short seed would then be needed).

However, no algorithm using a finite state machine can produce a truly random sequence, since the finiteness forces the sequence to be periodic. The best we can do is use very long period sequences, called *pseudo-random sequences.*

What properties should a pseudo-random sequence have to make it look like a random sequence?

A.2.1 GOLOMB'S PRINCIPLES

- **G1**: The number of zeros and ones should be equal as possible per period.
- **G2**: Half the runs in a period have length 1, one-quarter have length 2, ..., $1/2^i$ have length i. Moreover, for any length, half the runs are blocks and the other half gaps. (A *block* is a subsequence of the form ...011110... and a *gap* is one of the form ...10000001..., either type is called a *run*).
- **G3**: The out-of-phase autocorrelation $AC(k)$ has the same value for all k.

$$AC(k) = \frac{Agreements - Disagreements}{p} \tag{7-39}$$

where we are comparing a sequence of period p and its shift by k places. The autocorrelation is out-of-phase if p does not divide k.

LFSRs are a commonly used method of producing pseudo-random sequences and are used in BIST, cryptography and error coding theory. An LFSR of length n (n-stage) consist of:

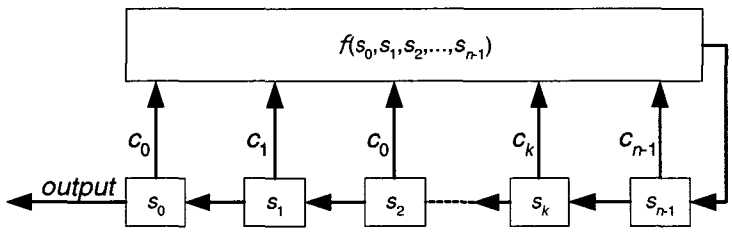


Figure A-1: LFSR structure

We first consider the case that f is a linear, i.e.,

$$f(\vec{s}) = \sum_{i=0}^{n-1} c_i s_i \tag{7-40}$$

The output of this LFSR is determined by the initial values and the linear recursion relationship:

$$s_{k+n} = \sum_{i=0}^{n-1} c_i s_{k+1}, \quad k \geq 0 \tag{7-41}$$

or equivalently

$$\sum_{i=0}^{n} c_i s_{k+i} = 0, \quad k \geq 0 \tag{7-42}$$

where c_n=1 by definition.

A sequence produced by a length n LFSR that has period 2^n-1 is called a *PN*-sequence (or a pseudo-noise sequence).We can characterize the LFSR's that produce *PN-sequences* using the *characteristic polynomial* of an LFSR as the polynomial,

$$f(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1} + x^n = \sum_{i=0}^{n} c_i x^i \tag{7-43}$$

where c_n=1 by definition and c_0=1 by assumption.

A.2.2 SOME FACTS AND DEFINITIONS FROM ALGEBRA

- Every polynomial $f(x)$ with coefficients in GF(2) having $f(0)$=1 divides x^m+1 for some m. The smallest m for which this is true is called the *period* of $f(x)$.
- An *irreducible* (can not be factored) polynomial of degree n has a period, which divides 2^n-1.
- An irreducible polynomial of degree n whose period is 2^n-1 is called a *primitive polynomial.*

Theorem: *A LFSR produces a PN-sequence if and only if its characteristic polynomial is a primitive polynomial.*

Definition: Let $\Omega(f)$ denote the set of all sequences that can be produced from an LFSR with characteristic polynomial $f(x)$.

Since each starting state produces a different (we are considering shifts as different) sequence, there are 2^n elements in $\Omega(f)$ since there are that many starting states. The sum of two sequences in $\Omega(f)$ is again in $\Omega(f)$ since the sum will satisfy the same recursion relationship (i.e., the sum corresponds to a different starting state).

We can characterize the elements of $\Omega(f)$ in terms of the *reciprocal polynomial* of f.

Definition: The reciprocal polynomial of $f(x)$ of degree n, denoted $f^*(x)$ is:

$$f^*(x) = x^n f(1/x) = c_0 x^n + c_1 x^{n-1} + \ldots + c_n$$

Note that if $f(x) = g(x)h(x)$ then $f^*(x) = g^*(x)\, h^*(x)$.

Theorem: $\Omega(f) = \{t(x)/f^*(x)$ where deg $t(x) < n\}$.

Pf: We show that each element of $\Omega(f)$ can be uniquely expressed in the desired form, and the result will follow since there are exactly 2^n binary polynomials of degree $< n$.

Let $S(x)$ be in $\Omega(f)$, where $S(x) = s_0 + s_1 x + s_2 x^2 + \ldots$ and $f^*(x) = c_0 x^n + \ldots + c^n$. Then

$$\begin{aligned} S(x)f^*(x) &= \left(\sum_{k=0}^{\infty} s_k x^k\right)\left(\sum_{i=0}^{n} c_{n-i} x^i\right) \\ &= \sum_{j=0}^{\infty}\left(\sum_{i=0}^{\min(j,n)} c_{n-i} s_{j-i}\right)\cdot x^j \\ &= \sum_{j=0}^{n-1}\sum_{i=0}^{j} c_{n-i} s_{j-i} x^j + \sum_{j\geq n}\sum_{i=0}^{n} c_{n-i} s_{j-i} x^j \\ &= \tau(x) + \sum_{j\geq n}\left(\sum_{i=0}^{n} c_i s_{(j-n)+i}\right)x^j = \tau(x) \end{aligned}$$

Lemma 1: Let $h(x)$ and $f(x)$ be the characteristic polynomials of an m-stage and respectively n-stage LFSR. Then $\Omega(h)$ is contained in $\Omega(f)$ iff $h(x)$ divides $f(x)$.

Lemma 2: Let $S(x)$ be in $\Omega(f)$ with $S(x) = t(x)/f^*(x)$. Then there exists an $h(x)$ with $h(x)$ dividing $f(x)$ and $h(x)$ not equal to $f(x)$ with $S(x)$ in $\Omega(h)$ iff $\gcd(t(x), f^*(x)) \neq 1$.

We will now consider the pseudo-randomness of *PN*-sequences.

- **G1**: Since every non-zero state appears once per period and the leftmost bit of the state is the next output value of the sequence, it is easy to see that there are 2^{n-1} ones and 2^{n-1}-1 zeros.
- **G2**: For $k \leq n-2$, a run of length k will occur in the sequence whenever the leftmost k+2 states are of the form 0111...110 or 100...001. Since all states occur, the number of each of these state sequences is 2^{n-k-2}. There is one state 011...111, and it is followed by state 111...111 since f is primitive $f(1)=1$, and that state is followed by 111...110, thus there is no block of size n-1 and one block of size n. Similarly, there is no gap of size n and only of size n-1. We can therefore calculate the number of runs as,

$$2\sum_{k=1}^{n-2} 2^{n-k-2} + 2 = 2(2^{n-2} - 1) + 2 = 2^{n-1} \tag{7-44}$$

and of these $1/2^k$ of them are of length k.

G3: Let $\{s_i\}$ be a *PN*-sequence and $\{s_{i+k}\}$ be the same sequence shifted k places. The sum of these two sequences satisfies the same recursion relation as the both of them do and so is a *PN*-sequence as well. The number of agreements in the two sequences will be the number of 0's in the sum and the number of disagreements is the number of 1's in the sum. So by **G1**,

$$AC(k) = \frac{(2^{n-1}-1)-(2^n-1)}{2^n-1} = \frac{-1}{2^n-1} \quad \text{for all } 1 \leq k \leq 2^n - 1 \tag{7-45}$$

Thus we see that *PN*-sequences satisfy all of Golomb's conditions for pseudo-randomness.

Thus, linear feedback shift registers should not be used in cryptographic work (despite this, LFSR's are still the most commonly used technique). However, this argument does not apply to non-linear FSR's so we need to examine them next.

An FSR with a possibly non-linear feedback function will still produce a periodic sequence (with a possible non-periodic beginning). If the period is p, then the LFSR with characteristic function $1+x^p$ and starting state equal to the period of the sequence, will produce the same sequence; possibly other LFSR's will also. Hence, the following definition makes sense.

Theorem: Let $S(x)$ be the generating function of a periodic binary sequence with period p. Let $S^{(p)}(x)$ be the truncated polynomial of degree p-1. Then there exists a unique polynomial $m(x)$ with

a) $S(x) \in \Omega(m)$, and

b) if $S(x) \in \Omega(h)$ then $m(x) \mid h(x)$.

$m(x)$ is called the *minimal characteristic polynomial* of $S(x)$, and

$$m^*(x) = \frac{1+x^p}{\gcd\left(S^{(p)}(x), 1+x^p\right)}$$

Proof. Let $S(x)$ be in $\Omega(m)$, but not in $\Omega(f)$ for any proper divisor f of m. We shall prove that m is unique. Note that $S(x) = S^{(p)}(x)/(1+x^p)$. Since $S(x) \in \Omega(m)$, we know that there exists a $t(x)$ with degree < degree of m, such that $S(x) = t(x)/m^*(x)$. By Lemma 2, gcd $(m^*(x), t(x)) = 1$, so

$$\gcd\left(m^*(x), \frac{S^{(p)}(x)m^*(x)}{1+x^p}\right) = 1$$

$$\gcd\left(m^*(x)(1+x^p), S^{(p)}(x)m^*(x)\right) = 1+x^p$$

$$m^*(x)\gcd\left(1+x^p, S^{(p)}(x)\right) = 1+x^p$$

$$m^*(x) = \frac{1+x^p}{\gcd\left(1+x^p, S^{(p)}(x)\right)}$$

Corollary: The linear equivalence of $S(x)$ with period p is the degree of $m(x)$ above.

We see that the use of non-linear functions does not gain any cryptographic security since we can always find a LFSR to give the same sequence. In an attempt to get this security, various means of combining the outputs of LFSR's in a non-linear way have been attempted. Clearly, sums, shifts and products of outputs don't work. Most of the information on these techniques is classified (so, someone does believe that the required security can be obtained this way). One such approach, which is in the public domain, is the multiplexing algorithm of *Jennings*.

Take an m-stage (the a_i) and an n-stage (the b_i) LFSR with primitive characteristic polynomials and non-zero starting states. Choose $h \le \min(m, \log_2 n)$ entries from the set of subscripts $\{0, 1, ..., m-1\}$ and order them $0 \le i_1 < i_2 < \ldots < i_h < fsg\ 0$. At time t, define,

$$N(t) = \sum_{j=1}^{n} a(t) \cdot 2^{j-1} \qquad (7\text{-}46)$$

Let t be any one-one mapping from $\{0,1,...,2^b-1\}$ into $\{0,1,..., n-1\}$. Define the output of the multiplexed sequence to be $u(t) = b_{N(t)}(t)$

Theorem: *if* $(m,n) = 1$ *this multiplexed sequence has period* $(2^m - 1)(2^n - 1)$.

Theorem: *If* $(m,n) = 1$ *and* $h = m - 1$, *then the sequence has linear equivalence* $n(2^m - 1)$.

A.3 DISCRETE LOGARITHM PROBLEM (DLP)

Let $\mathbf{F} = GF(q)$ and take μ as a primitive element of $\mathbf{F}$. Any c in $\mathbf{F}^*$ has a unique representation as $c = \mu^m$, for $0 \le m \le q-1$, c can be computed from μ and m with only $2\lfloor \log_2 q \rfloor$ multiplications. The binary representation of m gives the order of the needed multiplications, which consist only of squaring and multiplying by μ. For instance, if $m = 171$ then $171 = 128+32+8+2+1 = (10101011)_2$ and the computation of μ^{171} is carried out by starting with 1, then, working from the most signifi-

cant bit down, we square the current value and if there is a 1 in the binary representation we also multiply by μ. Thus, $\mu^{171} = (((((((1)^2\mu)^2)^2\mu)^2)^2\mu)^2)^2\mu)^2\mu$.

On the other hand, given c and μ, finding m is a more difficult proposition and is called the *discrete logarithm problem* (DLP).

If taking a power is of $O(t)$ time, then finding a logarithm is of $O(2^{t/2})$ time. And this can be made prohibitively large if $t = \log_2 q$ is large.

A.3.1 DIFFIE-HELLMAN KEY EXCHANGE

The difficulty of taking logarithms makes exponentiation in a finite field a one-way function (not a trapdoor function however). This can be used in a public key exchange protocol that has two system's parameters p and μ. They are both public and could be used by all the users in a system. Parameter p is a prime number and parameter $\mu < p$ (usually called a generator) is an integer with the following property: for every number n between 1 and p-1 inclusive, there is a power m of μ such that $n = \mu^m \bmod p$, where n is the public key and m is the private key. To exchange keys without transmission, A and B generate a random private exponent m_A and m_B. A looks up B's public key and exponentiates it with his own secret exponent . B does the same to A's public key. Thus, each of them calculates the same shared key value $\mu^{m_B m_A} = \mu^{m_A m_B}$, calculation done modulo the prime number p. There does not appear to be any means of obtaining this value without first finding one of the secret exponents... i.e., solving the discrete logarithm problem for this p. Diffie & Hellman suggest using a value of p which is at least few hundred bits long.

A.3.2 EL-GAMAL CRYPTOSYSTEM

For a prime p which is intractable (i.e., very large), let μ be a generator of $\mathbf{Z_p}^*$. Each user selects a secret element a in $\mathbf{Z_{p-1}}$ and makes public the value $ß = \mu^a \bmod p$. Thus, μ, ß, and p are publicly known. To send a message, Alice randomly selects a secret k in $\mathbf{Z_{p-1}}$ and if x is the message, sends the ordered pair $(\mu^k, x ß^k) \bmod p$, where ß is Bob's ß. To decrypt, Bob raises the first component to his secret exponent a, finds the inverse (mod p) of this number, and multiplies the second component by this inverse to get the message back. This computation is,

$$(x ß^k)\,(\mu^{ka})^{-1} = x ß^k (ß^k)^{-1} = x \bmod p$$

A.3.3 SHANK'S ALGORITHM FOR SOLVING THE DISCRETE LOGARITHM PROBLEM

This algorithm is known as a *Time-Memory Trade Off*, that is, if you have enough memory at your disposal you can use it to cut down the amount of time it would normally take to solve the problem.

Let p be a prime, μ a generator of $\mathbf{Zp}^*$. We wish to find a, given ß where $ß = \mu^a \bmod p$. Let $m = [(p\text{-}1)^{1/2}]$.

- Step 1: Compute $\mu^{mj} \bmod p$ for $0 \le j \le m-1$.
- Step 2: Sort the pairs $(j, \mu^{mj} \bmod p)$ by second coordinate in a list L_1.
- Step 3: Compute $ß\,\mu^{-i} \bmod p$ for $0 \le i \le m-1$.
- Step 4: Sort the pairs $(i, ß\,\mu^{-i} \bmod p)$ by second coordinate in a list L_2.

- Step 5: Find a pair in each list with the same second coordinate, i.e., (j, y) in L_1 and (i, y) in L_2.
- Step 6: $a = mj + i \bmod (p\text{-}1)$.

A.3.4 POHLIG-HELLMAN ALGORITHM

There are certain cases in which the DLP can be solved in less than $O(q^{1/2})$ time, for instance when q-1 has only small prime divisors. An algorithm for dealing with this special case was developed in 1978. We first look at a special case:

Suppose that $q - 1 = 2^n$.

Let μ be a primitive element in $GF(q)$. Noting that in this case, q is odd, we have $\mu^{(q-1)/2} = -1$. Let m, $0 \le m \le q-2$, be the exponent of μ that we wish to find, i.e. $c = \mu^m$, and write m in its binary representation: $m = m_0 + m_1 2 + m_2 2^2 + \ldots + m_{n-1} 2^{n-1}$. Now,

$$c^{\frac{q-1}{2}} = (\mu^m)^{\frac{q-1}{2}} = (\mu^{m_0 + m_1 2 + m_2 2^2 + \ldots + m_{n-1} 2^{n-1}})^{\frac{q-1}{2}} = \mu^{m_0((q-1)/2)} = \begin{cases} 1 if m_0 = 0 \\ -1 if m_0 = 1 \end{cases} \quad (7\text{-}47)$$

So the evaluation of $c^{(q-1)/2}$ which costs at most 2 $[\log_2 q]$ operations, yields m_0. We then determine $c_1 = c\,\mu^{-m_0}$, and repeat the basic computation again to obtain m_1.

$$c_1^{\frac{q-1}{4}} = (\mu^{m_1 + m_2 2 + \ldots + m_{n-1} 2^{n-2n}})^{\frac{q-1}{2}} = \mu^{m_1((q-1)/2)} = \begin{cases} 1 if m_1 = 0 \\ -1 if m_1 = 1 \end{cases} \quad (7\text{-}48)$$

This procedure can then be repeated until each of the m_i are obtained. The total number of operations is thus $n(2[\log 2q] + 2) \sim O((\log_2 q)^2)$.

The general case is dealt with by repeating the analogue of the special case for each of the prime factors of q-1 and then combining the results using the Chinese Remainder Theorem.

A.3.5 CHINESE REMAINDER THEOREM

Theorem: Suppose that $m_1, m_2, \ldots, m_r$ are pair-wise relatively prime positive integers, and let $a_1, a_2, \ldots, a_r$ be integers. Then the system of congruences, $x = a_i \pmod{m_i}$ for $1 \le i \le r$, has a unique solution modulo $M = m_1 \times m_2 \times \ldots \times m_r$, which is given by:

$$x = a_1 M_1 y_1 + a_2 M_2 y_2 + \cdots + a_r M_r y_r \pmod{M} \quad (7\text{-}49)$$

where $M_i = M/m_i$ and $y_i = (M_i)^{-1} \pmod{m_i}$ for $1 \le i \le r$.

Pf: Notice that $\gcd(M^i, m_i) = 1$ for $1 \le i \le r$. Therefore, the y_i all exist (and can be determined easily from the extended Euclidean Algorithm). Now, notice that since $M_i y_i = 1 \pmod{m_i}$, we have $a_i M_i y_i = a_i \pmod{m_i}$ for $1 \le i \le r$. On the other hand, $a_i M_j y_i = 0 \pmod{m_j}$ if j is not i (since $m_j \mid M_i$ in this case). Thus, we see that $x = a_i \pmod{m_i}$ for $1 \le i \le r$.

If there were two solutions, say x_0, and x_1, then we would have $x_0 - x_1 = 0 \pmod{m_i}$ for all i, so $x_0 - x_1 = 0 \pmod{M}$, i.e., they are the same modulo M.

Example

Find the smallest multiple of 10, which has remainder 2 when divided by 3, and remainder 3 when divided by 7.

We are looking for a number which satisfies the congruences, $x = 2 \bmod 3$, $x = 3 \bmod 7$, $x = 0 \bmod 2$ and $x = 0 \bmod 5$. Since, 2, 3, 5 and 7 are all relatively prime in pairs, the Chinese Remainder Theorem tells us that there is a unique solution modulo 210 (=2×3×5×7). We calculate the M_i's and y_i's as follows:

$M_2 = 210/2 = 105; y_2 = (105)^{-1} \pmod 2 = 1$

$M_3 = 210/3 = 70; y_3 = (70)^{-1} \pmod 3 = 1$

$M_5 = 210/5 = 42; y_5 = (42)^{-1} \pmod 5 = 3$ and

$M_7 = 210/7 = 30; y_7 = (30)^{-1} \pmod 7 = 4$.

So, $x = 0(M_2y_2) + 2(M_3y_3) + 0(M_5y_5) + 3(M_7y_7) = 0 + 2(70)(1) + 0 + 3(30)(4) = 140 + 360 = 500 \bmod 210 = \mathbf{80}$.

Remark 1: The theorem is valid in much more general situations than we have presented here.

Remark 2: The condition given is sufficient, but not necessary for a solution. Necessary and sufficient conditions exist but we are not presenting them.

Remark 3: It is purported that Sun Tsu was aware of this result in the first century A.D.

APPENDIX B

B BIT PARALLEL ARITHMETICS IN GF(2^4)

B.1 REPRESENTATIONS OF GF(2^4)

Power of α	Standard basis $\{1, \alpha, \alpha^2, \alpha^3\}$	Normal basis $\{1, \alpha^3, \alpha^2, \alpha\}$	Dual basis $\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^9\}$
-	0000	0000	0000
0	1000	1000	1111
1	0100	0001	1001
2	0010	0010	1100
3	0001	0100	1000
4	1100	1001	0110
5	0110	0011	0101
6	0011	0110	0100
7	1101	1101	1110
8	1010	1010	0011
9	0101	0101	0001
10	1110	1011	1010
11	0111	0111	1101
12	1111	1111	0010
13	1011	1110	1011
14	1001	1100	0111

Table A-1: Polynomial, dual and normal basis representations of GF(2^4), generated by the irresducible polynomial $p(x)=1+x+x^4$

B.2 STANDARD MULTIPLICATION IN GF(2^4)

The product $C(x)$ of two elements $A(x)$, $B(x) \in GF(2^4)$, $A=(a_0,a_1,a_2,a_3)$, $A=(b_0,b_1,b_2,b_3)$, is the 4-bit binary number corresponding to the coefficients of the polynomial

$$C(x)=(a_0+a_1x+a_1x^2+a_1x^3)(b_0+b_1x+b_1x^2+b_1x^3) \bmod Q(x)$$

where $Q(x)$ is the primitive polynomial used to generate the field. This can be expressed directly in matrix form as:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = A \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

where, for $Q(x)=1+x+x^4$,matrix A is given by

$$A=\begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0+a_3 & a_2+a_3 & a_1+a_2 \\ a_2 & a_1 & a_0+a_3 & a_2+a_3 \\ a_3 & a_2 & a_1 & a_0+ \end{pmatrix}$$

A straightforward implementation requires 16 AND and 15 XOR gates.

B.3 STANDARD MULTIPLICATION BY CONSTANT w^{14} IN $GF(2^4)$

The multiplication of $A(x) \in GF(2^4)$, $A=(a_0,a_1,a_2,a_3)$, with the constant element w^{14} can be accomplished by

$$w^{14}A=\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0_1 & 0 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

which requires a single XOR gate.

B.4 STANDARD SQUARING IN $GF(2^4)$

Squaring of an element $A(x) \in GF(2^4)$, $A=(a_0,a_1,a_2,a_3)$ gives

$$A^2(x)=(a_0+a_2)+a_2x+(a_1+a_3)x^2+a_3x^3 \bmod Q(x)$$

B.5 DIRECT INVERSION IN $GF(2^4)$

A direct inversion of an element $A(x) \in GF(2^4)$, $A=(a_0,a_1,a_2,a_3)$ result in the element $B(x) \in GF(2^4)$, $B=(b_0,b_1,b_2,b_3)$ given by

$b_0 = a_0+a_1+a_2+a_0a_2+a_1a_2+a_0a_1a_2+a_3+a_1a_2a_3$

$b_1 = a_0a_1+a_0a_2+a_1a_2+a_3+a_1a_3+a_0a_1a_3$

$b_2 = a_0a_1+a_2+a_0a_2+a_3+a_0a_3+a_0a_2a_3$

$b_3 = a_1+a_2+a_3+a_0a_3+a_1a_3+a_2a_3+a_1a_2a_3$

This operation can be performed with no more than 10 AND gates and 15 XOR gates.

APPENDIX C

C PERFORMANCES EVALUATION OF GF(2^{607}) MULTIPLICATION AND EXPONENTIATION

C.1 GF(2^{607}) MULTIPLICATION

Architecture	D=1		D=4		D=8		D=16	
	MSR	LSA	MSR	LSA	MSR	LSA	MSR	LSA
Supply voltage [V]	1.8	1.8	1.8	1.8	1.8	1.8	1.8	1.8
Nb. of gates	6180	11533	10505	21689	15975	44510	26323	89284
Cycle-time [ns]	48.08	0.96	25.27	1.85	13.95	3.81	11.29	7.73
Latency	1214	1821	304	456	152	228	76	114
Total Delay [µs]	58.369	1.74816	7.6821	0.8436	2.1204	0.8687	0.85804	0.8812
Cell Internal Power [mW]	2.7911	227.8638	6.2268	162.9072	11.549	81.384	13.9425	40.762
Net switching Power [mW]	1.1509	120.4	3.0432	95.8767	5.6598	48.042	7.2597	23.917
Total Dynamic Power [mW]	3.942	348.2641	9.2699	258.7839	17.209	129.43	21.2023	64.679
Cell Leakage Power [nW]	438.6591	1103	624.9595	1483.2	855.08	2381.5	1354.8	3593.2
Energy [nJ]	230.09	608.8214	71.212	218.31	36.49	112.43	18.19	56.996
Energy Delay [nJ. µs]	13430.19	1064.317	547.06	184.166	77.37	97.671	15.61	50.225

Table A-2: Specifications of MSR and LSA multipliers over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (1.8v non optimized).

Architecture	D=1	D=4	D=8	D=16
Supply voltage [V]	1.8	1.8	1.8	1.8
Nb. of gates	12140	20477	43363	87544
Cycle-time [ns]	0.99	1.82	3.55	6.92
Latency	1821	456	228	114
Total Delay [µs]	1.80279	0.82992	0.8094	0.78888
Cell Internal Power [mW]	215.6584	136.7823	65.1512	35.8156
Net switching Power [mW]	116.233	80.0537	37.9278	21.06295
Total Dynamic Power [mW]	331.8915	216.836	103.079	56.87855
Cell Leakage Power [nW]	1176.6	1513.6	2312.3	3571.3
Energy [nJ]	598.331	179.9565	83.43214	44.8704
Energy Delay [nJ. µs]	1078.665	149.35	67.52998	35.397

Table A-3: Specifications of LSA gated clock multiplier over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (1.8v non optimized).

Architecture	D=1		D=4		D=8		D=16	
	MSR	LSA	MSR	LSA	MSR	LSA	MSR	LSA
Supply voltage [V]	1.8	1.8	1.8	1.8	1.8	1.8	1.8	1.8
Nb. of gates	7225	16984	12411	23715	17891	45934	28153	90876
Cycle-time [ns]	49.99	1.1	26	1.98	14	3.91	12	7.72
Latency	1214	1821	304	456	152	228	76	114
Total Delay [µs]	60.68786	2.0031	7.904	0.90288	2.128	0.8915	0.912	0.8801
Cell Internal Power [mW]	0.86469	43.1227	1.1542	32.7358	2.8997	45.105	4.5507	27.423
Net switching Power [mW]	0.78203	76.8519	1.8933	67.3089	4.2321	39.542	5.2074	14.917
Total Dynamic Power [mW]	1.6467	119.9746	3.0475	100.0446	7.1318	84.647	9.7581	42.34
Cell Leakage Power [uW]	0.58687	1691.7	0.82667	1.9608	1.0697	2.5989	1.5289	3.7545
Energy [nJ]	99.935	240.321	24.08744	90.3283	15.176	75.461	8.8994	37.263
Energy × Delay [nJ. µs]	6064.823	481.387	190.387	81.56	32.296	67.274	8.11624	32.796

Table A-4: Specifications of MSR and LSA multipliers over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (1.8v, power optimized).

Architecture	D=1	D=4	D=8	D=16
Supply voltage [V]	1.8	1.8	1.8	1.8
Nb. of gates	15647	24583	43934	86830
Cycle-time [ns]	1.23	1.98	3.73	6.77
Latency	1821	456	228	114
Total Delay [µs]	2.23983	0.90288	0.85044	0.77178
Cell Internal Power [mW]	38.1329	28.7036	47.0138	31.625
Net switching Power [mW]	77.4811	59.6161	30.2195	12.254
Total Dynamic Power [mW]	115.614	88.3197	77.2333	43.879
Cell Leakage Power [nW]	1.7472	1.9394	2.314	3.3457
Energy [nJ]	258.956	79.7421	65.68229	33.865
Energy × Delay [nJ. µs]	580.017	71.9975	55.85885	26.1363

Table A-5: Specifications of LSA clock gated multiplier over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (1.8v, power optimized).

Architecture	D=1		D=4		D=8		D=16	
	MSR	LSA	MSR	LSA	MSR	LSA	MSR	LSA
Supply voltage [V]	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
Nb. of gates	6163	10926	11415	22294	16685	56668	34662	117496
Cycle-time [ns]	144.65	3.19	72.65	5.38	41.72	8.84	23.01	13.84
Latency	1214	1821	304	456	152	228	76	114
Total Delay [µs]	175.605	5.809	22.086	2.4533	6.3414	2.0155	1.7488	1.5778
Cell Internal Power [mW]	0.33615	26.2992	0.48499	12.5967	0.9314	9.1068	1.6373	4.213
Net switching Power [mW]	0.15125	13.4374	0.24365	7.205	0.4428	5.3202	0.83153	2.453
Total Dynamic Power [mW]	0.4874	39.7365	0.72864	19.8012	1.3742	14.427	2.4689	6.666
Cell Leakage Power [nW]	137.391	361.0628	207.0966	475.3315	278.53	959.79	509.05	1354.6
Energy [nJ]	85.5899	230.83	16.0925	48.58	8.7145	29.08	4.318	10.517
Energy × Delay [nJ. µs]	15030.02	1340.89	355.418	119.176	55.262	58.607	7.5505	16.594

Table A-6: Specifications of MSR and LSA multipliers over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (0.9v, non optimized).

Architecture	D=1	D=4	D=8	D=16
Supply voltage [V]	0.9	0.9	0.9	0.9
Nb. of gates	12140	21233	51930	102788
Cycle-time [ns]	3.18	5.7	8.78	12.32
Latency	1821	456	228	114
Total Delay [µs]	5.7908	2.599	2.002	1.4045
Cell Internal Power [mW]	24.746	10.4299	7.3051	2.213
Net switching Power [mW]	12.879	5.9461	4.1626	1.243
Total Dynamic Power [mW]	37.625	16.376	11.4677	3.456
Cell Leakage Power [nW]	373.836	470.945	895.86	1265.8
Energy [nJ]	217.88	42.5645	22.96	4.854
Energy × Delay [nJ. µs]	1261.69	110.625	45.96	6.817

Table A-7: Specifications of LSA clock gated multiplier over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (0.9v, non optimized).

Architecture	D=1		D=4		D=8		D=16	
	MSR	LSA	MSR	LSA	MSR	LSA	MSR	LSA
Supply voltage [V]	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
Nb. of gates	7265	12561	12508	25188	17704	58249	35989	124371
Cycle-time [ns]	146	3.99	73.99	6	42	9	23	14
Latency	1214	1821	304	456	152	228	76	114
Total Delay [µs]	177.244	7.266	22.493	2.736	6.384	2.052	1.748	1.596
Cell Internal Power [mW]	0.1122	10.1706	0.093721	3.3508	0.2269	2.9947	0.56499	1.736
Net switching Power [mW]	0.1085	10.1121	0.161533	6.2217	0.3254	4.7208	0.61995	1.897
Total Dynamic Power [mW]	0.221	20.2828	0.25525	9.5725	0.5524	7.7155	1.1849	3.633
Cell Leakage Power [nW]	187.132	280.0224	265.607	693.0391	336.75	1150.9	556.2718	1.4657
Energy [nJ]	39.1709	147.371	5.74133	26.1904	3.527	15.832	2.0712	5.7983
Energy × Delay [nJ. µs]	6942.8113	1070.795	129.14	71.66	22.5133	32.487	3.6204	9.254

Table A-8: Specifications of MSR and LSA multipliers over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (0.9v, power optimized).

Architecture	D=1	D=4	D=8	D=16
Supply voltage [V]	0.9	0.9	0.9	0.9
Nb. of gates	13567	23214	56249	122371
Cycle-time [ns]	3.98	6	9	14
Latency	1821	456	228	114
Total Delay [µs]	7.248	2.736	2.052	1.596
Cell Internal Power [mW]	8.8719	2.5129	2.0064	1.305
Net switching Power [mW]	9.5627	4.948	3.5416	1.5086
Total Dynamic Power [mW]	18.4346	7.4672	5.548	2.8136
Cell Leakage Power [nW]	517.3934	636.7453	1025.09	1.3466
Energy [nJ]	133.606	20.4303	11.385	4.49
Energy × Delay [nJ. µs]	968.38	55.8972	23.36	7.1668

Table A-9: Specifications of LSA clock gated multiplier over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (0.9v, power optimized).

C.2 GF(2^{607}) EXPONENTIATION

Architecture	D=1		D=4		D=8	
	clock free	clock gated	clock free	clock gated	clock free	clock gated
Supply voltage [V]	1.8	1.8	1.8	1.8	1.8	1.8
Nb. of gates	13552	12826	24516	21268	50173	46026
Cycle-time [ns]	1.46	1.86	2.85	1.55	5.09	2.15????
Latency	738113	738113	186049	186049	93633	93633
Total Delay [ms]	1.077645	1.3728902	0.53024	0.288376	0.476592	0.201311
Cell Internal Power [mW]	275.2802	244.5784	265.9354	215.1978	164.0043	263.0234
Net switching Power [mW]	136.6517	123.4497	143.2264	113.2237	100.4114	160.9728
Total Dynamic Power [mW]	411.9319	368.0281	409.1617	328.4215	264.4157	423.9962
Cell Leakage Power [µW]	1.3102	1.3626	1.6918	1.6223	2.8705	2.6665
Energy [µJ]	443.916	505.2622	216.954	94.709	126.0184	85.3551
Energy × Delay [µJ. ms]	478.384	693.6695	115.04	27.31176	60.0594	17.18292

Table A-10: Specifications of LSA exponentiator over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (1.8v, non optimized).

Architecture	D=1		D=4		D=16	
	clock free	clock gated	clock free	clock gated	clock free	clock gated
Supply voltage [V]	1.8	1.8	1.8	1.8	1.8	1.8
Nb. of gates	18129	16364	26308	24235	52887	45656
Cycle-time [ns]	1	1.78	3	1.72	5.08	3
Latency	738113	738113	186049	186049	93633	93633
Total Delay [ms]	0.738113	1.3138411	0.558147	0.32	0.475656	0.280899
Cell Internal Power [mW]	88.0412	64.3607	78.5737	73.5297	66.3909	102.8597
Net switching Power [mW]	92.5031	84.838	95.3351	75.2533	66.9479	114.3551
Total Dynamic Power [mW]	180.544	149.1987	173.9088	148.7829	133.3389	217.2148
Cell Leakage Power [µW]	1.8371	1.8702	2.1418	2.0297	3.1651	2.9101
Energy [µJ]	133.26211	196.02339	97.067	47.611165	63.4234	61.01542
Energy × Delay [µJ. ms]	98.3625	257.5436	54.1775	15.326	30.16772	17.13917

Table A-11: Specifications of LSA exponentiator over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (1.8v, power optimized).

Architecture	D=1		D=4		D=8	
	clock free	clock gated	clock free	clock gated	clock free	clock gated
Supply voltage [V]	0.9	0.9	0.9	0.9	0.9	0.9
Nb. of gates	15382	14053	25522	23009	58988	62394
Cycle-time [ns]	2.75	2.57	8.55	3.96	15.58	8.85
Latency	738113	738113	186049	186049	93633	93633
Total Delay [ms]	2.02981	1.8969504	1.591	0.736754	1.4588	0.8287
Cell Internal Power [mW]	31.5473	22.3263	16.2979	17.007	11.3807	13.8152
Net switching Power [mW]	15.1720	12.5613	8.3771	8.6958	7.0326	9.0955
Total Dynamic Power [mW]	46.7193	34.8875	24.675	25.7028	18.4134	22.9107
Cell Leakage Power [nW]	422.5316	460.4687	564.3649	546.8743	1023.5	1092.6
Energy [µJ]	94.831337	66.179857	39.251	18.93664	26.86151	18.985
Energy × Delay [µJ. ms]	192.4896	125.5399	62.448	13.95164	39.186	15.732

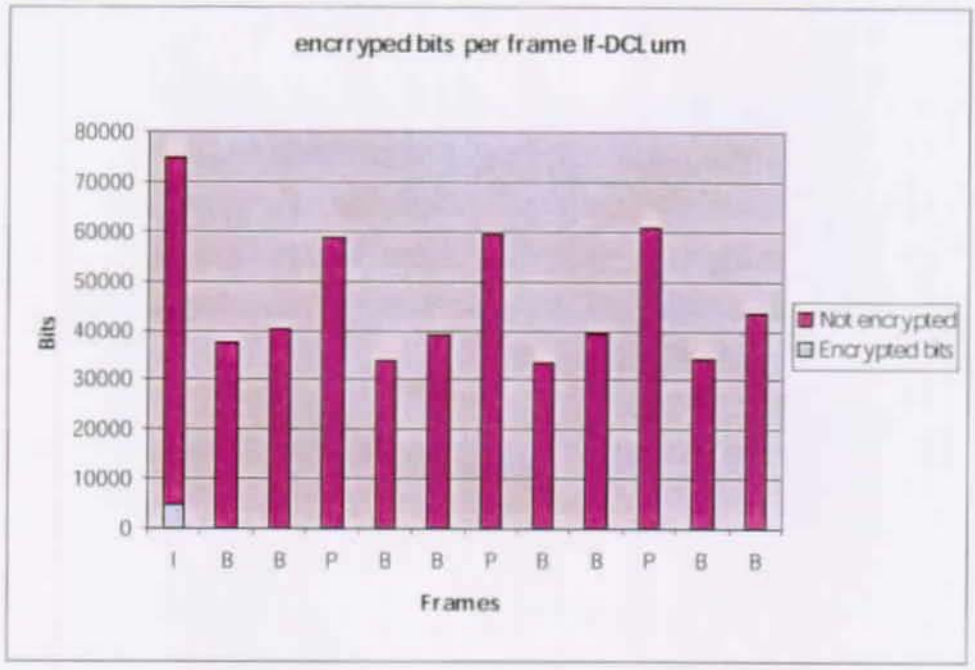
Table A-12: Specifications of LSA exponentiator over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18µm PROCESS (0.9v, non optimized).

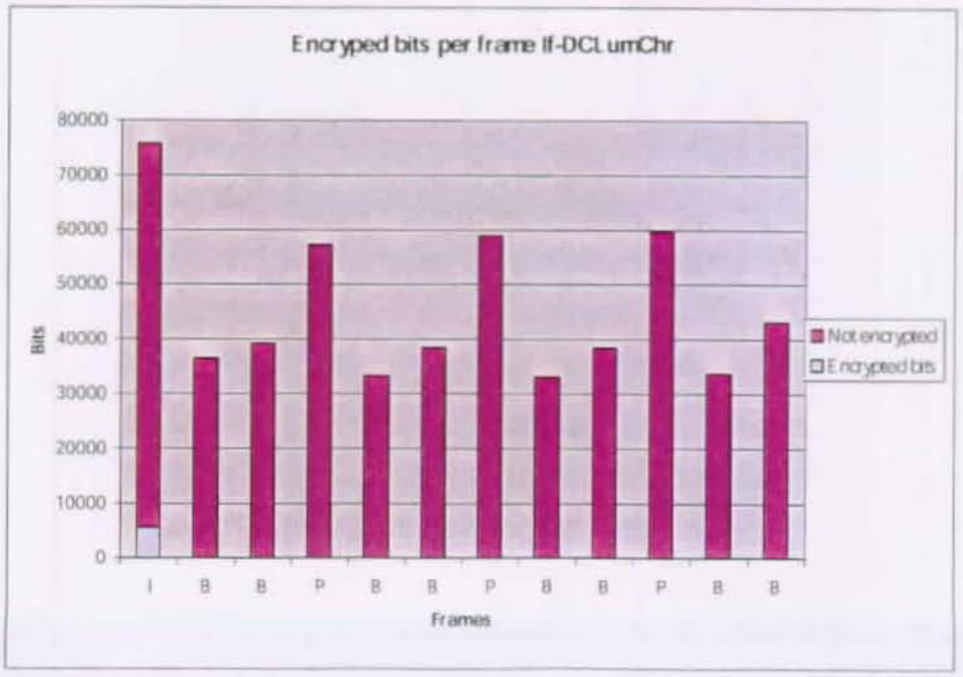
Architecture	D=1		D=4		D=8	
	clock free	clock gated	clock free	clock gated	clock free	clock gated
Supply voltage [V]	0.9	0.9	0.9	0.9	0.9	0.9
Nb. of gates	18613	15708	30108	24087	64729	59627
Cycle-time [ns]	2.53	3.91	8.46	5.96	15.48	10
Latency	738113	738113	186049	186049	93633	93633
Total Delay [ms]	1.8674259	2.8860218	1.5739745	1.1088520	1.449439	0.93633
Cell Internal Power [mW]	10.7201	7.1210	5.5338	6.1166	4.3317	5.4385
Net switching Power [mW]	11.2703	9.766	6.20008	6.6001	4.7659	6.2259
Total Dynamic Power [mW]	21.9904	16.887	11.7347	12.7167	9.0976	11.6644
Cell Leakage Power [nW]	588.9568	614.064	773.9136	701.3419	1154.6	1.1547
Energy [μJ]	41.065442	48.73625	18.470119	14.100939	13.18642	10.92173
Energy × Delay [μJ. ms]	76.68667	140.65388	29.071497	15.635855	19.11290	10.22634

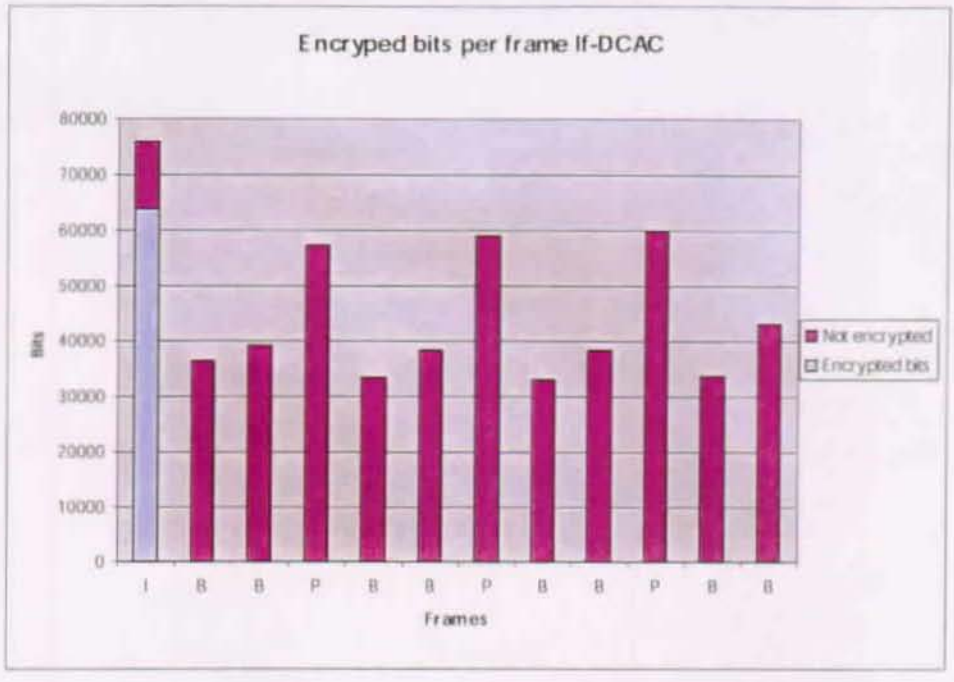
Table A-13: Specifications of LSA exponentiator over GF(2^{607}) in Low-Power/Low-Voltage COOLIB .18μm PROCESS (0.9v, power optimized).

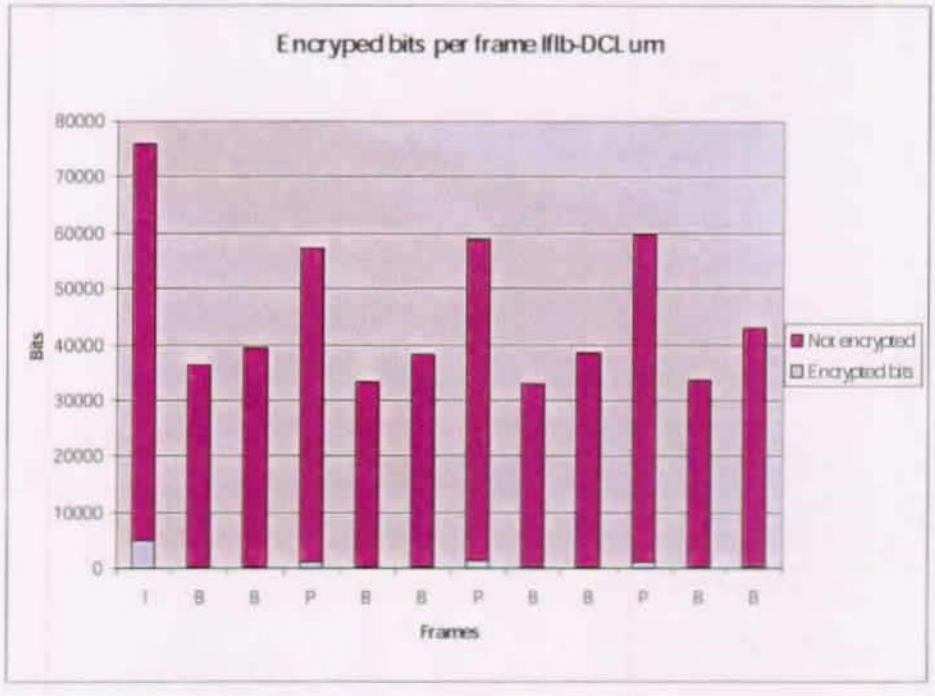
APPENDIX D

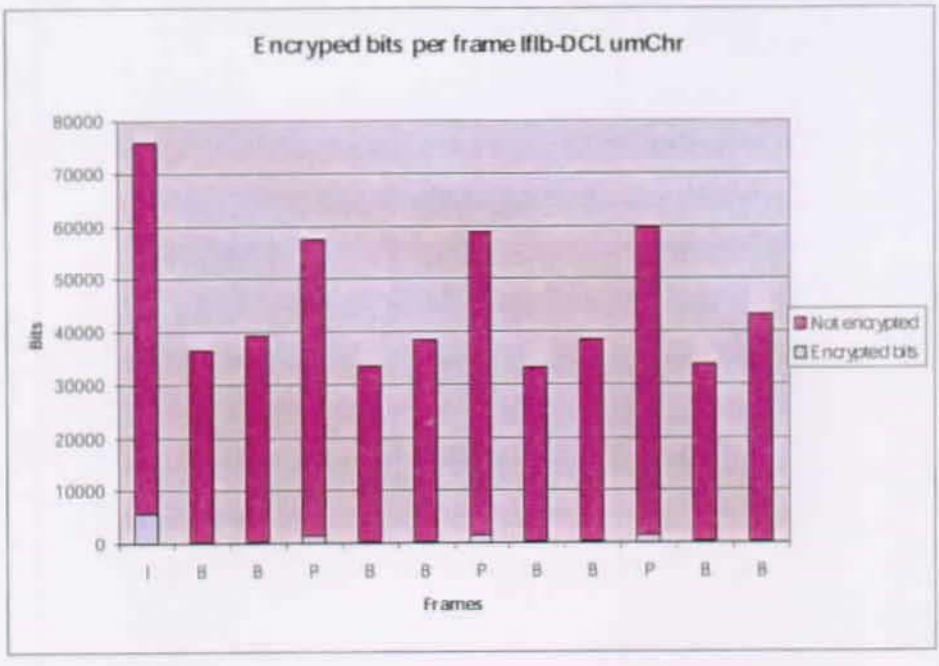
D SELECTIVE ENCRYPTION BITS FOR A SEQUENCE

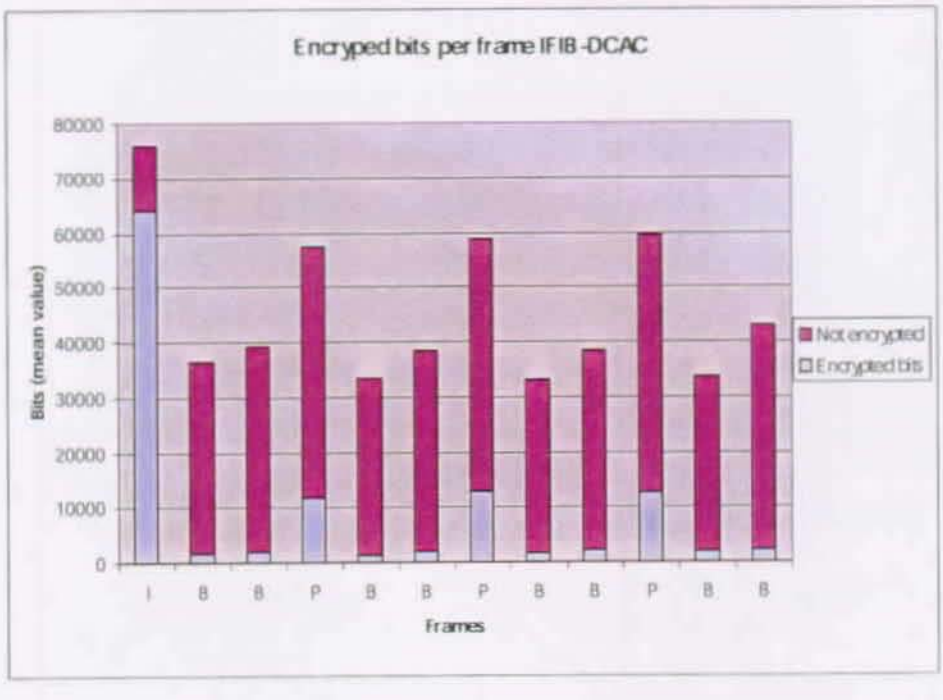


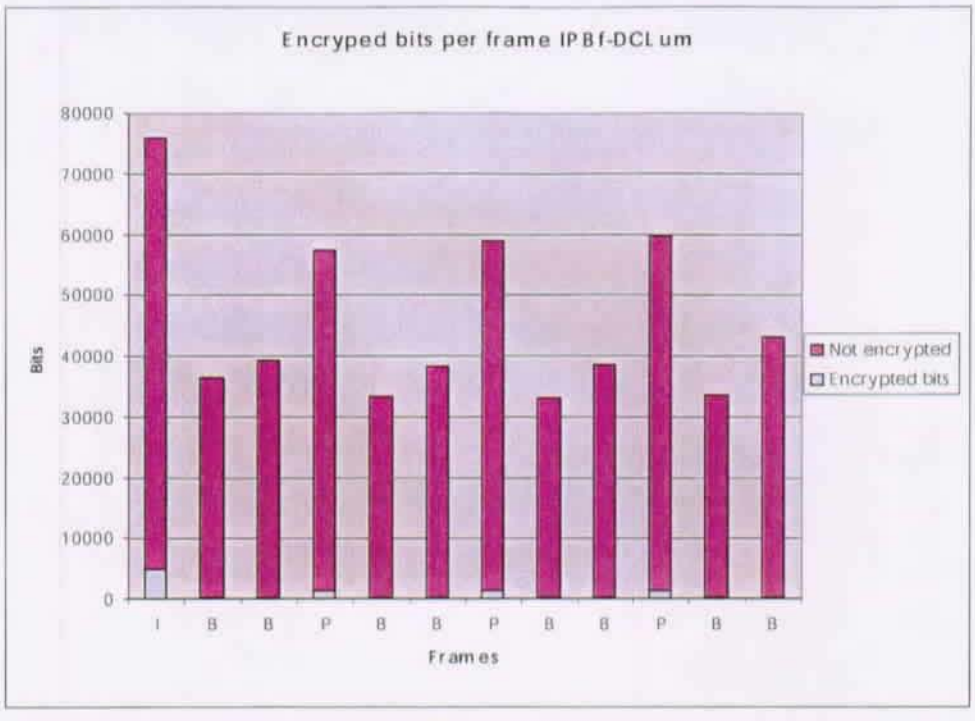


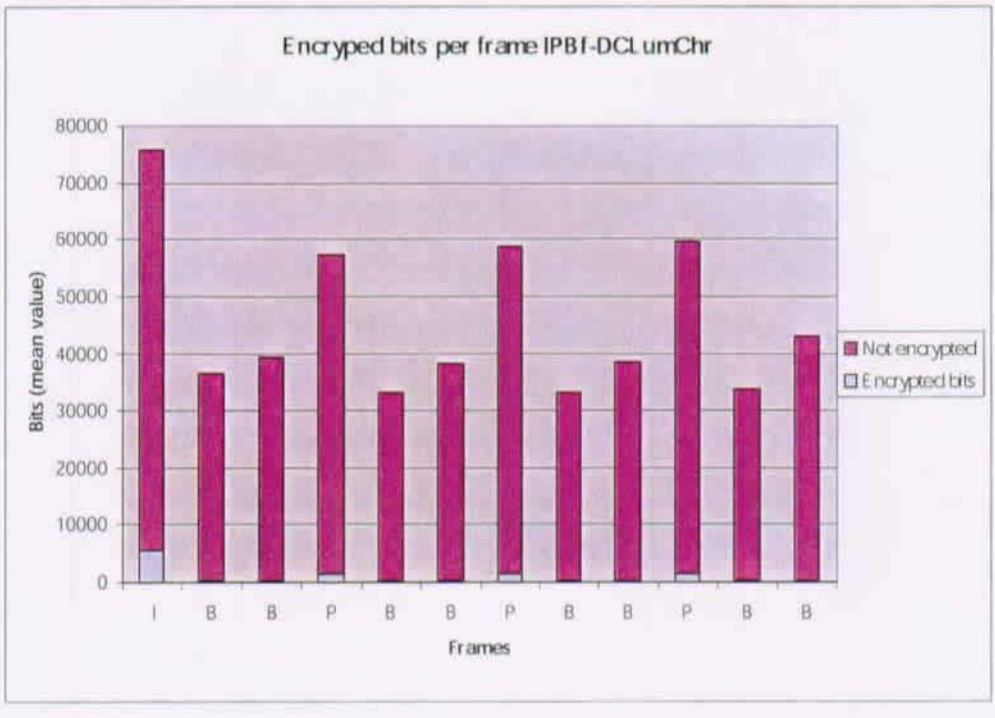

Encryped bits per frame If-DCAC
80000
70000
60000
50000
40000
30000
20000
10000
0
Bits
I B B P B B P B B P B B
Frames
Not encrypted
Encrypted bits

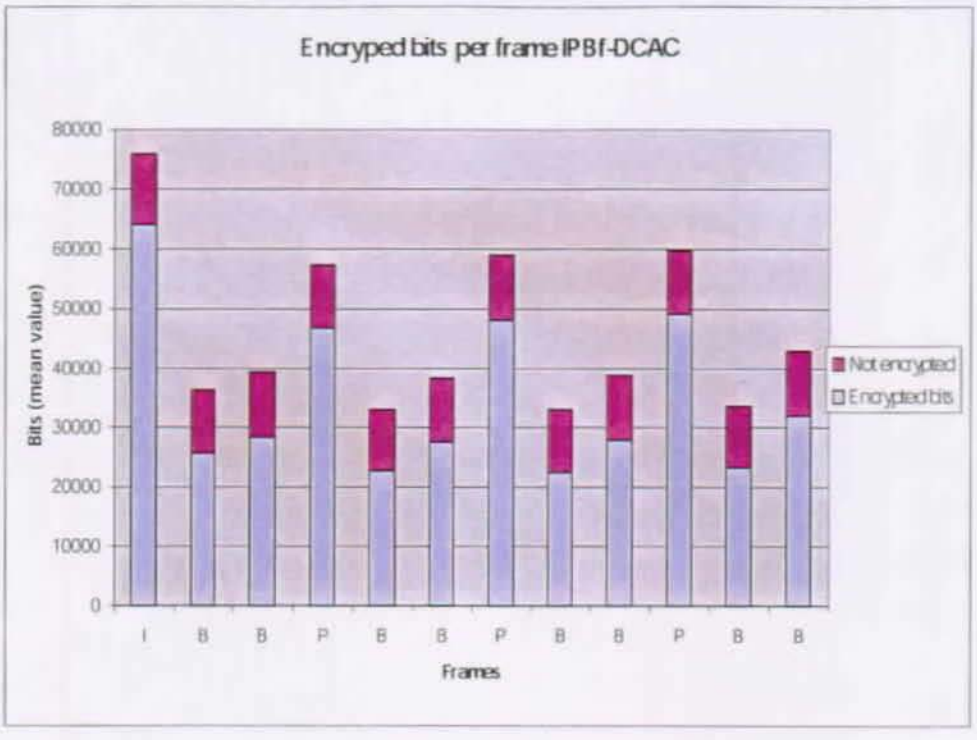

Encryped bits per frame Iflb-DCLum
80000
70000
60000
50000
40000
30000
20000
10000
0
Bits
I B B P B B P B B P B B
Frames
Not encrypted
Encrypted bits


Encryped bits per frame IFIb-DCLumChr
80000
70000
60000
50000
40000
30000
20000
10000
0
Bits
I B B P B B P B B P B B
Frames
Not encrypted
Encrypted bits

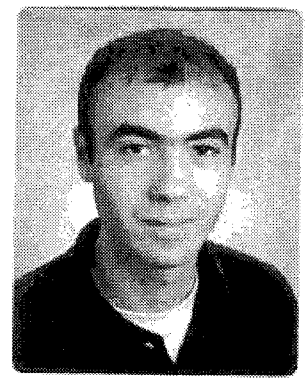

Encryped bits per frame IFIB-DCAC
80000
70000
60000
50000
40000
30000
20000
10000
0
Bits (mean value)
I B B P B B P B B P B B
Frames
Not encrypted
Encrypted bits


Encryped bits per frame IPBf-DCLum
80000
70000
60000
50000
40000
30000
20000
10000
0
Bits
I B B P B B P B B P B B
Frames
Not encrypted
Encrypted bits


Encryped bits per frame IPBf-DCLumChr
80000
70000
60000
50000
40000
30000
20000
10000
0
Bits (mean value)
I B B P B B P B B P B B
Frames
Not encrypted
Encrypted bits


Encryped bits per frame IPBf-DCAC
Bits (mean value)
80000
70000
60000
50000
40000
30000
20000
10000
0
I B B P B B P B B P B B
Frames
Not encrypted
Encrypted bits

CHERIGUI Feth Allah



Born: 15/05/1973 Algiers
Nationality: Algerian
Status: Single
Nb. child: 0

Diploma title: MS-EE

Degree Level/Status: PhD candidate

Education

- Nov. 1999-present. **PhD Candidate in Electrical Engineering**, Swiss Federal Institute of Technology, LSI Lab. Thesis: "High-performance On-Chip solutions implementing strong cryptographic primitives"
- Jan.1999-Apr. 1999. **Postgrade Course**, Swiss Federal Institute of Technology.
  "Intelligent Interfaces & Systems"
- Sep. 1997-Mar. 1998. **M.S. in Electrical Engineering**, Swiss Federal Institute of Technology, Diploma thesis: "Hardware development of RISC processor dedicated to control an artificial neural network"
- Oct. 1993-Sep. 1997. **B.S. in Electrical Engineering**, Swiss Federal Institute of Technology.
  Degree of Electrical Engineering (in VLSI).

Professional Activity

Since 1997, he was involved in several projects and a number of ICs have been developed and implemented, among these are:

- **8-bit RISC processor** dedicated to control an **artificial, integrated, mixed analog-digital neural network** (AMS 0.8µ-CMOS, 5v), VHDL modeling and synthesis (Synopsis), P&R (Cadence Cell/Block Ensemble)
- Special RAM & controller for modular transmitter systems involving a library of reusable modules: channel encoders, source encoders, framing devices, controllers (AMS 0.8µ-CMOS, 5v), **full-custom** design, generation, extraction and verification, VHDL modeling, Synthesis, Verification (Mentor Graphics).
- ACME (**ATM Cell Multiplexing /Extracting**) ASIC used for the insertion and extraction of ATM channels from a flow of throughput equal to 2488.32 Mb/s. (AMS 0.6µ-CMOS, 3.3v), full-custom design, extraction, verification and P&R (Cadence: Silicon Ensemble), VHDL modeling and synthesis (Synopsys).
- PK-SSMG (**Public-key stream based cryptosystem**) in the class of discrete logarithm schemes, featuring high-throughput/low-power characteristic. The architecture is programmable on security and implements a security level up to 1279 bits key-length based on stream cipher including self-synchronization mechanisms and supports SPI interface according to the MPEG-2 codec application requirements (TSMC 0.18µ-CMOS, 1.8v), VHDL modeling, synthesis and verification (Synopsys), P&R (Cadence Silicon Ensemble).
- **High speed/low-power** circuits implementing **RIPEMD-160 hash function** and **Rijndael block cipher** (TSMC 0.18µ-CMOS, 0.9v & 1.8v), VHDL modeling, synthesis and verification, P&R (Cadence Silicon Ensemble)
- Software development of an electrical simulator kernel (Lex, Yacc, C, TCL-TK)
- Design of a multimedia telecommunication architecture based on ATM technology to perform transmission reliability of medical data (C)

He has been taking part in **MOTOROLA MIOS** project, whose goal is to develop hardware sub-modules for a Modular I/O Subsystem (MIOS), which is an integral module of **Motorola's CISC and 16-bit RISC** Modular Embedded Controller dedicated to the automotive applications. He was involved in RPK project whose goal is hardware specification and implementation of high throughput cryptosystem based on RPK key-exchange.

Teaching

Teaching responsibilities include VLSI practical Labs and exercises for undergraduate VLSI course, semester and diploma projects supervision and management of **EDA** (Electronic Design Automation) tools, with a particular emphasis on defining methodologies and design flows for **logic entry** including modeling, simulation and synthesis with hardware description languages (**VHDL, Verilog**) and **physical entry** including full-custom design and P&R floorplaning.

Technical Skills

Programming Languages: VHDL, Verilog, C, 68XXX Assembler, shell script, TCL-TK, Lex, Yacc, Pascal.
CAD Tools: Synopsys, Cadence, Mentor Graphics.
IC simulators: SPICE, HSPICE, Spectre, cdsSpice, Smart.
Operating Systems: UNIX (Linux, Sun Solaris/SunOs, HP U/X), Windows NT (workstation-server), MS-DOS.
Others: Frame Maker, Mathworks Matlab.

Other Skills

Languages: French (fluent), English (good working knowledge), German (basic skills), Arabic (Native)

Publications

1.Feth Allah Cherigui, Daniel Mlynek, "Low energy digit serial architectures for large $GF(2^m)$ multiplication" IEE Proceedings – Circuits, Devices and Systems, 2001.
2.Feth Allah Cherigui, Daniel Mlynek, "A new area efficient, low power, digit-serial $GF(2^m)$ exponentiator ", submitted to IEEE Trans. on VLSI, 2001.
3. Feth Allah Cherigui, Daniel Mlynek, "MPEG-2 Stream Encryption using Strong Selective and Full Encryption" submitted to ACM, 2001.
4. Feth Allah Cherigui, Daniel Mlynek, "MPEG-2 stream encryption using PK-SSMG a low power public-key stream based scheme" IEE Transactions on Digital Systems, 2001.
5. Feth Allah Cherigui, Daniel Mlynek, "Efficient Hardware Implementation of Rijndael Block Cipher" International Journal of electronics, 2001.

Contact Information

General Wille strasse 18
CH-8002 Zurich
Switzerland
Tel.: +41 76 558 50 68
Email: Fethi.Cherigui@phonak.ch