

République algérienne démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Électronique

End-of-Study Project Report

For the degree of State Engineer in Electronics

Hardware Architecture and IP-Core Development for Real-Time Subspace-Based Audio Cryptography in an FPGA-SoC

Rached SAADANE & Mohamed Rayan AGUENAROUS

Publicly presented and defended on 27/06/2026

Composition of the jury:

President:	Mohamed Oussaid TAGHI	ENP
Examiner:	Maamar OULADJ	ENP
Supervisor:	Adel BELOUCHRANI	ENP
Co-Supervisor:	Atef MERMOUL	LDCCP/ENP
Co-Supervisor:	Djawad RAMDANI	LDCCP/ENP
Guest :	Yacine BENDRISSOU	

ENP 2026

République algérienne démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Électronique

End-of-Study Project Report

For the degree of State Engineer in Electronics

Hardware Architecture and IP-Core Development for Real-Time Subspace-Based Audio Cryptography in an FPGA-SoC

Rached SAADANE & Mohamed Rayan AGUENAROUS

Publicly presented and defended on 27/06/2026

Composition of the jury:

President:	Mohamed Oussaid TAGHI	ENP
Examiner:	Maamar OULADJ	ENP
Supervisor:	Adel BELOUCHRANI	ENP
Co-Supervisor:	Atef MERMOUL	LDCCP/ENP
Co-Supervisor:	Djawad RAMDANI	LDCCP/ENP
Guest :	Yacine BENDRISSOU	

ENP 2026

République algérienne démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Électronique

Mémoire de Projet de Fin d'Études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

**Architecture Matérielle et Développement d'un IP-Core
pour la Cryptographie Audio Temps Réel Basée sur les
Sous-Espaces sur une Carte FPGA-SoC**

Rached SAADANE & Mohamed Rayan AGUENAROUS

Présenté et soutenu publiquement le 27/06/2026

Composition du jury :

Président :	Mohamed Oussaid TAGHI	ENP
Examineur :	Maamar OULADJ	ENP
Promoteur :	Adel BELOUCHRANI	ENP
Co-Promoteur :	Atef MERMOUL	LDCCP/ENP
Co-Promoteur :	Djawad RAMDANI	LDCCP/ENP
Invité :	Yacine BENDRISSOU	

ENP 2026

ملخص

يتناول هذا المشروع التنفيذ المادي لنظام تشفير صوتي يعمل في الوقت الفعلي ومزدوج الاتجاه (Full-Duplex) بالاعتماد على التشفير القائم على الفضاءات الجزئية (SBE). تعتمد على إسقاط إشارة الكلام في فضاء جزئي للإشارة، وحصر ضجيج الإخفاء غير الخطي في الفضاء الجزئي المتعامد، مما يضمن استرجاعاً دقيقاً للمعلومات عبر إسقاط خطي واحد. تم تنفيذ النظام على لوحتي مصفوفة البوابات المنطقية القابلة للبرمجة (FPGA) من نوع Zybo Z7-20، مع دمج بنية تجمع بين خوارزمية ChaCha20 ومولد أرقام عشوائية حقيقية (RO-TRNG) لتوليد المفاتيح، والتي تم التحقق من صحتها باستخدام حزمة اختبارات المعهد الوطني للمعايير والتقنية (NIST SP-800-22). تعتمد البنية المقترحة على الحساب ذي النقطة الثابتة (Fixed-Point)، مع توظيف خوارزمية الحاسب الرقمي لدوران الإحداثيات (CORDIC) لتنفيذ العمليات الرياضية بكفاءة عالية وبتكلفة عتادية منخفضة، وقد أثبت النظام قدرته على العمل في الزمن الحقيقي عند تردد تشغيل قدره 125 MHz. كما أظهرت النتائج جودة عالية في عمليتي التشفير وفك التشفير، مع الحفاظ على دقة استرجاع الإشارة الصوتية حيث تراوحت نسبة إشارة إلى ضجيج (SNR) تتراوح بين 87 و 109 dB، بالإضافة إلى توليد إشارة مشفرة غير قابلة للتمييز عن الضوضاء البيضاء، وذات ارتباط شبه معدوم مع إشارة الكلام الأصلية.

الكلمات المفتاحية: التشفير القائم على الفضاءات الجزئية (SBE)، مصفوفة البوابات المنطقية القابلة للبرمجة (FPGA)، خوارزمية ChaCha20، مولد الأرقام العشوائية الحقيقية (TRNG)، المعهد الوطني للمعايير والتقنية (NIST)، بروتوكول I2S، الحساب بالنقطة الثابتة (Fixed-Point)، اتصال مزدوج الاتجاه (Full-Duplex).

Résumé

Ce travail présente l'implémentation matérielle d'un système de chiffrement audio en temps réel et full-duplex basé sur l'approche SBE (Subspace-Based Encryption). Le SBE projette le message clair dans un sous-espace signal et confine un bruit de masquage non linéaire dans le sous-espace orthogonal, assurant un décodage exact par projection linéaire. Déployé sur deux FPGA Zybo Z7-20, le système intègre une architecture hybride ChaCha20/RO-TRNG pour la génération de clés, validée par la suite de tests NIST SP 800-22. L'architecture s'appuie sur un pipeline en virgule fixe combinant une cascade CORDIC sans multiplicateur et une projection algébrique à fuite nulle. Validée à 125 MHz, l'implémentation génère un cryptogramme indiscernable d'un bruit blanc, caractérisé par un SNR de 87 à 109 dB et une corrélation quasi nulle avec le signal d'origine.

Mots-clés : Chiffrement basé sur les sous-espaces (SBE), FPGA Zybo Z7-20, ChaCha20, Générateur de véritables nombres aléatoires (TRNG), NIST, protocole I2S, arithmétique en virgule fixe, communication full-duplex.

Abstract

This work explores real-time audio encryption using Subspace-Based Encryption (SBE), which embeds the plaintext into a signal subspace and confines a non-linearly coupled masking noise to its orthogonal subspace, thereby enabling perfect recovery via a single linear projection. A full-duplex system is implemented on two Zybo Z7-20 FPGA boards, with a hybrid ChaCha20/RO-TRNG key generation validated against NIST SP 800-22 suite, and a fixed-point arithmetic pipeline using a multiplier-free CORDIC cascade and an algebraic zero-leakage projection. The system meets timing closure at 125 MHz and produces ciphertext indistinguishable from white noise, with SNR between 87 and 109 dB and near-zero plaintext correlation.

Keywords: Subspace-Based Encryption (SBE), Zybo Z7-20 FPGA, ChaCha20, True Random Number Generator (TRNG), NIST, I2S protocol, Fixed-point arithmetic, Full-duplex communications.

Dedication

To my beloved parents, to whom I owe everything. No words, no effort, no lifetime would ever be enough to repay what you have given me. To my mom, my first teacher, the one who shaped everything I know today and who will always remain my greatest role model. To my dad, for your constant presence and care, for never saying no, for always being there whenever I needed you.

To my grandparents, Allah yerhamkoum. You always believed in me, always encouraged me, and were always proud. I dedicate this small glimpse of success to you. To my uncle Mohamed, who felt more like a grandfather. You were there when I took my first steps, and now that I stand on my own feet, I know I owe a part of that to you.

To my brother, even if we were never the closest, you were always there when it mattered, for support, for help, or simply for those moments of craziness we shared.

To Ali, Yasmine, and Nada, my Samy pals, the ones who first showed me what friendship really means. Thank you for putting up with me all these years, I will always be grateful.

To Amine, my first friend at ENP, the first person I spoke to. Without you, I don't know how things would have turned out. To Massil, Amira, Malak, and Lylia, for those long prepa nights, the endless Meets sessions, the laughter, the stress, and the memories we built together.

To my ELN friends, Meskali, Rached, Meddas, Chicha, Ferroukhi, Hamid, and Bouchelaghem. You opened my eyes to new perspectives, pushed me beyond what I thought I could do, and brought me closer to competitions and challenges, one step closer to Eurobot. Special shoutout to PolyBot.

To Malek Youcef, for all the chaos, the random talks, the mood swings, and everything in between. Those moments meant more than they seemed. To Dhiaa, one of the newest yet closest friends. Through our late-night talks and long conversations, you gave me a new meaning of friendship I didn't know I was missing.

To Tarek Bouznad, who was the President of IEEE at the time, thank you for everything you've done for me. You helped me unlock my potential, guided me when I needed it the most, and were always present at the right moments. You played a huge role in shaping IEEE into what it became during that period, and you were not only a great leader but also a mentor, especially in competitions and IEEE-related activities. To IEEE, and especially to the HoDs I had the chance to work with, as well as the new generation who carry its spirit today—Sabrian, Friza, De9a, Bedka, and Samed. You are the new energy and heartbeat of the club. IEEE gave me so much, and I only hope I was able to give back even a small part of it.

To my teammate Rached, I am truly grateful to have worked with you. You supported my craziest ideas and my chaotic way of working with patience and kindness. You are one of the nicest people I have met, and this journey would not have been the same without you.

Rayan.

Dedication

To the source of my strength, my peace, and my courage, to the candles that illuminate my life, my parents. Mom and Dad, there are not enough words to thank you for everything you have done for me. Thank you for every sacrifice you made without ever asking for anything in return, for every prayer whispered for my success, and for every moment you stood beside me, even when I doubted myself. If this work has seen the light today, it is because your unwavering faith in me never allowed mine to fade

To my brothers, thank you for being my first friends, my supporters, and my constant source of strength. Thank you for every laugh, every argument, and every moment that reminded me that no matter where life takes us, I will never walk alone, because I will always have you by my side

To all my ENPC teachers, thank you for laying the foundations upon which I built this journey. You taught me discipline, resilience, and the confidence to aim higher than I ever thought possible

To my friends, whose names are too many to write and whom I would certainly forget if I tried to list them all, thank you for every conversation, every shared moment, and every memory that made this journey lighter and happier, and the friendships we built will remain one of the greatest gifts of that chapter

To the ELN family, thank you for making these years unforgettable. We shared stress, sleepless nights, impossible deadlines, victories, failures, and countless memories that I will carry with me forever. Thank you for every coffee break, every laugh in the middle of chaos, and every moment that made this journey feel a little less difficult

And finally, to my brother in this adventure, my teammate, Mohamed Rayan. From Eurobot nights to endless debugging sessions, thank you for being there through it all. We built memories that will stay with me for a lifetime. And we learned that sometimes, one click can turn a quiet evening into a very long journey

Rached.

Acknowledgments

This work marks the beautiful end of our academic journey, closing a defining chapter of our lives while opening the door to a new one. We are deeply grateful to everyone who walked this path with us and contributed to the success of this project.

Our ultimate thanks go to Allah for granting us the strength and tenacity to overcome every single obstacle in our way. We say Alhamdulillah for everything, for the highs, the lows, and the future that lies ahead.

First and foremost, we would like to express our deepest gratitude to Pr. Adel Belouchrani. When we first spoke last year, he placed his trust in us and believed in our potential before we even fully realized it ourselves. We deeply appreciate the opportunity he provided, and we poured our hearts into this work, hoping to honor his faith and bring this project to life in the best way possible.

To our families, we owe a debt of gratitude that words can never fully repay. Thank you for your unconditional love, your endless patience, and for working hard to provide for us and support us financially and emotionally throughout our entire studies. Your sacrifices made this dream a reality.

We also extend our sincere gratitude to our co-supervisors, Mr. Atef MERMOUL and Djawad RAMDANI, for their invaluable help and constant guidance. We are deeply thankful for their unwavering support and the time they dedicated to helping us.

We extend our heartfelt appreciation to Mr. Taghi, Head of the Electronics Department, for graciously accepting the role of President of the Jury. Beyond his administrative duties, he has been a true mentor and a father figure to us. His demanding standards, even in those moments when he refused us extra days or challenged us with heavy schedules, forged our resilience, taught us discipline, and planted in us a deep devotion to our work.

Our sincere thanks also go to Mr. Ouladj for honoring us by accepting to evaluate our work and serving as our examiner. We are grateful for his time and invaluable academic insight.

To all our electronic teachers: thank you for the continuous help, patient guidance, and knowledge you have so generously shared with us. Through the easy days and the tough ones, every lesson and every challenge has truly shaped who we are today, and we are better for it.

To our classmates, our friends, and those who have become like brothers to us: thank you for your constant presence and solidarity. The ELN department is like a huge, dysfunctional family, but everyone is always there for each other, and for that, we are truly thankful.

Last but not least, we thank Polytec' for the enriching environment and the many opportunities it provided. Special thanks goes to Mr. Zouaghi and Miss. Sabiha, who acted as our bridge to opportunities beyond the classroom. They always welcomed us with an open door, offering wise advice and a steady hand whenever we faced new challenges.

Rayan & Rached.

Contents

List of Tables

List of Figures

List of Acronyms

General Introduction	18
1 State of the Art	21
1.1 The Fundamental Security Principles	21
1.2 The Classical Era	21
1.2.1 Classical Cipher Methods	22
1.2.2 Limits of the Classical Approach	22
1.3 Modern Cryptography Techniques	23
1.3.1 Symmetric Key Cryptography	23
1.3.2 Asymmetric Key Cryptography	24
1.3.3 Practical Limitations and the Motivation for New Paradigms	25
1.4 Related Work	25
1.5 BSS-Based Encryption	26
1.5.1 BSS as a Cryptographic Technique	26
1.5.2 Joint plaintext Key Mixing	27
1.5.3 Pre-Shared Matrix Decryption	28
1.6 Subspace-Based Encryption	28
1.7 Conclusion	29
2 Subspace-Based Encryption	30
2.1 Orthogonal Subspace-Based Encryption	30
2.1.1 Global Scheme	30
2.1.2 Encryption and Decryption Process	31
2.2 The Non-linear Function and Its Role	33

2.2.1	Necessity of the Non-linear Coupling	33
2.2.2	Design Constraints on $g(\cdot)$	33
2.2.3	Security Against Attacks	34
2.3	Oblique Subspace-Based Encryption	35
2.3.1	Oblique Projection	35
2.3.2	Encryption and Decryption Process	35
2.4	Conclusion	36
3	Key Generation	37
3.1	Random Number Generator	37
3.2	Hardware Implementation of the LFSR Keystream Generator	38
3.2.1	Architecture Overview	38
3.2.2	Step-by-Step Hardware Implementation	38
3.2.3	Cryptographic Weakness: The Berlekamp-Massey Attack	39
3.3	Hardware Implementation of the ChaCha20	40
3.3.1	Step-by-Step Hardware Implementation	40
3.3.2	Known Cryptanalytic Attacks on ChaCha20	42
3.4	Comparative Analysis Between LFSR and ChaCha20	43
3.4.1	Cryptanalytic Comparison	43
3.4.2	Hardware Implementation Comparison	43
3.5	Hardware Implementation of the R0-TRNG	44
3.5.1	Architectural Overview	44
3.5.2	Step-by-Step Hardware Implementation	45
3.6	Hybrid Key Generation: ChaCha20/RO-TRNG	47
3.7	Randomness Validation: NIST SP 800-22 Test Suite	48
3.7.1	Overview of the NIST SP 800-22 Framework	49
3.7.2	NIST Analysis and Empirical Results	49
3.8	Conclusion	50
4	Communications Channel	51
4.1	Theoretical Foundations	51
4.1.1	Audio Sampling and Quantization	51
4.1.2	Transmission Bit Rate	52
4.2	Audio Interface and Clocking Scheme	52
4.2.1	Clock Hierarchy	52
4.3	AXI4-Stream Standard Interface	53
4.3.1	Protocol Overview	53

4.4	Proposed communications Scheme	54
4.4.1	Transmitter Architecture	54
4.4.2	Receiver Architecture	55
4.5	Network protocol and transmission mode selection	56
4.5.1	Network protocols	56
4.5.2	Transmission Modes	58
4.6	Architectural Constraints and Data Path Optimization	59
4.6.1	DMA Double Buffering Strategy	59
4.6.2	Ethernet Payload Sizing and IP Fragmentation	59
4.6.3	Mathematical Optimization of Hardware Parameters	60
5	FPGA Implementation	62
5.1	Hardware Platform	63
5.1.1	SoC Architecture Overview	63
5.1.2	Motivation for the Choice of the Zybo z7-20	63
5.2	Audio Codec Architecture and Control Interface	64
5.2.1	System Configuration and I^2C Control Path	64
5.2.2	Data Streaming, Sampling, and Digital Filtering	65
5.2.3	Analog Routing: Mic-In and Line-In Configurations	66
5.2.4	Initialization Sequence and Register Mapping	66
5.3	Communications Channel Implementation	67
5.3.1	Gigabit Ethernet Controller	67
5.3.2	Implementation of the DMA-Based Audio Data Transfer	69
5.4	Fixed-Point Representation Overview	71
5.5	Encryption Architecture	73
5.5.1	AXI4-Stream Interface and Audio Framing	73
5.5.2	Top-Level Encryption System	73
5.5.3	The Non-linear Function: CORDIC-Based Realization	75
5.5.4	Orthogonal Projection and Coefficient Generation	78
5.5.5	The Encryption Arithmetic Pipeline	80
5.6	Decryption Architecture	83
5.6.1	Mathematical Basis	84
5.6.2	Top-Level Decryption System	84
5.6.3	Decryption Arithmetic Stages	84
5.7	Full-Duplex System Architecture	85

6	Results and Analysis	89
6.1	Hardware Implementation Results	89
6.1.1	Resource Utilization	90
6.1.2	Timing Analysis	90
6.1.3	Power Consumption	91
6.1.4	Pipeline Latency	92
6.2	Security Analysis	93
6.2.1	Time and Frequency Domain Analysis	93
6.2.2	Signal-to-Noise Ratio	95
6.2.3	Statistical Analysis	95
6.3	Subspace and Array Processing Attacks	97
6.4	Comparison with Related Work	99
	Conclusion	101
	Bibliography	103
	Appendix	108

List of Tables

1.1	Comparison of symmetric key algorithms [1, 2].	24
1.2	Comparison of asymmetric key algorithms [1, 2].	25
1.3	Summary of related work in encryption research.	26
1.4	Security property comparison of reviewed encryption schemes.	28
2.1	Security of the proposed SBE scheme against cryptanalysis attacks.	34
2.2	Comparison between orthogonal and oblique SBE schemes.	36
3.1	Summary of known cryptanalytic attacks on reduced-round ChaCha20.	42
3.2	Cryptanalytic comparison between LFSR and ChaCha20.	43
3.3	Hardware implementation comparison on Xilinx Zynq-7000 (Zybo Z7-20).	44
3.4	NIST Results: ChaCha20 (Fixed/TRNG Nonce) and RO-TRNG.	50
5.1	Zynq-7020 components and their role in the secure audio system.	64
5.2	SSM2603 I^2C register configuration and initialization sequence.	67
5.3	AXI4-Stream TDATA framing convention.	73
5.4	First ten CORDIC micro-rotation angles $\alpha_i = \arctan(2^{-i})$	76
6.1	Implementation resource utilization on the Zybo Z7-20.	90
6.2	Implementation timing quality [3].	91
6.3	Timing analysis results.	91
6.4	Implementation resource utilization on the Zybo Z7-20.	92
6.5	Project latency assessment.	93
6.6	SNR and correlation analysis for mono audio configuration.	96
6.7	SNR and correlation analysis for stereo audio configuration.	96
6.8	NIST randomness test results for the encrypted samples.	97
6.9	Comparison with existing FPGA-Based Audio encryption architectures.	99
6.10	Correlation coefficient comparison with [4].	99

List of Figures

1.1	Modern cryptography techniques [1].	23
2.1	Global SBE Scheme [5].	31
3.1	Hardware architecture of the LFSR scheme.	38
3.2	ChaCha 20 architecture overview [6].	40
3.3	ChaCha20 quarter-round algorithm [6].	41
3.4	True random number generator scheme.	45
3.5	TRNG output bit distribution.	46
3.6	TRNG with post-processing output bit distribution.	46
3.7	Design flow for the updated RO hardware architecture.	47
3.8	TRNG with updated RO hardware architecture output bits distribution.	47
3.9	RTL IP core for the SBE key generation module within Vivado.	48
3.10	Hybrid key generation architecture: ChaCha20 for A and RO-TRNG for nonce, B and K	48
4.1	I ² S protocol.	52
4.2	AXI4-Stream handshake	54
4.3	Proposed transmitter architecture.	55
4.4	Proposed receiver architecture.	56
4.5	TCP/IP protocol layers scheme.	57
4.6	Simplex data transmission flow [7].	58
4.7	Half-duplex data transmission flow [7].	58
4.8	Full-duplex data transmission flow [7].	58
5.1	Zynq-7000 SoC functional overview.	63
5.2	IP block of the system architecture: the PS's I^2C control and the PL's I^2S data streaming.	65

5.3	SSM2603 internal analog routing architecture [8].	66
5.4	Block diagram of the Gigabit Ethernet controller [9].	68
5.5	AXI DMA configuration parameters used in the system.	69
5.6	Data structure payload format.	70
5.7	Organization of a 160-bit encrypted payload serialized into five 32-bit DMA transfer words.	71
5.8	Signed fixed-point representation in $Q_{m.n}$ format [10].	72
5.9	Top-level architecture of the SBE encryption system.	74
5.10	CORDIC scaling factor and magnitude convergence.	76
5.11	Two CORDIC cascade chain.	77
5.12	FSM of the orthogonal projection coefficient calculator.	79
5.13	Data transfer between registers within timing constraints	80
5.14	Encryption architecture: Phase 1 schematics.	81
5.15	Encryption architecture: Phase 2 schematics.	82
5.16	Encryption architecture: Phase 3 schematics.	82
5.17	Encryption architecture: Phase 4 schematics.	83
5.18	Top-level of the decryption architecture.	84
5.19	Data flow diagram of the decryption system.	85
5.20	Physical implementation of the dual-board full-duplex secure audio communi- cations system.	86
5.21	Complete IP block design of the full-duplex cryptographic system.	87
6.1	Power analysis for the evaluated system configurations.	92
6.2	Time, frequency domains, and spectrogram analysis comparing all audio streams.	94
6.3	Plot of original and blind-attack-recovered signals.	98

List of Acronyms

- **3DES:** Triple Data Encryption Standard
- **ADC:** Analog-to-Digital Converter
- **AES:** Advanced Encryption Standard
- **AHB:** Advanced High-performance Bus
- **ANN:** Artificial Neural Network
- **APB:** Advanced Peripheral Bus
- **ARX:** Add-Rotate-XOR
- **ASCII:** American Standard Code for Information Interchange
- **AXI:** Advanced eXtensible Interface
- **BCLK:** Bit Clock
- **BOSR:** Base Over-Sampling Rate
- **BRAM:** Block Random Access Memory
- **BSP:** Board Support Package
- **BSS:** Blind Source Separation
- **BUFG:** Buffer Global Clock
- **CLB:** Configurable Logic Block
- **CMOS:** Complementary Metal-Oxide-Semiconductor
- **CORDIC:** Coordinate Rotation Digital Computer
- **CPU:** Central Processing Unit
- **CRC:** Cyclic Redundancy Check
- **DAC:** Digital-to-Analog Converter
- **DDR:** Double Data Rate

- **DES:** Data Encryption Standard
- **DFF:** D-Flip-Flop
- **DFT:** Discrete Fourier Transform
- **DMA:** Direct Memory Access
- **DoA:** Direction of Arrival
- **DSP:** Digital Signal Processor
- **ECC:** Elliptic Curve Cryptography
- **EMIO:** Extended Multiplexed Input/Output
- **FF:** Flip-Flop
- **FIFO:** First-In, First-Out
- **FPGA:** Field Programmable Gate Array
- **FSM:** Finite State Machine
- **GEM:** Gigabit Ethernet MAC
- **GMII:** Gigabit Media Independent Interface
- **HPF:** High-Pass Filter
- **I²C:** Inter-Integrated Circuit
- **I²S:** Inter-IC Sound
- **ICA:** Independent Component Analysis
- **IDEA:** International Data Encryption Algorithm
- **ILA:** Integrated Logic Analyzer
- **IO:** Input/Output
- **IP:** Intellectual Property / Internet Protocol
- **ISR:** Interrupt Service Routine
- **JTAG:** Joint Test Action Group
- **LFSR:** Linear Feedback Shift Register
- **LRCLK:** Left Right Clock
- **LSB:** Least Significant Bit
- **LUT:** Look-Up Table
- **LUTRAM:** Look-Up Table Random Access Memory

- **lwIP:** Lightweight IP
- **MAC:** Media Access Control
- **MCLK:** Master Clock
- **MDIO:** Management Data Input/Output
- **MIO:** Multiplexed Input/Output
- **MM2S:** Memory-to-Stream
- **MMCM:** Mixed Mode Clock Manager
- **MSB:** Most Significant Bit
- **MSS:** Maximum Segment Size
- **MTU:** Maximum Transmission Unit
- **MUSIC:** Multiple Signal Classification
- **MVDR:** Minimum Variance Distortionless Response
- **NIST:** National Institute of Standards and Technology
- **OSI:** Open Systems Interconnection
- **P2P:** Peer-to-Peer
- **PAM-5:** Pulse Amplitude Modulation 5
- **PCA:** Principal Component Analysis
- **PCM:** Pulse-Code Modulation
- **PHY:** Physical Layer
- **PL:** Programmable Logic
- **PNB:** Probabilistic Neutral Bit
- **PRNG:** Pseudo-Random Number Generator
- **PS:** Processing System
- **QR:** Quarter-Round
- **RC4:** Rivest Cipher 4
- **RFC:** Request for Comments
- **RGMII:** Reduced Gigabit Media Independent Interface
- **RNG:** Random Number Generator
- **RO-TRNG:** Ring Oscillator True Random Number Generator

- **RSA:** Rivest-Shamir-Adleman
- **RTL:** Register-Transfer Level
- **S2MM:** Stream-to-Memory
- **SBE:** Subspace-Based Encryption
- **SNR:** Signal-to-Noise Ratio
- **SoC:** System-on-Chip
- **SOBI:** Second-Order Blind Identification
- **SQNR:** Signal-to-Quantization-Noise Ratio
- **TCP:** Transmission Control Protocol
- **TNS:** Total Negative Slack
- **UDP:** User Datagram Protocol
- **UTP:** Unshielded Twisted Pair
- **VHDL:** VHSIC Hardware Description Language
- **VLAN:** Virtual Local Area Network
- **WNS:** Worst Negative Slack
- **XOR:** Exclusive OR

General Introduction

The security of digital communications has traditionally relied on algebraic transformations applied uniformly to any data stream, regardless of its nature. While such methods are well established for text and generic data, real-time audio introduces stricter constraints: encryption and decryption must occur within a fixed sample-rate budget, and any computational overhead directly threatens the continuity of the conversation. Securing live audio, therefore, requires not only cryptographic robustness but also an architecture compatible with real-time embedded constraints.

Beyond the security requirements, the deployment platform plays a critical role in determining the functionality of a communications system. Software-based solutions executed on general-purpose processors may introduce variable execution times, which can affect deterministic operation. In contrast, dedicated hardware platforms offer opportunities for parallel processing, predictable latency, and efficient handling of continuous data streams. These characteristics make them well-suited for real-time signal processing applications.

This work explores a hardware realization of Subspace-Based Encryption (SBE), a signal-processing approach to cryptography, implemented on a pair of FPGA boards to achieve a complete, real-time, full-duplex secure audio communications system.

Motivations

Real-time audio communications is now embedded in countless devices, from intercoms to IoT voice assistants, yet most security solutions treat the audio stream as a generic sequence of bits. Algorithms such as AES or ChaCha20 apply confusion and diffusion at the bit level, entirely independent of the underlying signal structure.

The SBE algorithm offers an alternative: it exploits the geometric structure of the signal itself, embedding the plaintext into a signal subspace and confining masking energy to its orthogonal complement. The prospect of realizing this signal-processing paradigm as a working hardware system capable of securing a live conversation between two boards motivated this work.

Problem Context

Securing real-time audio raises two distinct issues: the cryptographic strength of the algorithm, and the implementability of its design under strict hardware and timing constraints.

Limitations of Existing Cryptographic Paradigms

Classical ciphers are broken by frequency analysis, while modern symmetric and asymmetric algorithms (AES, RSA, ECC) remain tied to key-distribution overhead or quantum vulnerability. Blind Source Separation (BSS)-based ciphers offered a geometric alternative, but their linear mixing structure proved fatally weak: differential and known-plaintext attacks cancel the masking term and expose the key. SBE resolves this by confining masking noise to the orthogonal of the signal subspace and coupling it non-linearly to the plaintext, while preserving perfect recovery through a single projection.

From Theory to Hardware

The Subspace-Based Encryption (SBE) was first introduced by Dr. A.Mermoul and Pr. A.Belouchrani in [5], where the authors proposed projecting the speech signal into signal and noise subspaces to achieve encryption. The method was generalized and validated in [11], which analyzed its cryptographic properties and resistance to attacks. An iterative extension to improve the encryption process was later proposed in [12] and presented at the 7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA 2011) [13]. A more comprehensive treatment of the theoretical foundations, robustness analysis, and security aspects of SBE was presented in the doctoral dissertation of Dr. A.Mermoul [14] and later mentioned in the book chapter on subspace-based encryption [15].

Translating SBE into a real-time fixed-point pipeline raises new challenges: arithmetic must preserve exact orthogonality, key matrices must be regenerated identically on two boards without transmitting secret material, and the complete encryption/decryption path must satisfy the timing constraints imposed by the audio sampling rate.

Objectives of This Work and Contributions

The overall objective of this work is to design, implement, and validate a complete real-time, full-duplex secure audio communications system based on SBE, deployed on two Zybo Z7-20 FPGA platforms. The specific technical objectives are:

- To design a hardware key-generation pipeline capable of securely producing all key-generation matrices required by the SBE scheme.
- To implement the SBE encryption/decryption pipeline with minimized information leakage and latency.
- To establish an end-to-end communications framework interconnecting the audio codec, the cryptographic core, and the Gigabit Ethernet interface, including a synchronization mechanism.
- To integrate the two communications directions into a full-duplex transceiver and to evaluate resource utilization, timing performance, power consumption, and latency.
- To assess the security of the implemented system via time- and frequency-domain analyses, signal-to-noise ratio (SNR) evaluation, correlation analysis, and statistical randomness testing, and to benchmark its performance against existing FPGA-based audio encryption solutions.

The principal contribution is a hardware-validated demonstration, previously evaluated only as proof of concept, that can be realized as a robust cryptographic system on FPGA hardware.

Work Structure

The work is organized into six chapters, progressing from theoretical foundations to hardware implementation and experimental validation:

- **Chapter 1 (State of the Art)** surveys the evolution of cryptography, from classical ciphers through modern standards and BSS, leading to SBE.
- **Chapter 2 (Subspace-Based Encryption)** develops the SBE mathematical framework, its orthogonal and oblique formulations, and the cryptographic role of the non-linear function.
- **Chapter 3 (Key Generation)** details the key generation hardware: the LFSR prototype, its replacement by ChaCha20 and RO-TRNG, and NIST validation.
- **Chapter 4 (Communications Channel)** establishes the communications channel architecture linking the audio interface, AXI4-Stream core, and network.
- **Chapter 5 (FPGA Implementation)** presents the complete FPGA implementation, including fixed-point arithmetic, the CORDIC non-linearity, the projection reformulation, synchronization, and full-duplex integration.
- **Chapter 6 (Results and Analysis)** evaluates the implemented system through resource, timing, power, and security analyses, concluding with a comparison to the literature.

State of the Art

This chapter presents a review of the evolution of cryptographic methods. It begins with an overview of the first cipher families and their associated mathematical limitations, moves through the foundations and standard algorithms of modern cryptography, and arrives at the signal-processing paradigm of subspace-based encryption, whose theoretical properties and open challenges are the central subject of the chapters that follow.

1.1 The Fundamental Security Principles

Any cryptographic algorithm must satisfy a set of security objectives to be considered reliable. These are [1, 16, 17]:

- **Confidentiality:** Data must only be accessible to authorized parties, blocking any unauthorized access.
- **Authentication:** The identity of the communicating parties must be verified, ensuring that information comes from a trusted source.
- **Integrity:** Information must remain accurate and remain unchanged throughout its life-cycle. Any unauthorized modification must be detectable.
- **Non-repudiation:** Neither the sender nor the receiver can deny their participation in a transmission after it has taken place.
- **Access control:** Resources and services must remain accessible to authorized users at all times.

These objectives define what it means for a cryptographic system to be secure.

1.2 The Classical Era

Before the formalization of information theory and computational complexity, cryptographic schemes relied on two elementary mechanisms: substitution and transposition. Their security relied entirely on the transformation that remains unknown to the adversary, rather than on any mathematical complexity.

1.2.1 Classical Cipher Methods

Substitution ciphers

Substitution ciphers simply replace each plaintext symbol with a different symbol drawn from a predefined correspondence. It can be divided into the following:

- **Caesar cipher:** Each letter p is shifted by a fixed offset k modulo the alphabet size [16]:

$$c = (p + k) \bmod 26, \quad p = (c - k) \bmod 26, \quad (1.1)$$

where the integer $k \in \{1, \dots, 25\}$ constitutes the entire key space. As noted in [17, 18], this scheme is the easiest cryptographic algorithm to break; a search over 25 values is trivial, and the alphabet shift structure is immediately recognizable.

- **Monoalphabetic (Simple Substitution) cipher:** It generalizes equation (1.1) by replacing the fixed shift with an arbitrary permutation of the alphabet [17], that leads to a key space of $26!$ elements. Despite this large number, the scheme is broken by frequency analysis mentioned in [19]. Because each plaintext letter always maps to the same ciphertext, the statistical distribution of natural language is fully preserved.
- **Vigenère Cipher:** It is a generalization of the Caesar cipher in which the keyword takes on various values, so that the same key is not used to cipher all characters [18]. Each plaintext letter is encoded by the intersection of its row and the corresponding key letter column in a 26×26 substitution table. Despite its increased complexity, the periodicity of the keyword allows the key length to be estimated and each sub-stream to be attacked independently [18].

Transposition Ciphers

Unlike substitution-based methods, transposition ciphers encrypt messages by rearranging plaintext letter positions rather than replacing them, producing a ciphertext that is a permutation of the original plaintext. Common variants include the Rail Fence Cipher, Scytale, Route Cipher, Row Transposition Cipher, and Columnar Transposition [16].

In the Row Transposition Cipher, the plaintext is written row by row into a rectangular grid and read column by column according to a permuted key. For example, encrypting “attack postponed until two am” gives TTNAAPTMTSUOAODWCOIXKNLYPETZ [16].

Since letters are never altered, the frequency distribution remains identical to the plaintext, leaving the cipher susceptible to attacks, with high memory consumption being an additional drawback [16].

1.2.2 Limits of the Classical Approach

Despite their variety, classical ciphers share fundamental weaknesses that render them unsuitable for modern security needs.

- **Key distribution:** Both parts must share the same secret key. If there is an interception, all encrypted communication is immediately exposed [16].
- **Weakness against letter frequency attacks:** Since classical ciphers preserve letter frequencies, frequency analysis is sufficient to recover the plaintext and the key without

any prior knowledge [19, 17].

- **Susceptibility to brute force attack:** The small key spaces of classical schemes (for example, only 25 possible keys for Caesar) make all possible keys a straightforward task [16, 17].

These limitations motivated the shift toward mathematics-based cryptography.

1.3 Modern Cryptography Techniques

Modern cryptographic techniques fall into two main families based on how keys are managed. In Figure 1.1, we have a hierarchical breakdown of these algorithms divided into symmetric and asymmetric key families, which include the following specific techniques:

- **RC4:** Rivest Cipher 4.
- **DES:** Data Encryption Standard.
- **3DES:** Triple Data Encryption Standard.
- **AES:** Advanced Encryption Standard.
- **IDEA:** International Data Encryption Algorithm.
- **Blowfish:** Named after its creator Blowfish.
- **RSA:** Rivest-Shamir-Adleman.
- **El-Gamal:** Named after its creator Taher Elgamal.
- **ECC:** Elliptic Curve Cryptography.

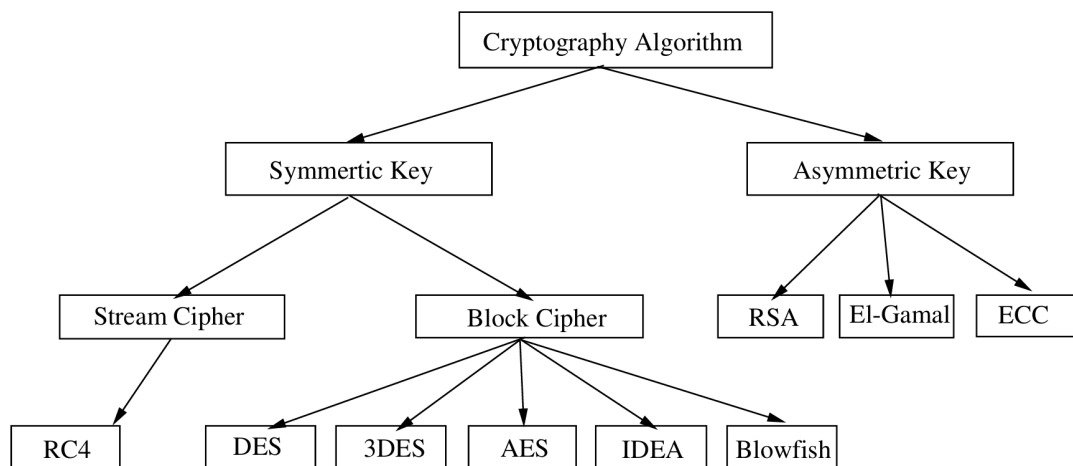


Figure 1.1: Modern cryptography techniques [1].

1.3.1 Symmetric Key Cryptography

In this type of cryptography, both sender and receiver use the same secret key for encryption and decryption. They are divided into stream ciphers, which encrypt data one byte at a time,

such as RC4, and block ciphers, which process fixed-size blocks [1]. Table 1.1 compares the most widely used algorithm designs for symmetric cryptography.

Table 1.1: Comparison of symmetric key algorithms [1, 2].

Algorithm	Structure	Key Size (bits)	Keys	Block (bits)	Rounds	Speed	Security
RC4	Stream	40 to 256	1	–	1	Fast	Weak
DES	Block	56	1	64	16	Slow	Weak
3DES	Block	112 / 168	3	64	48	Very slow	Moderate
AES	Block	128 / 192 / 256	1	128	10 / 12 / 14	Fast	High
Blowfish	Block	32 to 448	Variable	64	16	Fast	Moderate
IDEA	Block	128	1	64	8	Slow	Moderate

In addition to the algorithms summarized in Table 1.1, the S-Box-based encryption method is another widely used symmetric technique in image security, where it provides confusion and non-linearity by replacing plaintext elements using a predefined substitution table [20]. In shared key encryption, padding¹ is essential to ensure that input data fits the fixed block size of the cipher. This padding must be recognizable while revealing no information that could weaken the security of the message [21].

Moreover, the One-Time Pad (OTP) is a symmetric encryption technique that uses a truly random key with the same length as the plaintext message. Each key is used only once and then discarded. When implemented correctly, OTP provides perfect secrecy and is theoretically unbreakable. However, its practical use is limited due to the challenges associated with generating, distributing, and securely managing long random keys [22].

While symmetric algorithms are generally faster. The efficiency and speed of its encryption make it suitable for encrypting large volumes of data and for real-time processing requirements [23]. However, there is one major challenge, which is key management, especially the secure distribution and storage of secret keys, as both sender and receiver must exchange the same secret key before any communication begins [1].

1.3.2 Asymmetric Key Cryptography

Asymmetric cryptography, also known as public key cryptography, uses a pair of keys: a public key shared openly across the network and a private key kept secret by its owner. The sender encrypts the message using the recipient's public key, and only the recipient can decrypt it using their private key. Since the private key is never shared, this approach is more secure than symmetric schemes at a higher computational cost [1, 20]. The most widely used asymmetric algorithms are RSA, El-Gamal, and ECC algorithms, whose key characteristics are summarized in Table 1.2.

Asymmetric algorithms have higher computational complexity than symmetric ones, which makes them better for key exchange and digital signatures rather than encrypting large amounts of data [24]. They are commonly used for secure key distribution, authentication, digital signatures, and setting up secure communications channels [20].

¹Padding is an extra bit added to a data structure to achieve required word alignment.

Table 1.2: Comparison of asymmetric key algorithms [1, 2].

Algorithm	Technique	Key Size (bits)	Rounds	Speed	Security
RSA	Prime Number	1024–4096	1	Fast	High
El-Gamal	One Random Number	1024	1	Slow	Moderate
ECC	Elliptic Curve	Variable	1	Fast	High

1.3.3 Practical Limitations and the Motivation for New Paradigms

Both symmetric algorithms RC4 and 3DES are considered weak and vulnerable to quantum attacks [2].

On the asymmetric side, RSA and El-Gamal are computationally expensive due to large prime numbers and modular exponentiation, while also being vulnerable to quantum attacks [1, 2]. Although ECC is more efficient and better suited for constrained devices, it offers only partial quantum resistance [2].

These limitations motivate the need for cryptographic approaches that are both lightweight and resilient against modern threats.

1.4 Related Work

This section reviews existing encryption research organized by design approach, covering classical ciphers, hardware implementations, audio encryption, hybrid key management, and asymmetric constructions.

Classical Cipher Enhancements

The work in [25] combines two classical encryption layers, a modified Vigenère cipher and a Polybius square, to better resist frequency-based attacks and strengthen security.

Symmetric Encryption and Hardware Implementations

In [26], an AES-128 encryption pipeline is built on a hardware chip, achieving a speed of 1.33 Gbps while using minimal chip area.

In contrast, [27] hides the encryption key inside the ciphertext itself using simple shift and XOR operations, reaching 48.95 KB/ms.

Audio Encryption

The work in [28] encrypts audio files through three successive operations: position shuffling, key-based mixing, and byte substitution, resulting in a very large key space and near-zero predictability between samples. The main limitation is that the process is too slow for real-time use.

The work in [29] develops a full-duplex real-time audio communications system based on chaotic encryption and implements it on a Nexys A7 FPGA.

In [4], an ANN-based hyperchaotic pseudo-random number generator is implemented on an Artix-7 FPGA and employed for audio encryption through a lightweight XOR operation.

Hybrid Encryption and Key Management

Hybrid AES+RSA schemes have been validated at both the protocol and application levels. At the hardware protocol level, the study mentioned in [30] proposes combining AES for encryption with RSA for key exchange on an FPGA platform, achieving both confidentiality and authentication while solving the key distribution problem of symmetric-only schemes.

Asymmetric Encryption via Mathematical Constructions

The scheme in [31] builds an asymmetric cipher using integer equations as the hard mathematical problem, keeping the key count fixed regardless of message length. However, no formal proof has been provided that the underlying problem is truly hard to break.

Table 1.3 summarizes the surveyed works across all approaches:

Table 1.3: Summary of related work in encryption research.

Ref.	Year	Domain	Configuration	Key Contribution	Speed
[25]	2022	Text	Vigènere + Polybius cascade	Dual-layer confusion/diffusion	—
[26]	2013	General	AES-128 hardware pipeline	1.33 Gbps at minimal chip area	1.33 Gbps
[27]	2023	Text	XOR + key rotation	Key embedded in ciphertext	48.95 KB/ms
[29]	2024	Audio	Chaos-based PRNG + XOR	Full-Duplex communications	254.38 MHz
[4]	2025	Audio	ANN-hyperchaotic PRNG + XOR	Lightweight FPGA encryption	105 MHz
[28]	2017	Audio	Three-stage chaotic pipeline	2^{319} key space	Slow
[30]	2023	General	Hybrid AES + RSA on FPGA	Confidentiality + Authentication	Fast
[31]	2024	Text	Integer equation cipher	Fixed key count, scalable	—

1.5 BSS-Based Encryption

An alternative to classical cryptography relies on the computational difficulty of underdetermined Blind Source Separation (BSS) as a cryptographic primitive [32, 33].

1.5.1 BSS as a Cryptographic Technique

In the standard BSS problem, N sensors record M statistically independent source signals through an unknown mixing process [32, 11]. The observation model is described as the following:

$$x(t) = A s(t) \quad (1.2)$$

where $x(t) \in \mathbb{R}^N$ is the vector of observations, $s(t) \in \mathbb{R}^{M \times 1}$ the vector of unknown sources, and $A \in \mathbb{R}^{N \times M}$ the unknown mixing matrix. The goal of BSS is to recover $s(t)$ from $x(t)$ alone, without knowledge of A .

The standard approach for recovery is Independent Component Analysis (ICA), which seeks

a demixing matrix $W \in \mathbb{R}^{M \times N}$ such that the estimated sources [33]:

$$u(t) = W x(t) = W A s(t) \quad (1.3)$$

have statistically independent components. Under the ICA model, perfect separation yields $WA = PD$, where P is a permutation matrix and D a diagonal scaling matrix [33]. Recovery is possible only in the determined ($M = N$) or overdetermined ($M < N$) cases [32, 11].

When $M > N$, the equation (1.2) is underdetermined. There are more unknowns than equations, and the mixing matrix A cannot be identified from $x(t)$ alone [33, 11]. Source recovery becomes impossible, which motivates the use of the underdetermined BSS as a cryptographic scheme [32, 33].

1.5.2 Joint plaintext Key Mixing

In [32], P plaintext signals $s(t)$ are jointly mixed with P pseudo-random key signals $k(t)$ through a $P \times 2P$ underdetermined mixing matrix A_e , constructed as the concatenation of two sub-matrices:

$$x(t) = A_e \begin{bmatrix} s(t) \\ k(t) \end{bmatrix}, \quad A_e = \left[\underbrace{BC}_{A_I} \mid \underbrace{B}_{A_K} \right] \quad (1.4)$$

which expands as:

$$x(t) = \underbrace{A_I s(t)}_{\text{plaintext term}} + \underbrace{A_K k(t)}_{\text{key term}} \quad (1.5)$$

The key term dominates the ciphertext by an order of magnitude, effectively hiding the plaintext in the observation $x(t)$ [32].

At the receiver, the P received channels are augmented with the locally regenerated key signals (using a shared seed I_0) to form a $2P \times 2P$ well-determined system, from which ICA recovers the plaintext [32].

Despite its claimed analogy, the additive linear structure of equation (1.5) is an irreversible flaw [34]. In [34, 35], four attacks were demonstrated against it:

- **Differential attack:** Subtracting two ciphertexts encrypted under the same key causes the key term to cancel exactly, exposing the plaintext difference as a solvable BSS problem.
- **Known plaintext attack:** A single known plaintext–ciphertext pair is sufficient to recover the mixing coefficients and reconstruct the full key stream.
- **Chosen plaintext attack:** Sending a silent (all-zero) signal forces the ciphertext to equal the key term alone, directly revealing the secret key.
- **Low key sensitivity:** An incorrect key still yields a near-correct decryption. The effective key space can be reduced by attacking each row of the mixing matrix independently.

1.5.3 Pre-Shared Matrix Decryption

In [33], the ICA step is eliminated by transmitting the mixing matrix A to the receiver over a separate secure channel. The receiver applies the inverse A^{-1} directly. The encryption model is identical to equation (1.5), and experiments on 16 kHz speech ($P = 4$) report a perfect reconstruction exceeding 280 dB. The improvement is purely operational: since the additive structure of equation (1.5) is unchanged.

1.6 Subspace-Based Encryption

In [5], the linearity vulnerability is resolved through two complementary mechanisms: the key perturbation is confined to the orthogonal complement of the plaintext subspace, so vanishes algebraically upon decryption. Also, it is coupled non-linearly to the plaintext that prevents any differential cancellation.

Given secret key matrix $A(t) \in \mathbb{R}^{(M+1) \times M}$, the ciphertext is [5]:

$$x(t) = A(t)p(t) + \beta P_A^\perp(t) B(t) [k(t) \odot \mathbf{g}(p(t))] \quad (1.6)$$

where $P_A^\perp(t)$ is the orthogonal projector into the complement of signal space, $B(t) \in \mathbb{R}^{(M+1) \times M}$ is a random matrix, $k(t)$ is a key vector, β is a large scalar parameter, $\odot$ denotes the Hadamard product, and $g(v) = \frac{v}{\sqrt{1+v^2}}$ is a non-linear scalar function satisfying $g(0) = 0$. Its vector extension is defined as:

$$\mathbf{g}(p(t)) = [g(p_1(t)), g(p_2(t)), \dots, g(p_M(t))]^T$$

The non-linear coupling $\mathbf{g}(p(t))$ in equation (1.6) directly addresses the four vulnerabilities identified in [34, 35]: differencing two ciphertexts no longer cancels the key contribution since g entangles both plaintexts in the perturbation term; the mixing coefficients cannot be recovered from a plaintext–ciphertext pair due to the non-linear dependency; encrypting a zero plaintext yields a zero ciphertext by definition of g , providing no key information; and any key mismatch is amplified by β , producing a strong avalanche effect [5].

Experiments on 22.05 kHz, 16-bit speech with $M = 4$ and $\beta = 10^6$ reported an encrypted SNR of approximately -200 dB, a decrypted SNR exceeding 340 dB [5].

The scheme in [5] is the only construction in the BSS lineage to overcome the structural linearity flaw. Table 1.4 compares all reviewed schemes across the main security criteria.

Table 1.4: Security property comparison of reviewed encryption schemes.

Article Work	[32]	[33]	[5]
Encryption structure	Linear	Linear	Non-linear
Differential attack	Vulnerable	Vulnerable	Resistant
Known-plaintext attack	Vulnerable	Vulnerable	Resistant
Key sensitivity	Low	Moderate	High
Null-Information Property	No	No	Yes

The work in the next chapters builds directly on equation (1.6), extending the scheme to audio encryption with a comprehensive analysis.

1.7 Conclusion

This chapter reviewed the evolution of cryptographic methods, with a specific focus on audio encryption applications and their hardware implementations. By tracing the transition from classical and standard modern ciphers to signal-processing paradigms. Identifying these structural limitations provides the necessary theoretical foundation and context to direct the implementation and performance analysis presented in the following chapters.

Subspace-Based Encryption

Traditional cryptographic paradigms process data as abstract bit streams, relying on algebraic operations to achieve confusion and diffusion while remaining oblivious to the geometric properties of the physical signals. Subspace-Based Encryption (SBE) introduces a paradigm shift by leveraging the vector space structure of continuous signals. By embedding plaintext into a signal subspace and confining high-energy masking noise to a null space, SBE ensures perfect data recovery at the authorized receiver through a simple linear projection operation. This framework is realized through a foundational orthogonal formulation and a generalized oblique variant that independently decouples the subspaces for enhanced security. The remainder of the chapter is structured as follows:

- Section 2.1 introduces the foundational Orthogonal SBE architecture.
- Section 2.2 examines the critical cryptographic role of the non-linear function within the encryption process.
- Section 2.3 generalizes the geometric framework by presenting the Oblique SBE variant.

2.1 Orthogonal Subspace-Based Encryption

As the foundational architecture, orthogonal SBE uses algebraic projections to vectorize continuous signals and hide data within noise to achieve perfect security.

2.1.1 Global Scheme

The SBE scheme is motivated by the need to secure speech signals and exploit their geometric structure. It allows exact recovery of the plaintext at the receiver through a simple projection operation, without requiring knowledge of the noise parameters.

To enable this geometric framework, the plaintext audio signal is first divided into L non-overlapping frames of M consecutive samples each:

$$p(t) = [p_1(t), \dots, p_M(t)]^T \in \mathbb{R}^{M \times 1}, \quad t = 1, \dots, L \quad (2.1)$$

A larger M value increases the subspace dimension and hardens cryptanalysis, at the cost of heavier matrix operations per frame. Figure 2.1 shows the global block diagram of the SBE

system.

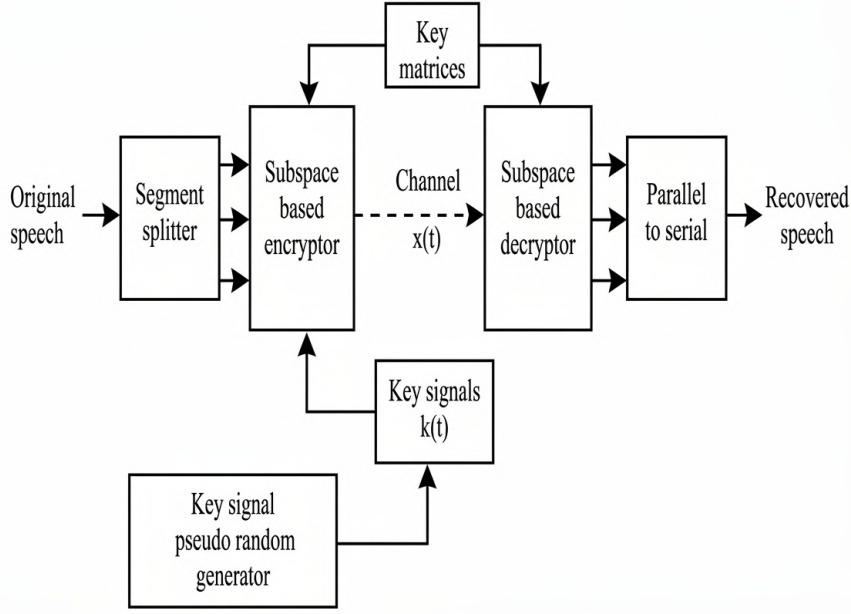


Figure 2.1: Global SBE Scheme [5].

The overall operation unfolds as follows:

1. **Segmentation:** The original signal is split into L frames of M samples each, yielding the plaintext vectors $p(t)$ according to equation (2.3).
2. **Encryption:** Each plaintext vector $p(t)$ is fed into the subspace-based encryptor along with $A(t)$, $B(t)$ and $k(t)$. The encryptor produces a ciphertext vector $x(t)$ of dimension $M + 1$, which is transmitted through the channel.
3. **Key generation:** At the transmitter, the secret seed drives three random generators that produce, for each segment t , the key matrix $A(t)$, the mixing matrix $B(t)$, and the key signal vector $k(t)$.
4. **Shared secret seed:** Both transmitter and receiver hold a common secret seed. It is the only parameter that must be exchanged securely before communication begins.
5. **Key regeneration at receiver:** Since both sides share the same secret seed, the receiver independently regenerates the same key matrices $A(t)$ using an identical pseudo-random generator. No additional transmission of $A(t)$ is required.
6. **Decryption:** The receiver applies the subspace-based decryptor, using only $A(t)$, to recover each segment $\hat{p}(t)$. The recovered segments are then assembled in parallel-to-serial order to reconstruct the original signal.

2.1.2 Encryption and Decryption Process

The encryption scheme splits the signal space $\mathbb{R}^{M+1}$ ¹ into two orthogonal subspaces:

1. **Signal subspace $\mathcal{S}$** where the information (plaintext) resides.

¹denotes the $(M + 1)$ dimensional vector space over the real numbers.

2. **Null subspace** $\mathcal{S}^\perp$ where the masking noise resides.

Encryption

Let $p(t)$ be the plaintext vector of dimension $M \times 1$. The ciphertext $x(t)$ of dimension $(M+1) \times 1$ is constructed as follows:

$$x(t) = \underbrace{A(t) p(t)}_{\text{Information Term}} + \underbrace{\beta P_A^\perp(t) B(t) [k(t) \odot \mathbf{g}(p(t))]}_{\text{Masking Noise Term}} \quad (2.2)$$

The components of the SBE encryption equation (2.2) are defined as follows:

- $A(t) \in \mathbb{R}^{(M+1) \times M}$: Information key matrix defining the signal subspace (reconstructed identically at the receiver).
- $\beta > 0$: Noise-scaling scalar, maximized to yield a severely degraded channel SNR.
- $P_A^\perp(t) \in \mathbb{R}^{(M+1) \times (M+1)}$: Orthogonal projector mapping the masking noise into the null subspace.
- $B(t) \in \mathbb{R}^{(M+1) \times M}$: Mixing key matrix used to diffuse the noise.
- $\odot$: Hadamard (element-wise) product operator.
- $k(t) \in \mathbb{R}^{M \times 1}$: Time-varying random key vector.
- $g(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$: A non-linear scalar function coupling the key signal with the plaintext. It operates element-wise on the plaintext vector $p(t)$, yielding:

$$\mathbf{g}(p(t)) = [g(p_1(t)), \dots, g(p_M(t))]^T \in \mathbb{R}^{M \times 1}, \quad t = 1, \dots, L \quad (2.3)$$

Decryption

The decryption relies on the orthogonality between $\mathcal{S}$ and $\mathcal{S}^\perp$. Projecting out all noise by applying $P_A(t) = A(t) (A(t)^T A(t))^{-1} A(t)^T$ to the ciphertext:

$$y(t) = P_A(t) x(t) = \underbrace{P_A(t) A(t)}_{=A(t)} p(t) + \underbrace{\beta P_A(t) P_A^\perp(t)}_{=0} B(t) [k(t) \odot g(p(t))] \quad (2.4)$$

Since $P_A P_A^\perp = 0$, the noise vanishes, leaving only the signal component. The plaintext is then recovered using the pseudo-inverse $A^\# = (A^T A)^{-1} A^T$:

$$y(t) = A(t) p(t) \implies \hat{p}(t) = A^\#(t) y(t) = \underbrace{A^\#(t) A(t)}_{=I} p(t) \implies \hat{p}(t) = p(t) \quad (2.5)$$

Furthermore, the projection and pseudo-inverse steps can be collapsed into a single operation. Since:

$$A^\# P_A^\perp = (A^T A)^{-1} A^T (I - A(A^T A)^{-1} A^T) = (A^T A)^{-1} A^T - (A^T A)^{-1} A^T = 0 \quad (2.6)$$

applying $A^\#$ directly to the ciphertext in equation (2.2) gives:

$$\hat{p}(t) = A^\#(t) x(t) = p(t) \quad (2.7)$$

This halves the number of matrix vector multiplications at the receiver. Importantly, the only secret required for decryption is the key matrix $A(t)$. The matrices $B(t)$, the key signal $k(t)$, and the non-linear function are used during encryption.

2.2 The Non-linear Function and Its Role

By coupling the masking noise directly to the plaintext, the non-linear function $g(\cdot)$ neutralizes differential attacks.

2.2.1 Necessity of the Non-linear Coupling

The SBE scheme builds its security on both confusion and diffusion:

- Confusion is achieved by obscuring the relationship between the plaintext and ciphertext through the non-linearity.
- Diffusion is ensured by the fact that every component of $x(t)$ depends on every component of $p(t)$ and $k(t)$, so that the statistics of the plaintext are spread throughout the ciphertext.

Let's consider the purely linear variant where the noise is independent of the plaintext:

$$x(t) = A(t) p(t) + \beta P_A^\perp(t) B(t) k(t) \quad (2.8)$$

In this case, if the same key is used to encrypt two plaintexts $p^{(1)}(t)$ and $p^{(2)}(t)$, the noise terms cancel exactly in the ciphertext differential:

$$\Delta x(t) = x^{(1)}(t) - x^{(2)}(t) = A(t) \Delta p(t) \quad (2.9)$$

This exposes $A(t)$ directly to a chosen plaintext attack, completely breaking the scheme.

In the full SBE equation (2.2), the masking noise is coupled with $g(p(t))$, so the differential becomes:

$$\Delta x(t) = A(t) \Delta p(t) + \beta P_A^\perp(t) B(t) \left[k(t) \odot \left(g(p^{(1)}(t)) - g(p^{(2)}(t)) \right) \right] \quad (2.10)$$

Because g is non-linear, the noise depends on the absolute plaintext values rather than their difference to prevent noise cancellation during key reuse. Isolating $\Delta p(t)$ therefore requires prior knowledge of the plaintexts. This makes the attack mathematically intractable.

2.2.2 Design Constraints on $g(\cdot)$

The typical choice is [5]:

$$g(v) = \frac{v}{\sqrt{1 + v^2}} \quad (2.11)$$

This function satisfies three constraints:

1. **Zero plaintext:** $g(0) = 0$, so $x(t) = 0$ when $p(t) = 0$. Silent frames produce no ciphertext, preventing any cryptanalysis in the absence of speech.
2. **Bounded derivative:**

$$g'(v) = \frac{1}{(1 + v^2)^{3/2}} \leq 1, \quad \forall v \in \mathbb{R} \quad (2.12)$$

Let ε denote a small perturbation in the input v , arising in practice from fixed-point quantization or rounding error introduced by hardware arithmetic. By the mean-value theorem, $|g(v + \varepsilon) - g(v)| \leq |\varepsilon|$, so the output error introduced by $g(\cdot)$ never exceeds the input error. This guarantees that quantization noise is never amplified as it propagates through the non-linear stage, which is critical for fixed-point hardware stability.

3. **Smooth injectivity:** $g : \mathbb{R} \rightarrow [-1, 1]$. Every plaintext value maps to a unique output, so no information is destroyed. The bounded output range $[-1, 1]$ also prevents overflow in finite precision implementations.

2.2.3 Security Against Attacks

The cryptographic robustness of SBE has been assessed against several classes of cryptanalysis attacks [11]. Table 2.1 gives a concise overview followed by the derivation of each result.

Table 2.1: Security of the proposed SBE scheme against cryptanalysis attacks.

Vulnerability	Outcome
Key mismatch (P_A sensitivity)	Decryption fails, minor projector errors ε are intentionally amplified by the factor β .
Key signal variability	Identical plaintexts yield distinct ciphertexts across segments, preventing pattern-matching and exhaustive searches.
plaintext sensitivity	small plaintext variations ($\varepsilon = 0.01$) produce entirely uncorrelated ciphertexts due to non-linear coupling.
Differential Attacks	Masking noise persists under key reuse, keeping the plaintext differential hidden and the system mathematically intractable.
	Continuous updating of the mixing matrix $A(t)$ prevents subspace estimation, as a single data sample per state provides insufficient information to characterize the signal subspace.

Key mismatch: Consider a mismatched projector $\hat{P}_A = P_A + \varepsilon I$. Applying it to the ciphertext gives:

$$\hat{P}_A x(t) = (1 + \varepsilon) A(t) p(t) + \beta \varepsilon P_A^\perp(t) B(t) [k(t) \odot \mathbf{g}(p(t))] \quad (2.13)$$

The noise term persists, scaled by $\beta \varepsilon$. Setting $\beta = O(1/\varepsilon)$ ensures that even a very small mismatch produces a result entirely different from $p(t)$.

Key signal variability: Since $k(t)$ is regenerated at each segment, two encryptions of the same plaintext under the same $A(t)$ always yield different ciphertexts. A cryptanalyst collecting N ciphertexts gains no information about whether they originate from N distinct plaintexts or fewer.

plaintext sensitivity: For $p^{(2)}(t) = p^{(1)}(t) + \varepsilon$, the ciphertexts differ in both the signal term $A(t)\Delta p(t)$ and the noise term, since $g(p^{(1)}(t)) \neq g(p^{(2)}(t))$. Even for $\varepsilon = 0.01$, inaudible to the human ear. The resulting ciphertexts are completely distinct.

Differential attack: As established in equation (2.10), encrypting two plaintexts with the same key. The noise term does not cancel. Isolating $\Delta p(t)$ requires knowing $p^{(1)}(t)$ and $p^{(2)}(t)$ individually.

2.3 Oblique Subspace-Based Encryption

In the orthogonal scheme, P_A and $P_A^\perp$ are complementary. Knowing the range subspace $\mathcal{S}$ automatically reveals the null subspace $\mathcal{S}^\perp$. This geometric coupling reduces the effective key search space for a cryptanalyst.

The oblique subspace-based encryption generalizes the orthogonal approach by replacing the orthogonal projector with an oblique projector. In this variant, the range subspace $\langle A \rangle$ and the null subspace $\langle B \rangle$ are independent of each other, so a cryptanalyst must estimate both $A(t)$ and $B(t)$ separately.

2.3.1 Oblique Projection

The oblique projection matrix into the range subspace $\langle A \rangle$ along the null subspace $\langle B \rangle$ is defined as:

$$E_{AB} = A(A^H P_B^\perp A)^{-1} A^H P_B^\perp \quad (2.14)$$

with its complementary projector:

$$E_{BA} = P_B^\perp (I - E_{AB}) \quad (2.15)$$

Unlike the orthogonal projector, E_{AB} is idempotent but not symmetric. It satisfies the key identities:

$$E_{AB}A = A \quad \text{and} \quad E_{AB}B = 0 \quad (2.16)$$

which are the algebraic properties exploited during decryption.

2.3.2 Encryption and Decryption Process

The oblique encryption equation takes the following form:

$$x(t) = \underbrace{A(t) p(t)}_{\text{Information Term}} + \underbrace{\beta B(t) [k(t) \odot \mathbf{g}(p(t))]}_{\text{Masking Noise Term}} \quad (2.17)$$

where $A(t) \in \mathbb{R}^{(M+1) \times M}$ and $B(t) \in \mathbb{R}^{(M+1) \times M}$ are both full-rank key matrices generated per

segment. Note that, unlike the orthogonal scheme (2.2), the explicit projector $P_A^\perp$ is absent, the noise term $\beta B(t)[k(t) \odot g(p(t))]$ now lies in the null subspace $\langle B \rangle$.

For decryption, the ciphertext is projected using the oblique projector E_{AB} :

$$y(t) = E_{AB} x(t) = A(t) p(t) \quad (2.18)$$

which follows directly from the identities (2.16). The plaintext is then recovered as:

$$\hat{p}(t) = A^\#(t) y(t) = p(t) \quad (2.19)$$

Table 2.2 provides a brief comparison of both SBE methods:

Table 2.2: Comparison between orthogonal and oblique SBE schemes.

	Orthogonal SBE	Oblique SBE
Noise confinement	$P_A^\perp$ forces noise into $\mathcal{S}^\perp$	$B(t)$ defines an independent null subspace $\langle B \rangle$
Projector	P_A (symmetric, orthogonal)	E_{AB} (idempotent, not symmetric)
Subspace dependency	$\langle A \rangle$ and $\mathcal{S}^\perp$ are complementary	$\langle A \rangle$ and $\langle B \rangle$ are independent
Cryptanalyst's task	Estimate $A(t)$ only	Estimate both $A(t)$ and $B(t)$
Security level	Strong	Stronger

The oblique scheme generalizes the orthogonal one. Specifically, when $\langle B \rangle = \mathcal{S}^\perp$, the oblique projector E_{AB} reduces to the orthogonal projector $P_A^\perp$, making the orthogonal scheme a special case of the oblique formulation.

In the remainder of this work, the orthogonal SBE is used as the basis for analysis, implementation, and hardware study, as it constitutes the foundational case from which extensions can be derived.

2.4 Conclusion

SBE allows to securely hide data within noise while ensuring perfect recovery through a simple linear projection. The non-linear coupling function prevents noise cancellation during key reuse, defeating differential attacks and satisfying essential signal stability constraints. The foundational subspace framework delivers a secure architecture ideally suited for real-time hardware implementation.

Key Generation

This chapter details the hardware implementation of the key generation pipeline for the Subspace-Based Encryption (SBE) system on the Zybo Z7-20 Field Programmable Gate Array (FPGA). The finalized design employs a robust hybrid architecture, integrating both Ring Oscillator-based True Random Number Generator (RO-TRNG) and ChaCha20 keystream generators. These primary entropy sources are orchestrated to continuously construct the required cryptographic key matrices under the NIST ¹ international norms.

To explore this architecture, the chapter is organized as follows:

- Section 3.2 introduces the initial Linear Feedback Shift Register (LFSR) based hardware implementation and examines its fundamental cryptographic limitations.
- Section 3.3 details the upgraded ChaCha20 hardware architecture, highlighting its pipeline design and security advantages.
- Section 3.5 explores the RO-TRNG architecture, including the hardware-level requirements to eliminate the deterministic bias.
- Section 3.7 concludes with an evaluation of the generated cryptographic bitstreams.

3.1 Random Number Generator

A Random Number Generator (RNG) is any computational or physical device designed to produce a sequence of numbers or symbols that exhibit statistical unpredictability [36]. There are two fundamentally distinct classes:

- **True Random Number Generators (TRNGs):** Derive entropy from non-deterministic physical phenomena, such as thermal noise in semiconductors, shot noise from Zener diodes, or a free-running oscillator jitter. Because their output depends on physical processes rather than any initial value, TRNGs are not reproducible by design.
- **Pseudo-Random Number Generators (PRNGs):** Are deterministic algorithms that expand a fixed initial value (seed) into a long sequence that passes statistical tests for randomness. Common implementations include Linear Feedback Shift Registers

¹National Institute of Standards and Technology

(LFSRs) and the Blum Blum Shub scheme [36].

While computationally convenient and hardware efficient, the statistical quality of a PRNG is inseparable from the unpredictability of its seed.

3.2 Hardware Implementation of the LFSR Keystream Generator

A Linear Feedback Shift Register is a shift register whose input bit is formed by a linear function of its current state, coupling a sequential register with combinational feedback logic to generate pseudo-random binary sequences [36].

3.2.1 Architecture Overview

A 32-stage Fibonacci LFSR was implemented in VHSIC Hardware Description Language (VHDL), in the Fibonacci topology, with all taps² outputs are collected and XORed together at a single feedback node, whose result is injected into the least significant bit (LSB) of the register on each clock edge.

3.2.2 Step-by-Step Hardware Implementation

The proposed design methodology comprises three principal stages: polynomial selection, architectural synthesis, and fault-state protection. An overview of this scheme is presented in Figure 3.1.

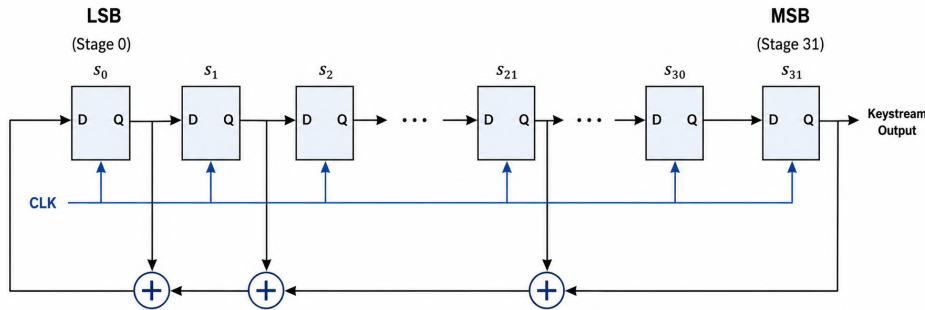


Figure 3.1: Hardware architecture of the LFSR scheme.

Step 1: Feedback Polynomial Selection

The feedback polynomial determines both the period and the statistical properties of the generated sequence. A primitive polynomial over the $\mathbb{F}_2$ ³ is required to guarantee the maximal period of $2^n - 1$ states for the n -stage register. For the 32-bit implementation, the primitive polynomial:

$$P(x) = x^{32} + x^{22} + x^2 + x^1 + x^0 \quad (3.1)$$

was selected, and the maximal length sequence of $2^{32} - 1 = 4,294,967,295$ output bits before repetition.

²specific bit position used to calculate the next input value.

³finite with two elements $\{0,1\}$, where addition is defined as XOR and multiplication as AND.

Step 2: Fibonacci Architecture Implementation

In the Fibonacci topology, the feedback bit is computed as the XOR of all register stages corresponding to the non-zero terms of the feedback polynomial and is inserted into bit 0 at each rising clock edge while the remaining bits shift one position toward the most significant end. For the polynomial of equation (3.1), the feedback function reduces to:

$$b_{fb} = s_{31} \oplus s_{21} \oplus s_1 \oplus s_0 \quad (3.2)$$

where s_i denotes the current value of stage i .

The selection of tap positions is crucial; it determines the resulting sequence period, and a poor choice leads to early repetition of states. Tables are available in the literature that list recommended tap configurations [37]. When an optimal primitive polynomial is used, an n -bit LFSR achieves its maximum theoretical period of $2^n - 1$ distinct states before the sequence repeats.

Step 3: Seed Loading and Zero-State Protection

The all-zero state is illegal for any LFSR. Once entered, the feedback of zero remains zero, and the register freezes permanently. A seed validation circuit was incorporated to detect this condition and substitute a hardcoded non-zero value (0xACE171B2) before loading the register. This protection is applied unconditionally on every seed load event, regardless of the source.

3.2.3 Cryptographic Weakness: The Berlekamp-Massey Attack

Despite their hardware efficiency, LFSRs are fundamentally unsuitable as standalone keystream generators. The feedback function relies exclusively on XOR operations, making the output sequence a linear recurrence over $\mathbb{F}_2$, a structure directly solvable by Gaussian elimination [38]. The Berlekamp-Massey algorithm exploits this linearity with a complexity of only $\mathcal{O}(n^2)$ ⁴, recovering the complete feedback polynomial of any n -stage LFSR from as few as $2n$ consecutive output bits [38]. Once the polynomial is known, the entire subsequent keystream is deterministically predictable.

Formally, given an output sequence $\{s_0, s_1, \dots, s_{2n-1}\}$, the algorithm solves:

$$s_k = \bigoplus_{i=1}^n c_i \cdot s_{k-i}, \quad k = n, \dots, 2n-1 \quad (3.3)$$

for the unknown tap coefficients c_i . For the real-time audio encryption context of this work, this vulnerability makes the LFSR entirely unacceptable, motivating the use of the non-linear ChaCha20 generator.

⁴denotes quadratic time complexity: the algorithm's execution time grows proportionally to the square of the input size n .

3.3 Hardware Implementation of the ChaCha20

The module, as shown in Figure 3.2, accepts a 256-bit secret key, a 96-bit number only used once (nonce), and a 32-bit block counter, and produces a 512-bit keystream block compliant with the IETF specification of RFC 7539⁵[39]. The output is provided through an AXI-Stream interface, a concept that will be elaborated in detail in Section 4.3, thus enabling direct integration with the downstream SBE matrix generator.

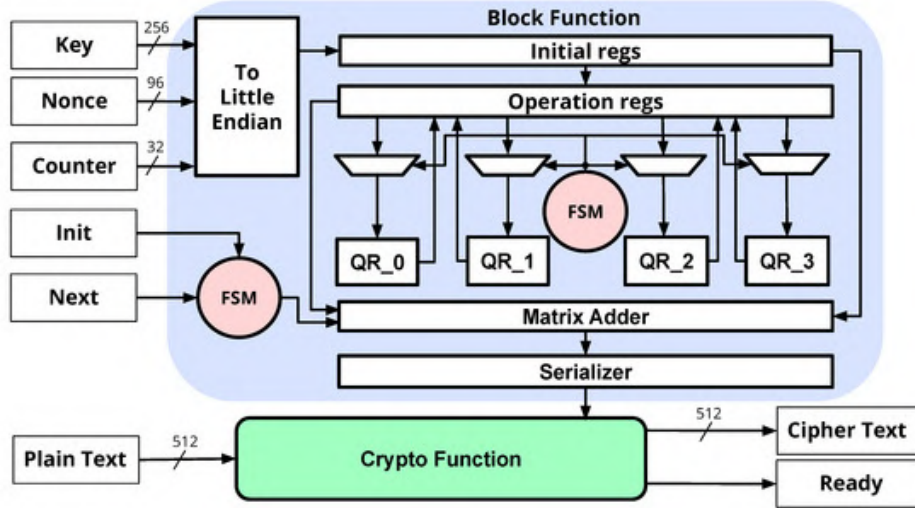


Figure 3.2: ChaCha 20 architecture overview [6].

3.3.1 Step-by-Step Hardware Implementation

The design flow is executed across four steps: initial state matrix construction, quarter-round ARX (Add-Rotate-XOR) transformation, cryptographic final word addition, and AXI-Stream output serialization combined with automated counter management.

Step 1: Initial State Construction

ChaCha20 operates on a 4×4 matrix of 32-bit words, forming a 512-bit internal state initialized as follows:

$$\begin{pmatrix} \text{const}_0 & \text{const}_1 & \text{const}_2 & \text{const}_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ \text{ctr} & n_0 & n_1 & n_2 \end{pmatrix} \quad (3.4)$$

- **Constants** (row 0): The ASCII string "expand 32-byte k" split into four 32-bit little-endian words that provide a fixed public initialization pattern that prevents weak-key degeneracies.
- **Key** (rows 1–2): The 256-bit secret key k divided into eight 32-bit words $k_0, \dots, k_7$.

⁵(Internet Engineering Task Force) RFC7539 is the standardized specification for ChaCha20-Poly1305 authenticated encryption.

- **Counter** (word 12): A 32-bit block counter initialized to zero and incremented after each 512-bit keystream block. This ensures that the same key and nonce never produce the same output twice.
- **Nonce** (words 13–15): A 96-bit nonce, providing per-message uniqueness even when the same key is reused.

In the early stages of the project, constant, pre-shared nonces were used for each data frame. This initialization is performed in a single clock cycle and establishes the baseline state S_0 for the round function.

However, to meet the proper security standards of the ChaCha20 algorithm, this approach was later modified. The system now uses a TRNG to generate a unique nonce for each frame. The design and implementation of this TRNG are detailed in Section 3.5.

Step 2: Quarter-Round Pipeline Stages

The core transformation is the quarter-round $QR(a, b, c, d)$, which applies four sequential ARX (Add-Rotate-XOR) operations to four 32-bit state words:

$$a += b; \quad d \oplus= a; \quad d \lll= 16 \quad (3.5)$$

$$c += d; \quad b \oplus= c; \quad b \lll= 12 \quad (3.6)$$

$$a += b; \quad d \oplus= a; \quad d \lll= 8 \quad (3.7)$$

$$c += d; \quad b \oplus= c; \quad b \lll= 7 \quad (3.8)$$

Here, $+=$ denotes 32-bit modular addition, $\oplus=$ bitwise XOR and $\lll=$ a left barrel rotation. The combination of modular addition over $\mathbb{Z}_{2^{32}}$ and XOR over $\mathbb{F}_2$ introduces non-linear mixing that is absent in LFSR-based designs, while also mapping efficiently onto FPGA fabric: each addition leverages carry-chain logic, each rotation is a zero-cost wire permutation, and each XOR requires only a single LUT layer.

Each cell in Figure 3.3 visually identifies one equation in the ARX quarter-round.

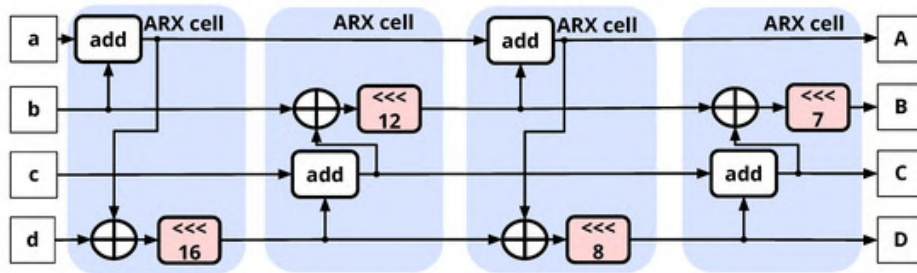


Figure 3.3: ChaCha20 quarter-round algorithm [6].

The full ChaCha20 block applies 20 rounds, structured as 10 alternating column and diagonal double rounds:

- **Column round:** The quarter-round is applied to the four columns of the state matrix:

$$QR(s_0, s_4, s_8, s_{12}), \quad QR(s_1, s_5, s_9, s_{13}), \quad QR(s_2, s_6, s_{10}, s_{14}), \quad QR(s_3, s_7, s_{11}, s_{15}) \quad (3.9)$$

- **Diagonal round:** The quarter-round is applied to the four main diagonals:

$$\text{QR}(s_0, s_5, s_{10}, s_{15}), \text{QR}(s_1, s_6, s_{11}, s_{12}), \text{QR}(s_2, s_7, s_8, s_{13}), \text{QR}(s_3, s_4, s_9, s_{14}) \quad (3.10)$$

Step 3: Final Word Addition

After all 20 rounds, the resulting state S_{20} is added word-by-word to the original initial state S_0 :

$$\text{keystream block} = \text{ChaCha20_block}(\text{key}, \text{counter}, \text{nonce}) = S_{20} + S_0 \quad (3.11)$$

This final addition is critical to the one-way construction. An attacker cannot invert the transformation to recover S_0 from the keystream output alone, since the initial state is unknown.

Step 4: AXI-Stream Output and Counter Management

The 512-bit keystream block is serialized and presented on the AXI-Stream interface. A counter register is automatically incremented after each block is consumed. When the counter reaches 2^{32} , a flag on a status port is asserted to refresh the key or nonce before further encryption.

3.3.2 Known Cryptanalytic Attacks on ChaCha20

Despite over fifteen years of sustained security evaluation, all known attacks against ChaCha20 are limited to weakened, reduced-round versions. To date, breaking the full 20-round cipher remains completely impossible with modern computing power. Table 3.1 summarizes the most significant results.

Table 3.1: Summary of known cryptanalytic attacks on reduced-round ChaCha20.

Year	Attack Type	Rounds	Complexity	Reference
2008	Differential (PNB)	6	2^{140}	Aumasson et al. [40]
2008	Differential (PNB)	7	2^{231}	Aumasson et al. [40]
2015	Chosen-IV differential	7	$2^{238.9}$	Maitra [41]
2020	Differential-linear	7	$2^{231.9}$	Coutinho & Neto [42]
2022	Differential-linear	7	$2^{221.95}$	Dey et al. [43]
2023	PNB-based differential	6	$2^{99.48}$	Dey et al. [44]
2024	Multi-step key recovery	6.5	$> 2^{128}$	Garai & Dey [45]
Full ChaCha20 (20 rounds)		20	$> 2^{256}$	No attack known

Three major attack families have been investigated against ChaCha20:

- **Differential and Differential-Linear Cryptanalysis:** It is the dominant attack family. All results exploit probabilistic differential characteristics in the quarter-round to recover key bits, often using the Probabilistic Neutral Bits (PNB) technique to reduce the effective key search space [40]. Despite steady progress over fifteen years, the best known result reaches only 7 rounds at complexity $2^{221.95}$ far beyond any computational reach [43]. A

2023 attack on 6 rounds achieved $2^{99.48}$ [44], the first to fall below $2^{k/2}$ for $k = 256$, yet this affects only 30% of the full round count.

- **Rotational Cryptanalysis:** Attempts to exploit rotational invariants through ARX operations, but ChaCha20’s fixed constant row forces all inputs into distinct orbits of the state space. This makes the attack difficult to extend beyond a handful of rounds [46].
- **Time-Memory-Data Tradeoff:** Attacks are theoretically applicable to any stream cipher, but become impractical for ChaCha20 when the encrypted data volume is bounded below 2^{96} blocks. The constraint enforced by the 96-bit nonce for every execution [47].

The security margin of ChaCha20 can be quantified as:

$$\text{Security Margin} = \frac{N_{\text{full}} - N_{\text{best attack}}}{N_{\text{full}}} \times 100 = \frac{20 - 7}{20} \times 100 = 65\% \quad (3.12)$$

placing ChaCha20 among the most conservatively designed stream ciphers currently in use.

3.4 Comparative Analysis Between LFSR and ChaCha20

This section formalizes the comparison between LFSR-based and ChaCha20-based keystream generation from both cryptanalytic and hardware perspectives that provides the quantitative basis for the design choice considered in this work.

3.4.1 Cryptanalytic Comparison

As established in the preceding analysis, the linear feedback of an LFSR renders it highly susceptible to standard algebraic attacks, whereas the non-linear ARX architecture of ChaCha20 eliminates such threats. Table 3.2 provides a direct comparison of their respective cryptographic strengths.

Table 3.2: Cryptanalytic comparison between LFSR and ChaCha20.

Criterion	LFSR (n -stage)	ChaCha20 (20 rounds)
Feedback type	Linear (XOR only)	Non-linear (ARX)
Best known attack	Berlekamp-Massey	Differential (7 rounds)
Data required to break	$2n$ bits	$> 2^{221}$ operations
Attack complexity	$\mathcal{O}(n^2)$	$> 2^{221.95}$
Immune to linear/algebraic attacks	No	Yes
Security margin	None	65% (13/20 rounds)

3.4.2 Hardware Implementation Comparison

Table 3.3 contrasts the implementation metrics of both generators on the Zybo Z7-20.

Both generators satisfy the real-time audio requirements during the implementation, so the keystream generation does not limit system performance. At 125 MHz, an LFSR generates

Table 3.3: Hardware implementation comparison on Xilinx Zynq-7000 (Zybo Z7-20).

Criterion	LFSR	ChaCha20
LUT count (typical)	n (one per stage)	$\sim 1500\text{--}3000$
Flip-flop count	n	~ 512
DSP slices	0	0 (pure LUT adders)
Pipeline stages	1	20
Latency	1 cycle	20 cycles
Throughput (125 MHz)	1 bit/cycle	512 bits/cycle
Key/nonce support	Seed only	256 bit key + 96 bit nonce
Counter support	None	32 bit block counter

12.5 MB/s, while a pipelined ChaCha20 delivers 6.4 GB/s. Although the LFSR is structurally trivial compared to ChaCha20’s 512-bit state and 20-stage pipeline, its minimal hardware cost is outweighed by its poor cryptographic security. This makes the ChaCha20 the only viable keystream source for our proposed SBE pipeline.

3.5 Hardware Implementation of the R0-TRNG

Since the SBE decryption stage operates without the need for prior knowledge or alignment of B and K , these matrices are obtained directly from a TRNG instead of from the seeded ChaCha20 pipeline. To provide this functionality, a Ring Oscillator (RO)-based TRNG was implemented on the Zybo Z7-20, which also functions as the nonce generator for ChaCha20.

Ring oscillators exploit the thermal jitter inherent in CMOS inverter chains, where the oscillation period varies unpredictably from cycle to cycle due to non-deterministic noise processes. This provides a source of physical entropy that is entirely independent of any software state or deterministic algorithm [48].

3.5.1 Architectural Overview

The design consists of 10 parallel ring oscillator stages, each built from an odd-length chain of 5 inverters closed in a feedback loop. The outputs of all these stages are combined through an XOR reduction tree before being sampled by the system clock, as shown in Figure 3.4. This ensures the decorrelation between individual loop oscillators [49].

This design will afterward be revised to incorporate 16 parallel ring oscillator stages, each built from an odd-length chain of three inverters, in order to meet the NIST standards (Section 3.7). The specific number of inverters per stage and the total number of stages were selected based on the conclusions reported in [50].

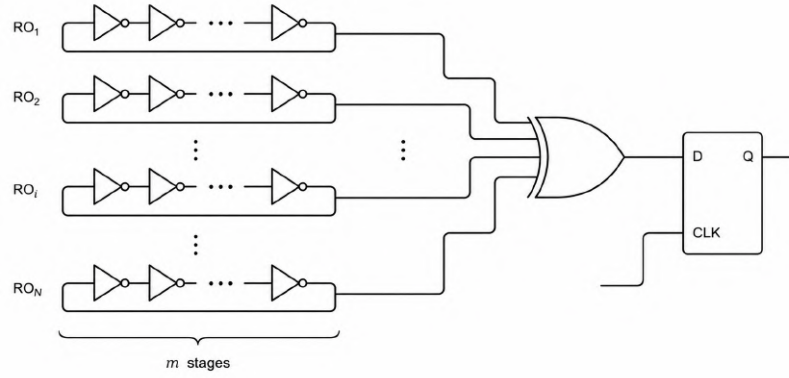


Figure 3.4: True random number generator scheme.

3.5.2 Step-by-Step Hardware Implementation

The entropy generation design flow consists of two foundational stages: a ring oscillator (RO) core and a multistage XOR network designed to sample and combine the discrete RO outputs. Furthermore, a post-processing step was introduced later in the project to bridge a statistical gap, providing the necessary enhancements to ensure the final system fully complies with NIST standards.

Step 1: Ring Oscillator Core

Each stage is a 5-inverter chain connected in a closed feedback loop. The odd number of inversions guarantees sustained oscillation. The feedback always inverts the current output, preventing the circuit from settling into a stable state.

In VHDL, the ring oscillator chain is declared with the `KEEP` and `DONT_TOUCH` synthesis attributes to prevent the optimizer from merging or absorbing the inverter chain into surrounding logic, which would destroy the jitter structure.

Step 2: Multi-Stage XOR Combination and Sampling

The 10 ring oscillator stages are created as separate instances, each assigned a unique routing constraint to avoid identical placement. Their outputs are then combined using a balanced XOR tree, merging the uncorrelated jitter contributions from all 10 stages into one signal:

$$b_{\text{raw}}[k] = \bigoplus_{i=1}^{10} \text{RO}_i[k] \quad (3.13)$$

This combined signal is sampled on the rising edge of the system clock to produce the raw bit stream $\{b_{\text{raw}}[k]\}$. The sampling clock is asynchronous to the RO frequencies, ensuring that each captured bit reflects an independent phase of the free-running oscillation.

Step 3: Post-processing

Raw bit streams generated directly from Ring Oscillators often exhibit statistical imbalances that fail to meet strict cryptographic standards, as illustrated in Figure 3.5, which reveals a non-uniform pattern rather than the flat distribution mandated for secure cryptography. To address this, we evaluated a post-processing technique.

Von Neumann Debiasing

Initial characterization of the raw bit stream against the NIST SP 800-22 test suite revealed a statistically significant bit bias. This pattern arises from a known structural limitation of RO-based TRNGs: asymmetries in the CMOS fabrication process cause individual inverter stages to exhibit slightly unequal rise and fall times. This physical imbalance changes the long-run average of the sampled bits toward either zero or one [49].

To counter this, Von Neumann debiasing was applied as a primary post-processing layer solution [51]. To eliminate the bias, the corrector operates on consecutive pairs (b_{2k}, b_{2k+1}) of raw bits according to the following rule:

$$\text{output} = \begin{cases} 1 & \text{if } (b_{2k}, b_{2k+1}) = (0, 1) \\ 0 & \text{if } (b_{2k}, b_{2k+1}) = (1, 0) \\ \text{discard} & \text{if } b_{2k} = b_{2k+1} \end{cases} \quad (3.14)$$

For any biased but independently distributed bit source where $\Pr[b = 1] = p \neq 0.5$, the pairs $(0, 1)$ and $(1, 0)$ occur with an equal probability of $p(1 - p)$. Retaining only these pairs theoretically produces an unbiased stream.

Although the resulting throughput reduction still satisfies the requirements for key-generation refresh rates, further evaluation revealed that the output continued to exhibit statistical irregularities. As shown in Figure 3.6, the distribution remains non-uniform, with certain values occurring disproportionately more frequently than others. Consequently, it became evident that algorithmic debiasing alone was inadequate.

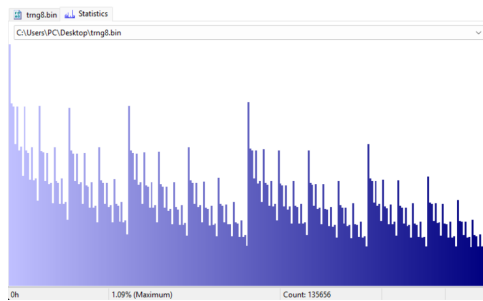


Figure 3.5: TRNG output bit distribution.

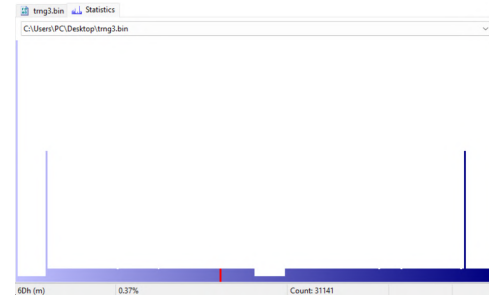


Figure 3.6: TRNG with post-processing output bit distribution.

Consequently, this approach was superseded by the hardware-level architecture, containing synchronous D-flip-flop (DFF) sampling and a binary XOR tree.

Step 4: Synchronous DFF Sampling and XOR Binary Tree

The remaining statistical flaws were traced directly to how the oscillator rings were sampled and combined. In standard asynchronous designs, the continuous, high-frequency outputs of multiple ROs are fed directly into a single XOR gate. This floods the gate with asynchronous transitions, pushing it into an overload state that distorts the output bias.

To completely resolve this issue, the purely algorithmic debiasing was replaced with a structured hardware enhancement. This proposed architecture was implemented according

to [50] and is illustrated in Figure 3.7. It relies on three key design principles to stabilize the entropy generation:

- **Synchronous DFF Sampling:** As depicted in the schematic 3.7, a DFF (1 through 16) is placed immediately after each of the 16 ROs. These DFFs sample the highly unstable RO signals at a fixed system clock edge to ensure that the downstream logic is not overloaded by raw oscillator transitions.
- **Binary XOR Tree Structure:** Instead of directing all asynchronous signals into a single gate, the synchronized DFF outputs are merged using a cascading binary tree of XOR gates. Because each gate processes only two synchronized input signals at a time, the processing load is uniform and strictly tied to the sampling clock. The final bit signal is then passed through a final DFF sampler before becoming the stable output.
- **Optimized FPGA Routing:** To maximize entropy generation, the individual rings were restricted to exactly three inverters, which yields the highest frequency, averaging around 400 MHz.

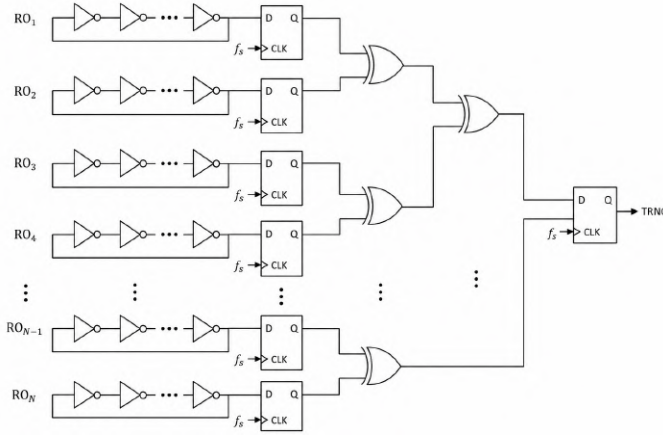


Figure 3.7: Design flow for the updated RO hardware architecture.

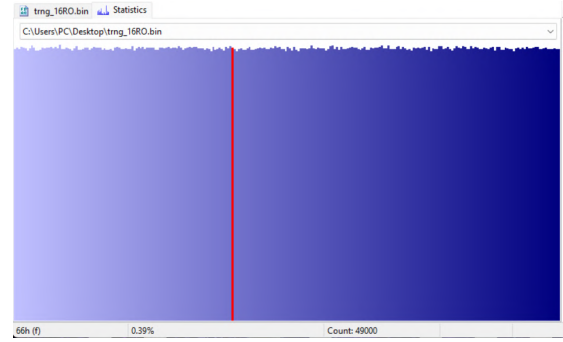


Figure 3.8: TRNG with updated RO hardware architecture output bits distribution.

By stabilizing the signal extraction process and structuring the XOR, this hardware level successfully eliminated the remaining biases as the uniform distribution is illustrated in Figure 3.8. The corresponding NIST evaluation results are comprehensively analyzed and discussed later in Section 3.7.

3.6 Hybrid Key Generation: ChaCha20/RO-TRNG

The SBE model relies on the generation of three structured matrices, A , B , and K , whose coefficients must be cryptographically unpredictable and dynamically refreshed for every audio frame. The system employs a parallelized hardware architecture that synchronizes both the RO-TRNG and the ChaCha20 stream cipher.

Within this framework, the RO-TRNG serves a dual, simultaneous purpose. One stream directly exploits physical thermal jitter to provide non-deterministic entropy for the B and K matrices. A critical architectural advantage of the SBE model is that B and K are used exclusively during the encryption phase.

Concurrently, another instance of the RO-TRNG supplies a true-random nonce to the ChaCha20 module. Initialized with this physical entropy, the ChaCha20 is specifically tasked with constructing the A matrix. Because the receiver must reconstruct A exactly to perform the correct subspace projection during decryption, A must be generated deterministically from the transmitted nonce.

By executing these hardware processes simultaneously, the system provides physical randomness for encryption while ensuring the strict determinism required for decryption, all without introducing latency bottlenecks for incoming audio frames. Figure 3.10 illustrates the IP block architecture of the SBE matrix generation layer, and Figure 3.9 presents the corresponding RTL IP block integrated into the Vivado environment.

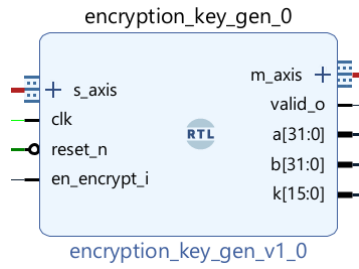


Figure 3.9: RTL IP core for the SBE key generation module within Vivado.

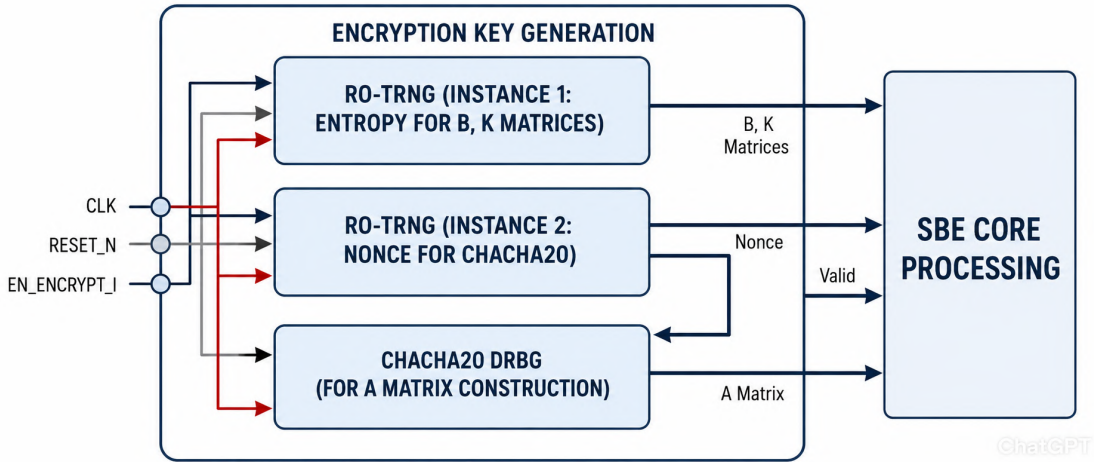


Figure 3.10: Hybrid key generation architecture: ChaCha20 for A and RO-TRNG for nonce, B and K .

After the matrix generation layer has derived these parameters, the SBE applies equation (2.2). This coordinated hardware routing results in a ciphertext that exploits true physical randomness to ensure the confidentiality of the audio stream.

3.7 Randomness Validation: NIST SP 800-22 Test Suite

NIST is the reference statistical test suite issued by the National Institute of Standards and Technology for evaluating the randomness of bit sequences produced by hardware or software

generators intended for cryptographic use. It comprises 15 distinct statistical tests, each targeting a specific weakness that a non-random generator is likely to produce.

3.7.1 Overview of the NIST SP 800-22 Framework

The NIST SP 800-22 suite evaluates randomness through 15 specialized tests. The **Frequency** and **Block Frequency** tests ensure an equal distribution of zeros and ones globally and locally, while **Runs** and the **Longest Run of Ones** detect oscillations or identical bit sequences that deviate from expected patterns. Structural integrity is further assessed via the **Binary Matrix Rank** for linear dependence, the **Discrete Fourier Transform (DFT)** for periodic patterns, and **Template Matching (Non-overlapping and Overlapping)** to find irregular m -bit occurrences. **Complexity** is measured using Maurer’s Universal Statistical test for compressibility, **Linear Complexity** to ensure the equivalent LFSR is sufficiently long, and **Serial** and **Approximate Entropy** tests for uniform pattern distribution. Finally, the **Cumulative Sums** and **Random Excursion** tests analyze the sequence as a random walk to detect deviations from expected path behaviors [52].

Each test produces a p -value under the hypothesis of randomness-density in which the sequence is randomly generated. A sequence passes a given test at significance level $\alpha = 0.01$ when $p\text{-value} \geq \alpha$. For 100 independent test sequences, each has 10^6 bits of data, the minimum acceptable pass rate per test is 96%, since statistical fluctuations allow up to 4 sequences to fail by chance alone [53].

The NIST SP 800-22 suite has been widely adopted as the primary validation tool for FPGA-based TRNGs [49, 54]. Passing its full tests is considered a necessary condition for the cryptographic deployment of any entropy source.

3.7.2 NIST Analysis and Empirical Results

To validate the statistical integrity of the developed key generation pipeline, bitstreams of 10^8 bits were collected from both the implementation of ChaCha20 and the Ring Oscillator (RO) using the Vivado⁶ design tool and subjected to the NIST tests. The results are summarized in Table 3.4, that provide the statistical validity of each module.

⁶Vivado is a software tool used to design, simulate, and implement hardware systems for Xilinx boards.

Table 3.4: NIST Results: ChaCha20 (Fixed/TRNG Nonce) and RO-TRNG.

Statistical Test		ChaCha20 (Fixed Nonce)		RO-TRNG		ChaCha20 (TRNG Nonce)	
		<i>p</i> -value	Status	<i>p</i> -value	Status	<i>p</i> -value	Status
Frequency (Monobit)		0.637119	PASS	0.401199	PASS	0.474986	PASS
Block Frequency		0.554420	PASS	0.494392	PASS	0.534146	PASS
Cumulative Sums (Forward)		0.637119	PASS	0.474986	PASS	0.025193	PASS
Cumulative Sums (Reverse)		0.474986	PASS	0.366918	PASS	0.249284	PASS
Runs		0.224821	PASS	0.534146	PASS	0.202268	PASS
Longest Run of Ones		0.897763	PASS	0.991468	PASS	0.202268	PASS
Binary Matrix Rank		0.262249	PASS	0.719747	PASS	0.319084	PASS
DFT (Spectral)		0.437274	PASS	0.023545	PASS	0.202268	PASS
Non-overlapping Template Matching		0.0013–0.9879	PASS	0.0002–0.9879	PASS	0.0089–0.9879	PASS
Overlapping Template Matching		0.991468	PASS	0.080519	PASS	0.935716	PASS
Maurer’s Universal Statistical		0.066882	PASS	0.935716	PASS	0.181557	PASS
Approximate Entropy		0.350485	PASS	0.759756	PASS	0.437274	PASS
Serial	Test 1	0.137282	PASS	0.474986	PASS	0.911413	PASS
	Test 2	0.401199	PASS	0.699313	PASS	0.236810	PASS
Linear Complexity		0.996335	PASS	0.657933	PASS	0.616305	PASS
Random Excursions		0.2430–0.7887	PASS	0.0956–0.9411	PASS	0.0462–0.9573	PASS
Random Excursions Variant		0.0956–0.8486	PASS	0.0298–0.9411	PASS	0.0213–0.9705	PASS

Based on the results, both ChaCha20 nonce strategies yield statistically equivalent output quality, successfully passing all NIST tests. While a fixed nonce is sufficient for isolated statistical validation, the RO-TRNG-seeded variant is employed in the deployed system to eliminate nonce reuse vulnerabilities and ensure cryptographic freshness across communication sessions.

For complete transparency and further analysis, the comprehensive NIST test logs and overall statistical results for all configurations are provided in the Appendix.

3.8 Conclusion

This chapter established the design, implementation, and validation of the SBE key generation on the Zybo Z7-20.

The design process followed a clear developmental trajectory. Initially prototyped using a 32-bit Fibonacci LFSR as a performance baseline, the architecture was upgraded to a ChaCha20-based expansion to enhance cryptographic strength across all operands. The final system used a hybrid architecture: a ChaCha20 generator provides coefficients for the A matrix, while the RO-TRNG provides genuine physical entropy for the B and K matrices. This specific design choice leverages the SBE model’s unique property since B and K are not required for the decryption process, allowing the integration of true randomness.

Communications Channel

The implementation of a secure, real-time audio transmission system on a programmable System-on-Chip (SoC) requires a specified communications channel. This chapter establishes the architectural design of the end-to-end communications, explicitly addressing the trade-off between high-fidelity data acquisition and low-latency cryptographic processing throughput. By interfacing physical analog domains with digital transport infrastructure, the proposed framework optimizes data-flow paths to achieve reliable performance.

The chapter is organized as follows:

- Section 4.1 establishes the fundamental digital signal processing and information theory.
- Section 4.2 outlines the physical audio interface and its clocking hierarchy.
- Section 4.3 details the AXI4-Stream protocol and its mechanics.
- Section 4.4 maps the proposed communications scheme deployed across the SoC fabric.
- Section 4.5 justifies the network topology and transmission modes.

4.1 Theoretical Foundations

The parameters of the proposed communications channel derive directly from established theorems in digital signal processing and information theory.

4.1.1 Audio Sampling and Quantization

To ensure high-fidelity digitization of the analog signal, the system parameters are established by the Nyquist-Shannon sampling theorem [55, 56]. An analog signal can be reconstructed provided the sampling frequency F_s that satisfies $F_s > 2 F_{\max}$. Given the human auditory range of 20 kHz, the system operates at a standard $F_s = 48$ kHz per channel, providing a sufficient guard band.

Following sampling, the continuous amplitude is quantized into 2^Q discrete levels. The theoretical maximum Signal-to-Quantization-Noise Ratio (SQNR) is approximated as [57]:

$$\text{SQNR} \approx 6.02 \cdot Q + 1.76 \quad (\text{dB}) \quad (4.1)$$

By selecting a bit depth of $Q = 24$ bits, the system achieves approximately 146 dB. Because Subspace-Based Encryption (SBE) injects high-energy orthogonal noise to mask the signal (Section 1.7), this resolution ensures that the combined amplitude is handled without causing amplitude saturation in the fixed-point format.

4.1.2 Transmission Bit Rate

The required transmission source bit rate R for the encrypted stereo audio stream at 2×48 kHz is evaluated under both the fixed and dynamic nonce configurations as follows:

$$R = 2 \times 64 \times 48,000 \approx 6.14 \text{ Mbps} \quad (\text{Fixed nonce}) \quad (4.2)$$

$$R = 2 \times (64 + 96) \times 48,000 \approx 15.36 \text{ Mbps} \quad (\text{Dynamic nonce}) \quad (4.3)$$

The use of a Gigabit Ethernet link (1000 Mbps) provides much more bandwidth than is actually needed for the transmitted data. The latter guarantees that the end-to-end latency is governed exclusively by hardware buffering rather than channel congestion.

4.2 Audio Interface and Clocking Scheme

The physical acquisition layer utilizes the Inter-IC Sound (I²S) protocol, a synchronous serial interface designed for communicating digital audio between an FPGA fabric and an external codec. The SSM2603 codec on the Zybo Z7-20 is driven by a strict clock hierarchy established from the 125 MHz system clock through the Mixed Mode Clock Manager (MMCM), which is responsible for generating stable and low-jitter Master Clock (MCLK), alongside the derived Bit Clock (BCLK) and Left Right Clock (LRCLK) that synchronize the serial data transmission.

4.2.1 Clock Hierarchy

The relationship between the audio sampling frequency F_s and the three I²S clock signals is defined by the SSM2603 hardware requirements [58], as shown in Figure 4.1:

$$\begin{cases} F_{\text{MCLK}} = 256 \times F_s = 12.288 \text{ MHz} \\ F_{\text{BCLK}} = \frac{F_{\text{MCLK}}}{4} = 3.072 \text{ MHz} \\ F_{\text{LRCLK}} = \frac{F_{\text{BCLK}}}{64} = 48 \text{ kHz} \end{cases} \quad (4.4)$$

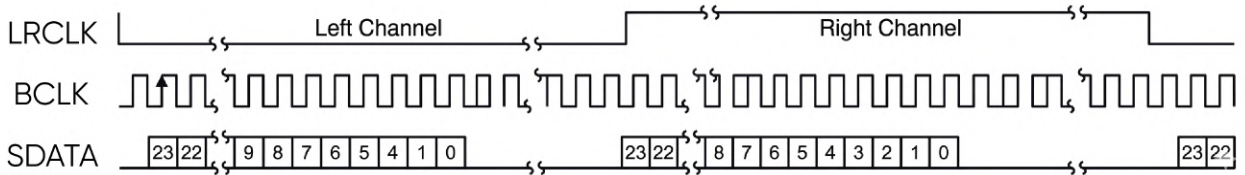


Figure 4.1: I²S protocol.

The three signals serve different roles within the protocol:

- **MCLK (Master Clock):** Reference clock fed directly to the SSM2603. It sets the codec's internal oscillator and determines the sampling rate.
- **BCLK (Bit Clock):** Clocks each serial data bit. Once complete, the audio sample bits shift out per channel between consecutive LRCLK transitions.
- **LRCLK (Left Right Clock):** Toggles at $F_s = 48\text{ kHz}$. A falling edge marks the start of the left channel sample, and a rising edge marks the start of the right channel sample. Both edges are exploited in this implementation, doubling the effective sample delivery rate to 96 kHz.

The I²S interface consists of these three signals plus the serial data line (SDIN/SDOUT). In this implementation, the codec is configured for 24-bit audio resolution.

4.3 AXI4-Stream Standard Interface

Every internal data transfer in the system, from the I²S to the key generator and the encryption engine, passes through to the decryption engine, is carried over the AXI4-Stream protocol defined in the ARM AMBA AXI-Stream specification [59]. Its handshake rules govern the timing and back-pressure behavior of every module in the design.

4.3.1 Protocol Overview

AXI4-Stream is a unidirectional, point-to-point data bus intended for high-throughput streaming connections where neither addressing nor bursting is required. The protocol defines three mandatory signals on each channel [59]:

- **TDATA** carries the payload. In this design, the width is either 32 bits (plaintext audio samples) or 64 bits (encrypted ciphertext pairs).
- **TVALID** is asserted by the source to indicate that the data on TDATA is stable and ready to be consumed.
- **TREADY** is asserted by the destination to indicate that it can accept a transfer on this cycle.

A data transfer occurs on every rising edge of the system clock at which both TVALID and TREADY are simultaneously declared, as shown in Figure 4.2. This mechanism, known as back-pressure, involves backward flow from the output to the input. If the downstream module cannot accept the data, it de-asserts TREADY, stalling the upstream source without losing any sample. Neither the source nor the destination may drop a transfer once TVALID has been raised. The source must hold a stable TDATA and TVALID signals until the transfer is accepted.

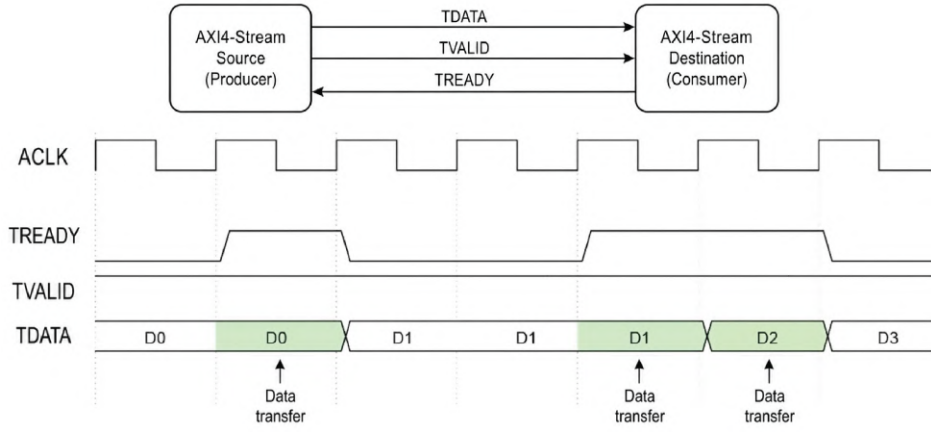


Figure 4.2: AXI4-Stream handshake

For example, in the encryption stage, if the downstream First-In First-out (FIFO) fills, the engine holds the data until the current sample is successfully forwarded. Similarly, the decryption engine drops TREADY to pause incoming data while the restoring divider calculates the pseudo-inverse. This hardware handshake prevents data overwrites and ensures both FPGA boards remain perfectly synchronized without requiring any complex logic.

4.4 Proposed communications Scheme

A primary contribution of this work is the development of an end-to-end system that integrates real-time audio acquisition, hardware-accelerated cryptographic modules (for both encryption and decryption), and efficient network encapsulation. Implemented on identical Zybo Z7-20 SoC FPGA platforms, the architecture is divided into distinct transmitter and receiver pipelines.

It should be noted that the unidirectional pipelines detailed in this section constitute the foundational backbone of the project. While initially designed to validate the core cryptographic dataflow and hardware routing, this baseline scheme was evolved into a complete full-duplex communications system. This evolution occurred in two stages: initially operating as a TCP-based full-duplex system utilizing a single session-generated nonce and advancing into a UDP-based full-duplex architecture featuring dynamic, per-frame nonce generation. Nevertheless, all full-duplex functionalities remain dependent on the fundamental hardware and data routing established in this scheme.

4.4.1 Transmitter Architecture

The transmitter is designed to convert a raw analog audio signal into a secure, packetized network stream, as illustrated in Figure 4.3. The dataflow originates at the SSM2603 audio codec, traverses the AXI4-Stream fabric into the Subspace-Based Encryption (SBE) hardware module, and ends in the software network stack for external transmission.

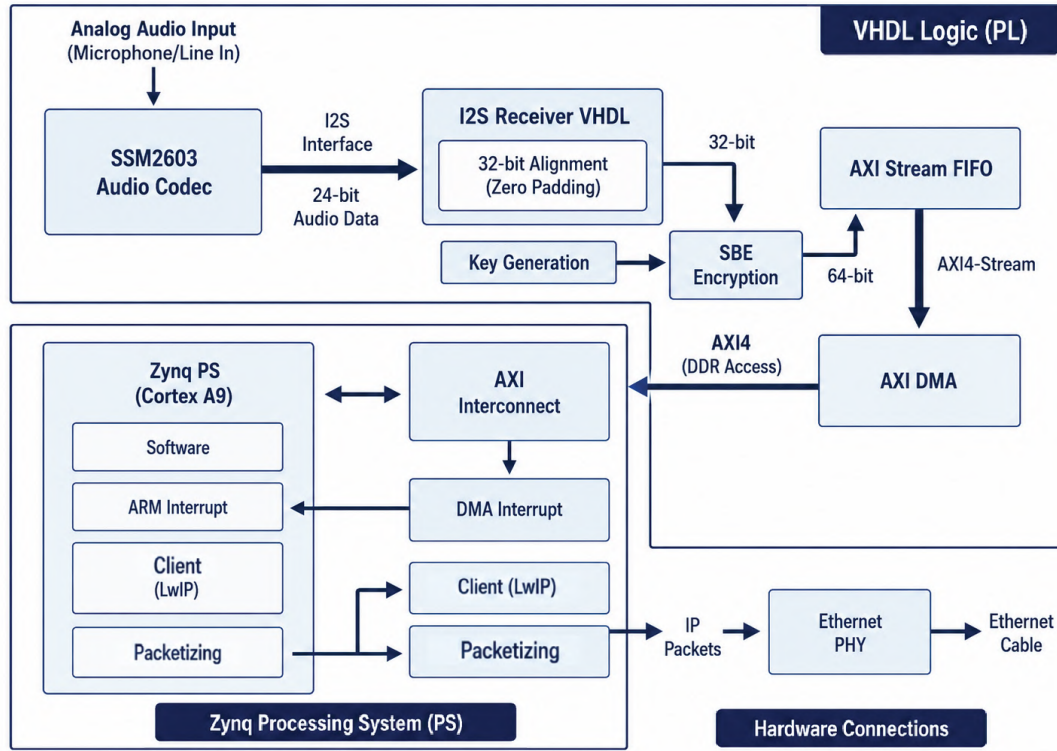


Figure 4.3: Proposed transmitter architecture.

The pipeline executes the following sequential operations:

1. **Digitization and Alignment:** Incoming 24-bit I²S audio samples are aligned and zero-padded to conform to the standard 32-bit data width of the AXI4-Stream bus.
2. **In-Stream Cryptography:** The aligned plaintext samples are routed directly into the custom SBE IP core. Here, the hardware merges the audio payload with the time-varying keys to produce 64-bit ciphertext samples.
3. **Direct Memory Access (DMA):** To preserve computational bandwidth, the encrypted payloads are accumulated in a hardware FIFO before being burst-transferred via the AXI DMA engine directly into the DDR memory, entirely bypassing Central Processing Unit (CPU) intervention.
4. **Network Encapsulation:** The Lightweight IP (lwIP) network stack, executing on the ARM Cortex-A9 processor within the Processing System (PS), retrieves the ciphertext from memory, encapsulates it into network segments (such as TCP segments or UDP datagrams), and delivers the data through the Gigabit Ethernet.

4.4.2 Receiver Architecture

The receiver performs the inverse operational sequence to accurately reconstruct the secured audio stream. Incoming network segments are captured and reassembled by the PS processor, pushed to the DDR memory, and streamed via DMA into the Programmable Logic (PL) part to provide data to the hardware decryption.

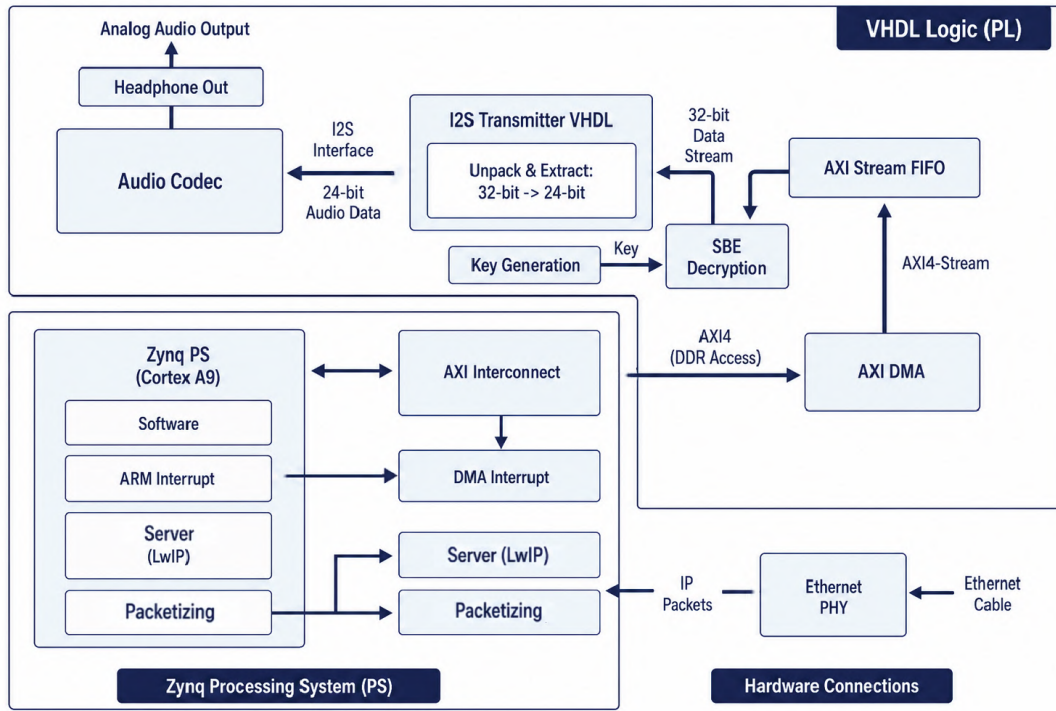


Figure 4.4: Proposed receiver architecture.

An important component of the receiver design is the AXI4-Stream FIFO, which acts as an asynchronous buffer between the DMA output and the digital-to-analog converter (DAC). This component is essential for absorbing any transport jitter induced by network latency fluctuations or DMA burst timing constraints. By doing so, it guarantees a continuous and uninterrupted plaintext samples to the downstream audio logic, thereby eliminating the risk of data starvation or missing audio samples.

4.5 Network protocol and transmission mode selection

This section establishes the network layers' infrastructure supporting the communications channel, alongside an architectural analysis of the transmission modes utilized throughout the development.

4.5.1 Network protocols

The Open Systems Interconnection (OSI) model serves as a conceptual framework that standardizes network communications by organizing functions into hierarchical layers. This analysis focuses on three foundational layers: the physical layer, the data link layer, and the transport layer. Each of these layers is illustrated in the Figure 4.5.

The Physical Layer

At the hardware level, the physical layer implements the 1000BASE-T Ethernet standard over Category 6 (Cat-6) Unshielded Twisted Pair (UTP) cabling. To achieve a transmission throughput of 1 Gbps, the architecture utilizes Pulse Amplitude Modulation 5 (PAM-5), which encodes multiple bits per signal change. Furthermore, a Cyclic Redundancy Check

(CRC) is implemented to detect signal corruption and verify frame integrity before data is passed higher up the stack.

The Data Link Layer

The data link layer is evaluated based on the fragmentation strategies required for system operations. This layer handles the framing and division of data blocks, ensuring that the payloads conform to Maximum Transmission Unit (MTU) constraints and preparing them for efficient physical layer propagation.

The Transport Layer: TCP and UDP

The transport layer manages end-to-end data delivery across the network. It utilizes either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), depending on specific application requirements.

- TCP is a connection-oriented protocol designed to ensure reliable and error-checked data delivery. It establishes a formal connection via a handshake mechanism and utilizes sequence numbers and acknowledgments to ensure that all packets are received correctly and in the proper order. This introduces protocol overhead, making TCP essential for applications that need perfect data fidelity.
- However, UDP is a connectionless, minimal protocol that transmits datagrams without establishing a prior connection or guaranteeing delivery. It eliminates acknowledgment mechanisms and flow control that reduce latency and protocol overhead. UDP is preferred for real-time or streaming applications where speed is critical and minor packet loss does not reduce the effectiveness of the system.

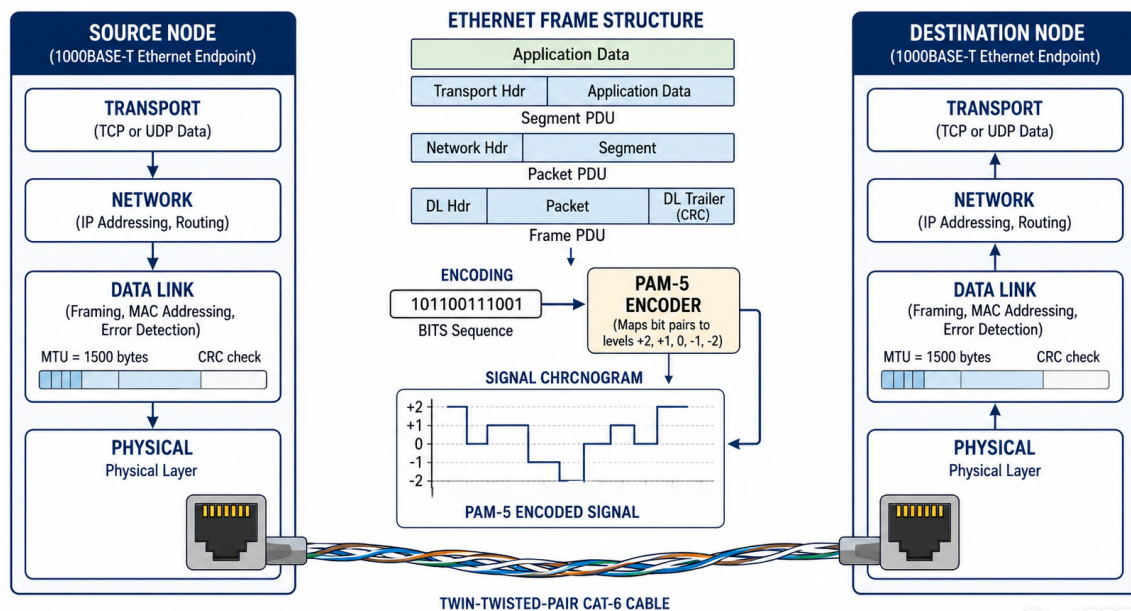


Figure 4.5: TCP/IP protocol layers scheme.

4.5.2 Transmission Modes

In data communications systems, the transmission mode dictates the directionality of signal exchange between connected nodes (in the context of this architecture, FPGAs). Depending on hardware capabilities and channel requirements, communications interfaces operate in one of three modes [7]:

- **Simplex transmission mode:** Enforces a unidirectional flow of data. As illustrated in Figure 4.6, the physical layer relies on a single-strand cable where one device functions exclusively as the transmitter and the other as the receiver. This topology is efficient for asymmetric applications that do not require acknowledgment or return data.

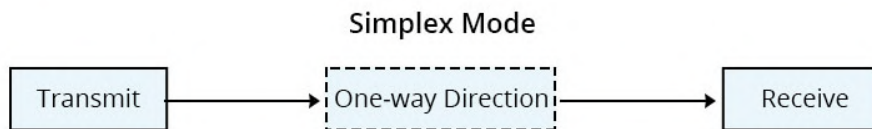


Figure 4.6: Simplex data transmission flow [7].

- **Half-duplex transmission mode:** Resolves the unidirectional limitation of simplex systems by enabling bidirectional data exchange, but not concurrently. As shown in Figure 4.7, both end devices possess transmitting and receiving capabilities. However, because they must time-share the communications medium, data can only flow in one direction at any given instant. While this alternating sequence saves bandwidth, it adds a slight delay whenever the direction of transmission switches.

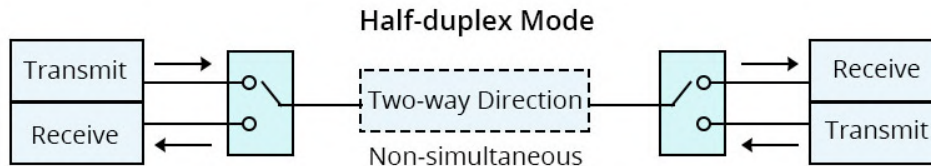


Figure 4.7: Half-duplex data transmission flow [7].

- **Full-duplex transmission mode:** This mode allows simultaneous and bidirectional data transfer. As demonstrated in Figure 4.8, this architecture functions as a composite of two independent simplex channels operating in opposite directions. Because both FPGA nodes are equipped with dedicated transmit and receive pathways, they can process incoming and outgoing data streams concurrently.

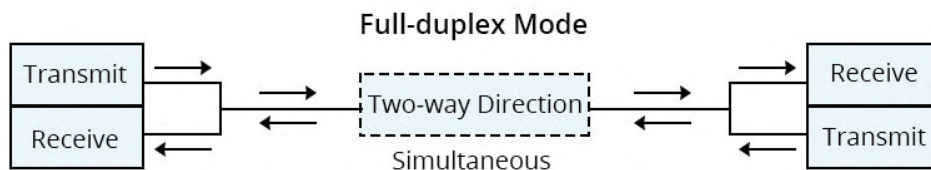


Figure 4.8: Full-duplex data transmission flow [7].

In the implementation, a hybrid deployment strategy is utilized. A simplex configuration is used during the initial setup and calibration phase before transitioning to full-duplex deployment to handle continuous and simultaneous bidirectional encryption and decryption workflows.

4.6 Architectural Constraints and Data Path Optimization

To bridge the theoretical communications channel with the physical SoC hardware, the architecture imposes memory and buffering constraints. Data traverses between the Processing System (PS) and the hardware Programmable Logic (PL) via DDR memory. For real-time execution without triggering network fragmentation or CPU starvation, the data path parameters must be optimized.

4.6.1 DMA Double Buffering Strategy

To decouple the real-time timing of the audio hardware from the asynchronous nature of network traffic, the system employs a double buffering (ping-pong) memory architecture. Two physically distinct memory regions, Buffer A and Buffer B, are allocated in the DDR memory.

The DMA engine streams audio data to Buffer A from the programmable logic, while the ARM Cortex-A9 processor packs incoming network payloads from Buffer B. Upon transmitting, a hardware interrupt signals DMA transfer completion, and the software automatically swaps the buffer pointers.

The primary trade-off in this design is determining the optimal buffer size:

- **Latency and Throughput:** Larger buffers improve throughput and handle network jitter better by reducing CPU interrupts, but they result in higher processing latency. Smaller buffers minimize latency for real-time audio, but risk audio dropouts due to the high frequency of DMA interrupts.
- **Cache Coherence Overhead:** Because the CPU and DMA share physical memory, the CPU must manually invalidate its cache before each transfer. Consequently, frequent buffer swaps increase the CPU time spent clearing the cache rather than processing audio packets.

4.6.2 Ethernet Payload Sizing and IP Fragmentation

To guarantee reliable delivery and minimize protocol processing overhead, the network transmission must be designed to avoid IP fragmentation. The latter occurs when a packet exceeds the physical layer's maximum capacity, forcing the network to split it into smaller segments.

The sizing parameters for standard Gigabit Ethernet establish the maximum audio payload per packet:

- **Maximum Transmission Unit (MTU):** It is defined as 1500 bytes and represents the absolute maximum size of the IP packet (including all headers and payload) that

can be encapsulated into a single physical Ethernet frame.

- **Protocol Overhead:** The IPv4 header consumes 20 bytes, while the TCP and UDP headers add 20 bytes and 8 bytes, respectively.

Thus, the Maximum Segment Size (MSS), which is the largest quantity of application layer data per segment without fragmentation, is calculated as:

$$MSS_{TCP} = 1500 \text{ (MTU)} - 20 \text{ (IPv4)} - 20 \text{ (TCP)} = 1460 \text{ bytes} \quad (4.5)$$

$$MSS_{UDP} = 1500 \text{ (MTU)} - 20 \text{ (IPv4)} - 8 \text{ (UDP)} = 1472 \text{ bytes} \quad (4.6)$$

So, the application layer must chunk the audio stream such that the payload never exceeds 1460/1472 bytes, per network packet.

4.6.3 Mathematical Optimization of Hardware Parameters

The constraints can be formalized into a mathematical model to determine the optimal DMA and FIFO sizing parameters. Let F_s denote the audio sampling frequency, N_c the number of channels ($N_c = 2$ for stereo), and Q the audio bit depth resolution.

Optimal Buffer Size

Let N_s represent the number of audio frames accumulated per buffer swap. The temporal duration of one buffer is $T_{\text{buf}} = N_s / F_s$. To prevent CPU starvation, T_{buf} must be strictly greater than the maximum software overhead required to process the hardware interrupt and invalidate the memory cache, denoted as T_{ISR} :

$$T_{\text{buf}} \gg T_{\text{ISR}} \implies N_s \gg T_{\text{ISR}} \cdot F_s \quad (4.7)$$

Simultaneously, to respect the MSS constraint derived in (4.5) and (4.6), the size of the buffer in bytes (S_{buffer}) must be bounded:

$$S_{\text{buffer}} = N_s \cdot N_c \cdot \left(\frac{W_{\text{DMA}}}{8} \right) \leq \text{MSS} \quad (4.8)$$

Isolating N_s yields the maximum number of samples transmitted per packet without fragmentation:

$$N_s \leq \left\lfloor \frac{\text{MSS} \cdot 8}{N_c \cdot W_{\text{DMA}}} \right\rfloor \quad (4.9)$$

For a 32-bit stereo configuration ($W_{\text{DMA}} = 32$, $N_c = 2$, $\text{MSS} = 1460$), equation (4.9) restricts the buffer capacity to a maximum of $N_s = 182$ samples. The selected buffer size must therefore satisfy the dual inequality:

$$T_{\text{ISR}} \cdot F_s \ll N_s \leq 182 \quad (4.10)$$

AXI-Stream FIFO Depth Sizing

The AXI-Stream FIFO serves as an essential buffer between the continuous data stream of the audio hardware and the burst-oriented transfers of the DMA. The primary factor in determining the required FIFO depth is the dead time (T_{dead}), which represents the system latency incurred when the DMA completes a transfer and waits for the processor to allocate the next memory buffer.

During this transition period, the audio interface continues to generate data at a fixed sample rate (F_s) for each channel. The FIFO must be adequately sized to store all incoming samples during this interval, while also reserving enough capacity to hold at least one data burst (L_{burst}) to maintain bus efficiency. The minimum depth (D_{FIFO}) is calculated as:

$$D_{\text{FIFO}} \geq \lceil T_{\text{dead}} \cdot F_s \cdot N_c \rceil + L_{\text{burst}} \quad (4.11)$$

If the FIFO is configured below this threshold, it will overflow during DMA buffer transitions, which leads to dropping data packets.

The values $L_{\text{burst}} = 32$ and $D_{\text{FIFO}} = 1024$ were chosen as they satisfy the theoretical depth requirement with a safety margin.

Conclusion

This chapter presented the communications architecture that supports the proposed secure audio transmission system. It detailed the theoretical communications requirements, the audio interface, the AXI4-Stream protocol, and the architectures for both the transmitter and receiver. Additionally, the chapter examined the network protocols and transmission modes, as well as the optimization of buffering and data transfer mechanisms needed to ensure real-time operation. Together, these components provide the communications infrastructure in which the hardware-based encryption and decryption modules will be integrated.

FPGA Implementation

This chapter presents the complete Field Programmable Gate Array (FPGA) implementation of a real-time secure audio communications system based on the Subspace-Based Encryption (SBE) algorithm, deployed across two peer-to-peer Zybo Z7-20 boards. The hardware architecture features a strict hardware-software partition: network stack management runs on the ARM Cortex-A9 Processing System (PS), while all cryptographic processing occurs entirely within the Programmable Logic (PL) fabric, with the Direct Memory Access (DMA) serving as a middleware interface for data transfer. Audio samples traverse the pipeline via high-throughput AXI4-Stream interfaces before being exchanged over the 1 Gbit/s Ethernet link. Rather than operating as a rigid emitter-receiver pair, both hardware nodes function as full-duplex transceivers, concurrently executing identical transmit (Tx) and receive (Rx) processes.

The contribution of this chapter is a fully pipelined, full-duplex hardware architecture capable of executing the complete SBE algorithm within the 1302 cycle budget imposed by the 48 kHz audio sampling rate. This system-level achievement is realized through several developments, which structure the chapter as follows:

- The physical deployment and integration of the SSM2603 codec, establishing the hardware baseline for this architecture, are presented in [Section 5.1](#) and [Section 5.2](#).
- A synchronization methodology that maintains absolute cryptographic phase alignment between Tx and Rx processes without requiring dedicated side-channels or additional control logic. The communications framework supporting this protocol is detailed in [Section 5.3](#).
- The encryption architecture is detailed in [Section 5.5](#), where the full arithmetic path is organized in an eight-stage pipeline to satisfy timing closure, and a dual-cascade, CORDIC-based hardware implementation of the non-linear function that operates entirely without dedicated hardware multipliers.
- [Section 5.6](#) presents the decryption architecture, in which recovery is performed via a hardware pseudo-inverse engine and a synchronization mechanism.

5.1 Hardware Platform

The physical implementation of the proposed system is deployed on the Digilent Zybo Z7-20 development board, built around the Xilinx Zynq-7020 SoC. Realizing secure full-duplex audio requires a hardware foundation capable of meeting three central criteria: real-time cryptographic processing, continuous audio acquisition, and high-throughput Ethernet communications. The following subsections detail the SoC architecture and resource mapping to demonstrate how the platform accommodates these constraints.

5.1.1 SoC Architecture Overview

The functional architecture of the Zynq-7000 SoC is illustrated in Figure 5.1.

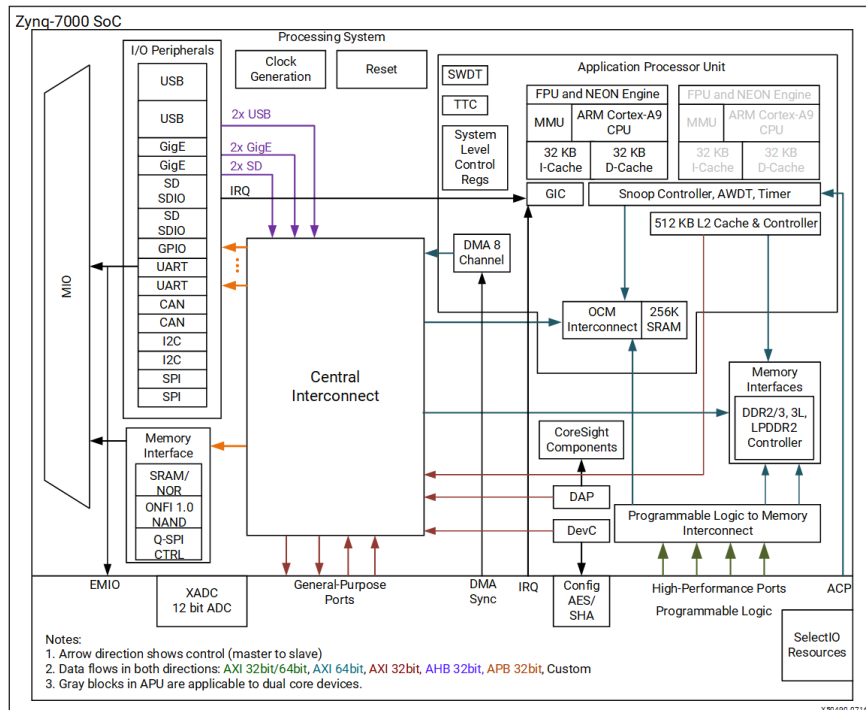


Figure 5.1: Zynq-7000 SoC functional overview.

The platform is structured around two subsystems. Table 5.1 summarizes the key components exhibited in Figure 5.1, and their roles in this work.

5.1.2 Motivation for the Choice of the Zybo z7-20

Implementing a secure audio communications system imposes demands that neither a pure microcontroller nor a standalone FPGA can satisfy efficiently. A microcontroller lacks the highly parallelized fabric required for real-time cryptographic execution, while a bare FPGA lacks the abstraction layers necessary for robust network stacks and complex system orchestration.

The Zynq-7020 resolves this issue through its hybrid architecture. By offloading the computationally intensive SBE pipelines and TRNG cores to the PL fabric, while relying on the PS to orchestrate DMA transfers, manage network stacks, and configure the audio codec via

Table 5.1: Zynq-7020 components and their role in the secure audio system.

Subsystem	Component [58]	Architectural Role in the system
Processing System (PS)	ARM Cortex-A9 (dual-core)	System control, DMA management, network stack
	L1/L2 caches, DDR3 interface	Audio buffer storage, data coherence
	DMA controller (PS-PL)	Audio frame transfer to/from PL fabric
	GbE MAC + Ethernet PHY	Audio streaming over Ethernet
Programmable Logic (PL)	CLBs, LUTs, registers	TRNG, ChaCha20, SBE control logic
	DSP slices	SBE matrix inner-product operations
	Block RAM	Asymmetric FIFOs
	SSM2603 codec interface (I2S)	Audio acquisition and playback
PS-PL Interconnect	AXI HP ports	High-throughput DMA audio data path
	AXI GP ports	Control register access (cipher cores)

high-speed AXI interconnects, the system achieves strict real-time execution without CPU bottlenecks.

Beyond the fundamental capabilities of the Zynq SoC, the development board provides critical integrated peripherals, specifically, the SSM2603 audio codec (with Line-In, Mic-In, and headphone interfaces) and a Gigabit Ethernet port. The availability of these built-in interfaces, combined with sufficient LUT and BRAM resources to accommodate the full system design. Furthermore, possessing two identical boards naturally facilitates a symmetric full-duplex transceiver experimental setup without requiring external bridging hardware.

Exactly these architectural advantages motivated the final selection of the Zybo Z7-20 for this work.

5.2 Audio Codec Architecture and Control Interface

Real-time audio capture and reconstruction are driven by the analog SSM2603, a low-power stereo audio codec.

To ensure optimal performance and avoid processing bottlenecks, the architecture divides the audio operations into two paths: a sequential control managed by the ARM PS and a parallel data streaming implemented in the PL fabric.

5.2.1 System Configuration and I^2C Control Path

Audio codec initialization requires precise configuration of sampling rates, power management states, and analog routing paths before signal acquisition. It is executed as a one-time setup during the system boot by the PS using the Inter-Integrated Circuit (I^2C) protocol [8]. Once these registers are locked, the I^2C peripheral remains dormant to preserve the processing cycles unless a user dynamically requests an input source change (switching from Mic-In to Line-In) at runtime.

The hardware-software boundary of the architecture is achieved by integrating an I^2C Intellectual Property (IP) core in the PL, which is connected to the PS via an AXI Interconnect peripheral as illustrated in Figure 5.2. Meanwhile, the independent top-level Register-Transfer Level (RTL) module bypasses the processor bus to route high-speed audio signals (MCLK, BCLK, and LRCLK as explained in Section 4.2.1) directly to the codec interface, while also

receiving samples from it and outputting the processed ones.

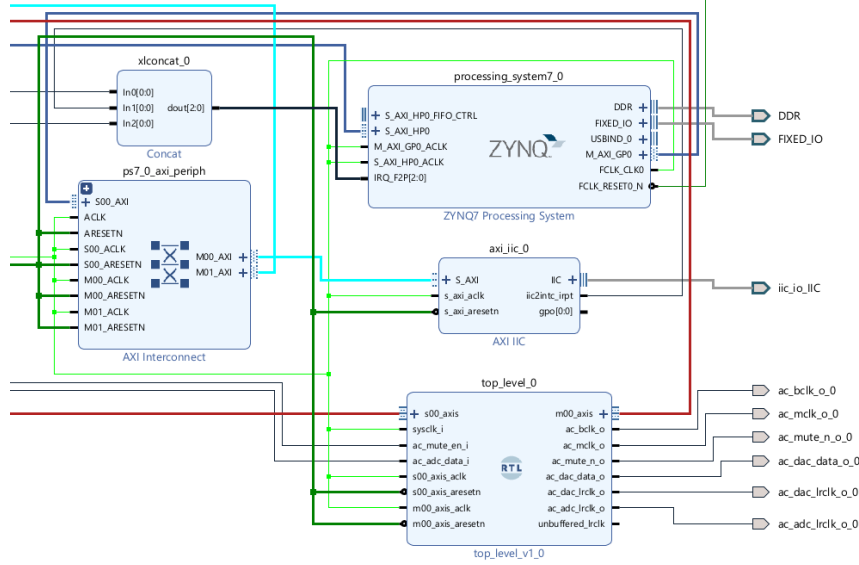


Figure 5.2: IP block of the system architecture: the PS's I^2C control and the PL's I^2S data streaming.

In contrast, implementing this initialization in the PL would require a custom I^2C master and a complex FSM. Assigning these tasks to the ARM processor preserves LUTs and flip-flops while reserving the PL fabric exclusively to the cryptographic pipelines.

5.2.2 Data Streaming, Sampling, and Digital Filtering

The system enforces a sampling frequency of $f_s = 48$ kHz established by writing a bit vector to the register (R8). This configuration adjusts the Base Over-Sampling Rate (BOSR)¹ and Sample Rate (SR)² dividers within the codec, allowing the internal clocks to remain synchronized with the external hardware Master Clock.

Similarly, analog circuits introduce a slow voltage drift known as a DC offset, which shifts the audio signal away from its true zero baseline. Although this shift does not affect normal listening, it can cause fixed-point overflow in the FPGA's arithmetic pipelines and corrupt encryption and decryption operations.

To correct this, the digital built-in High-Pass Filter (HPF)³, integrated into the codec was enabled by clearing the corresponding bit in the Digital Audio Path register (R5). This filter is applied immediately after the Analog-to-Digital Converter (ADC). It features a sharp cutoff frequency of -3 dB of 3.7 Hz [8] that decreases DC drift without modifying audible frequencies.

¹BOSR bit (Register R8, Bit D1) is the oversampling multiplier relative to the audio sampling rate, selecting $256f_s$ clocking profile in this implementation (the register's bits are mentioned in the Appendix).

²The 4 bit SR (SR[3:0], Register R8) configures internal hardware dividers to step down the MCLK into the precise clocks required for the 48 kHz I^2S stream.

³HPF is a hardware ADC filter, specifications: 3.7 Hz (-3 dB), 10.4 Hz (-0.5 dB), and 21.6 Hz (-0.1 dB) [8].

5.2.3 Analog Routing: Mic-In and Line-In Configurations

Depending on the application, the system can route two hardware input pipelines via the Analog Audio Path register (R4). As illustrated in the codec's internal block diagram in Figure 5.3, these inputs contain different electrical characteristics and internal routing topologies:

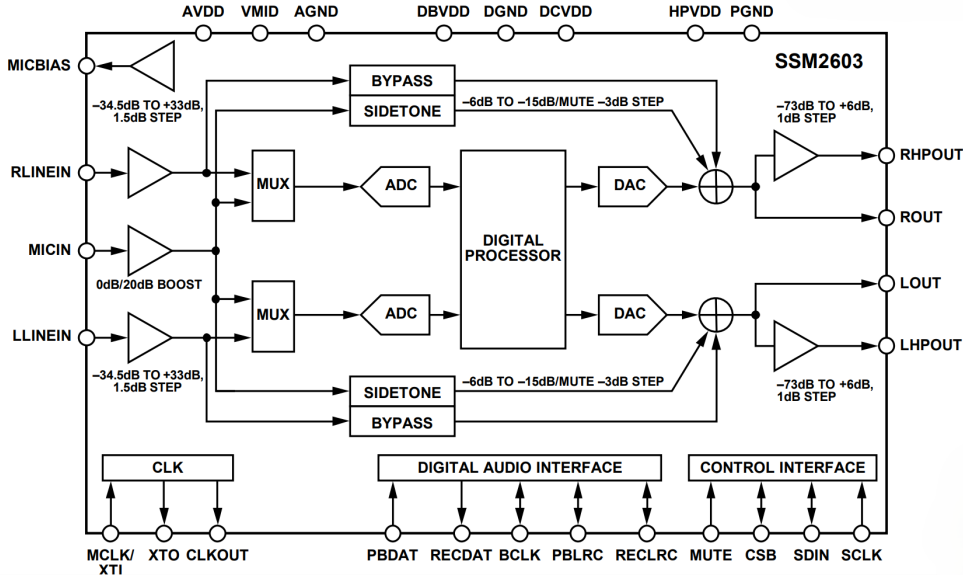


Figure 5.3: SSM2603 internal analog routing architecture [8].

- **Microphone Input:** Microphones output highly attenuated analog signals, typically in the millivolt range. When this input is selected, the codec routes the mono signal through an internal preamplifier. As highlighted in Figure 5.3, this path features a switchable +20 dB boost loop. Without this stage, the weak microphone signal would not utilize the full dynamic range of the ADC, leading to low SNR and quantization noise.
- **Line Input:** It is designed to accept standardized, higher voltage signals, such as smartphones or PC audio outputs. When selected, this path captures the stereo audio signal and bypasses the boost loop, as the latter may introduce voltage saturation and clipping at the ADC input. Instead, the fine-tuning of the stereo levels is handled by the independent Left and Right ADC volume registers (R0 and R1), which are initialized to unity gain of 0 dB.

5.2.4 Initialization Sequence and Register Mapping

The PS initializes the SSM2603 via the I^2C bus and transmits 9-bit payloads to configure the codec's power domains, analog routing, and digital framing. This sequence is summarized in Table 5.2.

Table 5.2: SSM2603 I^2C register configuration and initialization sequence.

Register	Purpose	Data	Functional Explanation
R15	Software Reset	0b000000000	Executes a software reset and clears all registers to default hardware states
R0	Left ADC Input Volume	0b000010111	Unmutes left ADC and calibrates it to a 0 dB (unity gain) baseline
R1	Right ADC Input Volume	0b000010111	Unmutes right ADC and calibrates it to a 0 dB (unity gain) baseline
R2	Left DAC Volume	0b101111001	Sets left DAC volume to 0 dB and enables simultaneous load to the right channel
R3	Right DAC Volume	0b101111001	Sets right DAC volume to 0 dB and enables simultaneous load to the left channel
R4	Analog Audio Path	0b000010010	Configures the analog path for Line-In, bypassing the microphone boost loop
R5	Digital Audio Path	0b000000000	Enables the DC High Pass Filter and unmutes the DAC
R6	Power Management	0b000110000	Powers on the ADC, DAC and the analog paths
R7	Digital Audio I/F	0b000001010	Configures the audio interface for 24-bit word lengths and standard I^2S formatting
R8	Sampling Rate	0b000000000	Sets Line-In mode, selects a $256f_s$ BOSR, and locks clock dividers for $f_s = 48$ kHz
R9	Active Control	0b000000001	Sets the Active bit to 1, engaging the digital core for live audio streaming

5.3 Communications Channel Implementation

Following the hardware–software partitioning introduced previously, this section details the data management mechanisms and network protocols that form the communications channel. Within the system, a Direct Memory Access (DMA) controller functions as the interface between subsystems, enabling efficient data transfer while reducing latency, jitter, and CPU load. The PS handles Ethernet connectivity and implements the network infrastructure required for this project.

5.3.1 Gigabit Ethernet Controller

The Gigabit Ethernet Controller (GEM) implements a 10/100/1000 Mb/s Ethernet MAC ⁴ that complies with the IEEE 802.3-2008 standard and operates in both half and full-duplex modes across all supported speeds. To minimize pin count, external access via the Multiplexed I/O (MIO)⁵ utilizes a Reduced Gigabit Media Independent Interface (RGMII)⁶. Access to the PL is routed through the Extended MIO (EMIO)⁷, which provides a standard Gigabit Media Independent Interface (GMII)⁸ interface [9]. The Figure below represents the block diagram of the GEM.

The configuration registers depicted in Figure 5.4 are used to manage MAC features, select operational modes, and monitor network management statistics.

The Figure 5.4 illustrates the overall architecture of the MAC controller. The specific functional blocks shown in the diagram provide the following key features:

- **IEEE 802.3-2008 compliant** (10/100/1000 Mb/s) with 802.1Q Virtual Local Area Network (VLAN) and priority tagging support.
- **Full and half duplex operation** including hardware flow control and pause frame generation/recognition.

⁴Ethernet MAC is the data-link layer component responsible for frame formatting, addressing, error checking, and medium access control.

⁵MIO provides direct access from the PS to the device pins.

⁶RGMII is a pin-efficient version of GMII.

⁷EMIO extends PS peripheral signals into the PL.

⁸GMII is the standard interface between an Ethernet MAC and a Gigabit Ethernet PHY.

- **Flexible interfaces:** RGMII (via MIO), GMII/MII (to PL), and Management Data Input/Output (MDIO)⁹ for physical layer management.
- **Advanced DMA:** 32-bit Advanced High-performance Bus (AHB) DMA master with scatter-gather¹⁰ capability and a 32-bit AXI High-Performance Port (APB) bus for control register access.
- **Hardware offloads:** Automatic TX pad/CRC generation¹¹, automatic discard of errored RX frames, and IPv4/IPv6 TCP/UDP checksum offloading.
- **Comprehensive interrupts** for RX/TX completion, errors, and wake-up events.

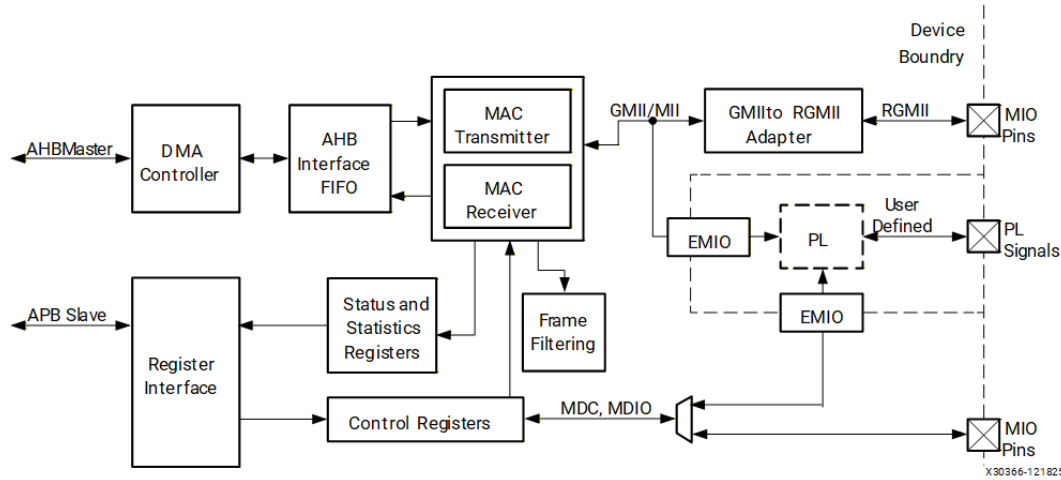


Figure 5.4: Block diagram of the Gigabit Ethernet controller [9].

Physical Layer Configuration

Configuring the Gigabit Ethernet physical layer (PHY) on the Zynq Processing System requires both hardware parameterization and software initialization. Hardware settings are defined in Vivado, AMD's design environment for adaptive SoCs and FPGAs, through the MDIO interface. In contrast, software driver configuration and optimization are performed within the Vitis¹² development environment.

- Hardware Configuration

- o **Ethernet Enablement:** Within the Zynq PS configuration interface, the ENET_0 peripheral is activated and the corresponding I/O bank voltage is configured to High-Speed Transceiver Logic (HSTL 1.8 V) for high-speed interfaces.
- o **MDIO Interface:** This interface is assigned to MIO pins (pins 52 and 53) rather than being routed through the FPGA fabric via EMIO. This sets a direct hardware path to the PHY for stable register access.
- o **Clock Synchronization:** The peripheral clock driving the Ethernet MAC is set

⁹MDIO is a serial management interface for configuring and monitoring Ethernet PHY devices from an Ethernet MAC.

¹⁰Scatter-gather is a DMA transfer mode.

¹¹Automatic TX pad/CRC generation allows the Ethernet MAC to automatically insert frame padding and generate the CRC-based Frame Check Sequence (FCS) during transmission.

¹²Vitis is a design tool used for FPGA software configurations [60].

to 125 MHz, matching the fundamental baseline frequency required for operational Gigabit line speeds.

- Software and Driver Initialization

- Following the hardware generation, the network stack is deployed using the Lightweight IP (lwIP) library template. To preserve connection stability when interfacing with the PHY, adjustments are applied within the Board Support Package (BSP).

5.3.2 Implementation of the DMA-Based Audio Data Transfer

A critical challenge in implementing real-time encryption was the synchronization between the two boards. Since the cipher requires a unique initial state at the receiver for accurate decryption, as detailed in Section 3.3, the 96-bit nonce must be transmitted over the network alongside the audio data for every batch.

AXI DMA Configuration

The data transfer backbone relies on the Xilinx AXI DMA controller configured in simple mode, with both the Memory-to-Stream (MM2S) and Stream-to-Memory (S2MM) channels enabled to support the full-duplex architecture. The stream interface is fixed to a 32-bit data width to match the audio payload length. The maximum burst length is set to 64 transfers, enabling large block transfers that reduce bus overhead and sustain a continuous data flow. The detailed configuration parameters of the AXI DMA core are shown in Figure 5.5.

The image shows a configuration window for the AXI DMA component. The component name is 'axi_dma_0'. Several options are available for enabling or disabling features like asynchronous clocks, scatter gather engine, micro DMA, multi-channel support, and control/status stream. The buffer length register width is set to 11 bits, and the address width is 32 bits. Both the Read and Write channels are enabled. For the Read Channel, the number of channels is 1, memory map data width is 64, stream data width is 32, and max burst size is 64. For the Write Channel, the number of channels is 1, memory map data width is set to 'AUTO' (showing 32), stream data width is set to 'MANUAL' (showing 32), and max burst size is 64. There are also checkboxes for 'Allow Unaligned Transfers' and 'Use Rlength In Status Stream'.

Figure 5.5: AXI DMA configuration parameters used in the system.

Structuring the DMA Payload

The DMA transfer size is selected according to the bounds derived in Equation 4.10. To maintain compatibility with both TCP and UDP transport layers, the system uses the lowest admissible bound, corresponding to the maximum segment size defined in Section 4.5.

- Data Payload:** 64 bits
- Encryption Flag:** 1 bit

- **Nonce:** 96 bits

A fresh nonce is generated by the hardware TRNG for every transmitted batch, as described in Section 3.5. The Least Significant Bit (LSB) of the 64-bit audio payload serves as an encryption flag, because altering the LSB of a high-resolution audio sample has an unnoticeable effect on audio quality. This optimization eliminates the need to allocate an additional 32-bit word for a single control bit. The final transmitted data contains 160 bits, as illustrated in Figure 5.6.

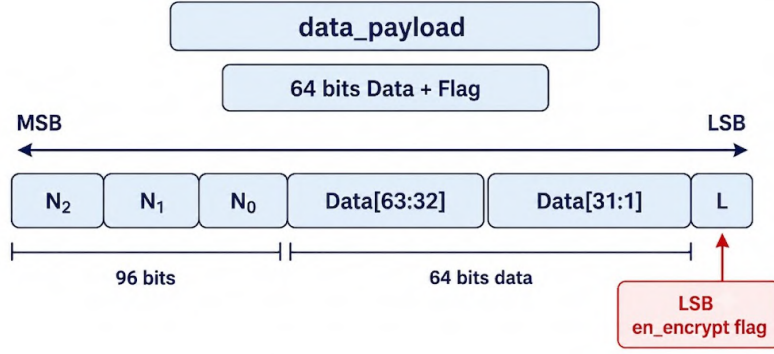


Figure 5.6: Data structure payload format.

Memory Layout and Payload Distribution

The transmission subsystem receives encrypted payloads from the PL through an AXI4-Stream interface. Each incoming payload is represented as a 160-bit word composed of three 32-bit nonce words and two 32-bit encrypted-data words. These payloads are first stored in an internal FIFO before being serialized for DMA transmission.

For compatibility with the AXI DMA configuration, the FIFO converts each 160-bit payload into five consecutive 32-bit AXI-Stream words. Consequently, every payload occupies five sequential DMA transfer words, corresponding to the memory layout illustrated in Figure 5.7. By design, our ChaCha20 implementation requires a new 96-bit nonce. Therefore, the first three 32-bit words of the payload are dedicated entirely to providing these three new nonce values (see Section 3.3). The remaining two words contain the encrypted audio data and the associated encryption flag.

During operation, the DMA continuously receives 32-bit words from the FIFO and stores them in the DDR buffer. A complete transmission frame consists of 365 consecutive 32-bit words, corresponding to a total payload size of 1460 bytes. The end of the frame is indicated by the assertion of the **TLAST** signal on the final transfer word.

Figure 5.7 illustrates the logical organization of a single encrypted payload, showing the placement of the nonce values, encrypted audio data, and encryption flag within the serialized 32-bit word stream.

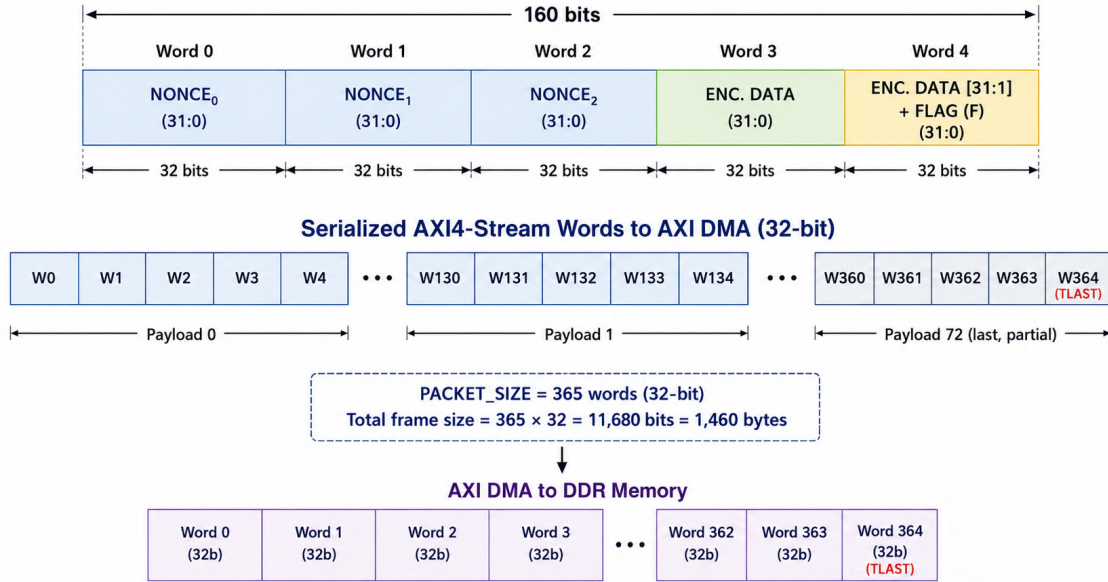


Figure 5.7: Organization of a 160-bit encrypted payload serialized into five 32-bit DMA transfer words.

- Transmission

During transmission, the MM2S DMA channel reads the active DDR ping-pong buffer and forwards a frame consisting of 365 consecutive 32-bit words to the network interface. The processor encapsulates the DMA frame within the transport-layer packet before initiating the socket transmission procedure. Since the resulting payload size is 1460 bytes, which remains below the Ethernet MTU, the complete frame is transmitted as a single network packet without IP-layer fragmentation.

- Reception

During reception, the S2MM DMA channel stores the incoming network payload into the active DDR ping-pong buffer. The processor subsequently parses the received data in groups of five 32-bit words, extracting NONCE₀, NONCE₁, NONCE₂, and the encrypted data.

5.4 Fixed-Point Representation Overview

Digital systems represent real-valued data using floating-point or fixed-point arithmetic. The IEEE-754 floating-point representation provides a wide dynamic range by storing a mantissa and an exponent, but requires complex hardware and consumes more FPGA resources. In contrast, fixed-point representation uses a fixed binary point position, resulting in simpler arithmetic units and reduced hardware utilization. For embedded systems based on an FPGA, fixed-point arithmetic is often preferred because it significantly reduces the usage of LUTs, flip-flops, and DSP resources while meeting the numerical accuracy requirements of the target application [10].

The proposed architecture uses a signed fixed-point representation using the standard $Q_{m,n}$ format, where m and n denote the numbers of integer and fractional bits, respectively. The binary point is fixed between the integer and fractional parts, and the signed values are represented using two's complement notation. The numerical range of a signed $Q_{m,n}$ number

is given by:

$$-2^{m-1} \leq x \leq 2^{m-1} - 2^{-n} \quad (5.1)$$

As illustrated in Figure 5.8, the Most Significant Bit (MSB) represents the sign, the next $m - 1$ bits correspond to the integer part, and the remaining n bits represent the fractional part.

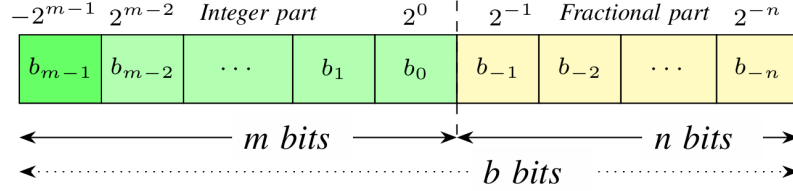


Figure 5.8: Signed fixed-point representation in $Q_{m,n}$ format [10].

Let $x \in Q_{m_x, n_x}$ and $y \in Q_{m_y, n_y}$ be two fixed-point variables, and z is the calculated result.

Addition and Subtraction

Fixed-point addition and subtraction are identical to normal integer operations. To handle a potential carry bit and prevent overflow, an extra bit is allocated to the output's integer part [10]:

$$m_z = \max(m_x, m_y) + 1, \quad n_z = \max(n_x, n_y) \quad (5.2)$$

For example, adding $x = 1.25 \in Q_{2,2}$ and $y = 0.5 \in Q_{1,1}$, y is left-shifted to match the fractional size of x , and then the vertical integer addition is executed:

$$\begin{array}{rcl} 01.01_2 & (1.25_{10}) & [Q_{2,2}] \\ + 00.10_2 & (0.50_{10}) & [Q_{2,2}] \\ \hline 001.11_2 & (1.75_{10}) & [Q_{3,2}] \end{array}$$

Multiplication and Division

Multiplication is executed as a standard signed integer multiplication, and the division involves iterative algorithms, such as restoring division, that will be used in the SBE implementation algorithm. The result's integer and fractional widths of both operations increase according to [10]:

$$m_z = m_x + m_y, \quad n_z = n_x + n_y \quad (5.3)$$

For instance, the product of $x = -1.5 \in Q_{2,1}$ and $y = -0.25 \in Q_{1,2}$ is given by:

$$\begin{array}{rcl} 10.1_2 & (-1.5_{10}) & [Q_{2,1}] \\ \times 01.11_2 & (-0.25_{10}) & [Q_{1,2}] \\ \hline 000.011_2 & (0.375_{10}) & [Q_{3,3}] \end{array}$$

To avoid quantization noise and precision loss across operations, a larger fractional bit-width n must be maintained. However, truncating or rounding these bits down to fit a standard bus

width introduces quantization noise, which can be reduced by selecting optimal bit-widths for each intermediate variable rather than using a uniform configuration across the entire pipeline.

5.5 Encryption Architecture

The encryption hardware translates the SBE algorithm into a real-time digital circuit. Rather than processing audio in blocks, the system treats each sample as an independent frame. The following part describes how these constraints are satisfied.

5.5.1 AXI4-Stream Interface and Audio Framing

Audio samples are transmitted between processing modules via the AXI4-Stream protocol. Each 32-bit of pre-encrypted data carries a 24-bit signed two's complement PCM sample in its lower three bytes, with the upper byte padded to zero:

$$\mathbf{tdata} = 0x00 \& p(t), \quad (5.4)$$

where $\&$ denotes bit concatenation. The audio sign bit, therefore, resides at bit position 23. The encrypted output is a 64-bit data containing two independent 32-bit signed ciphertext channels $x_1(t)$ and $x_2(t)$, packed in the lower and upper halves, respectively.

Table 5.3 summarizes the data packing convention used throughout the pipeline.

Table 5.3: AXI4-Stream TDATA framing convention.

Bus width	Bits [63:32]	Bits [31:0]	Meaning
32 bit (pre-encryption)	—	$p(t)$	plaintext PCM sample
64 bit (ciphertext)	$x_2(t) \neq 0$	$x_1(t)$	Encrypted output
64 bit (bypass)	0x00000000	$p(t)$	Encryption disabled
64 bit (decrypted)	0x00000000	$\hat{p}(t)$	Recovered plaintext

5.5.2 Top-Level Encryption System

The system clock is $f_{\text{clk}} = 125$ MHz, establishing a clock period of $T_{\text{clk}} = 8$ ns. Audio is captured and reproduced at $f_s = 48$ kHz for each channel. Because the Left Right clock (LRCLK) toggles on both the rising and falling edges to identify left and right channel samples, the effective sample delivery rate is $2f_s = 96$ kHz. All computation across every stage must therefore complete within:

$$N_{\text{budget}} = \frac{f_{\text{clk}}}{2f_s} = \frac{125 \text{ MHz}}{96 \text{ kHz}} = 1302 \quad \text{clock cycles per audio sample} \quad (5.5)$$

For a segment length $M = 1$ in equation (2.2), each audio sample is processed independently as a single frame. The sample $p(t)$ is encrypted into a two-component ciphertext vector $\mathbf{x}(t) = [x_1(t), x_2(t)]^T$ according to:

$$\begin{cases} x_1(t) = a_1(t) p(t) + \beta c_1(t) k(t) \tilde{g}(p(t)) \\ x_2(t) = a_2(t) p(t) + \beta c_2(t) k(t) \tilde{g}(p(t)) \end{cases} \quad (5.6)$$

where each parameter is refreshed on every sample clock edge to maximize cryptographic entropy:

- $[a_1(t), a_2(t)]^T$ are 16-bit signal space coefficients, making $A(t)$ a 32-bit vector, sourced from the ChaCha20.
- $[c_1(t), c_2(t)]^T$ are the orthogonal projection coefficients derive from $P_{A(t)}^\perp B(t)$, recomputed whenever the subspace basis changes.
- $k(t)$ is a 16-bit truly random scalar sourced from the RO-TRNG.
- $\tilde{g}(p(t))$ is the biased non-linear function.
- β is a fixed noise scaling constant.

Equation (5.6) requires ten multiplications and two additions to be performed within a single audio sample period (10.41 μ s). These operations are executed on the DSP slices to ensure deterministic latency between the key generation and encryption stages.

The top-level architecture, illustrated in Figure 5.9, routes the incoming AXI-Stream audio through a priority demultiplexer that selects between two execution paths:

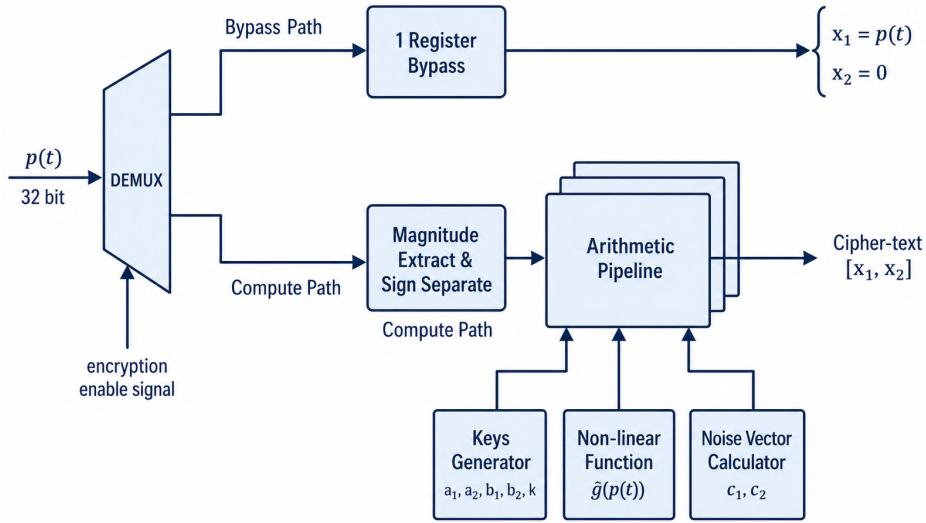


Figure 5.9: Top-level architecture of the SBE encryption system.

- **Compute Path:** Activated when the external encryption enable signal is asserted simultaneously with the internal key valid flag. The sample flows through the non-linear transformation and the eight-stage arithmetic pipeline, producing a 64-bit encrypted message.
- **Bypass Path:** Engaged whenever encryption is disabled, forwarding the raw sample with zero additional latency to preserve uninterrupted signal flow.

5.5.3 The Non-linear Function: CORDIC-Based Realization

The non-linear function couples the true-random key $k(t)$ to the audio sample that prevents the differential attack, described in Section 2.1, from isolating the key matrix through a simple subtraction. The optimal function proposed in [5] is:

$$g(v) = \frac{v}{\sqrt{1+v^2}}, \quad g: \mathbb{R} \rightarrow (-1, +1) \quad (5.7)$$

Direct hardware evaluation of (5.7) requires a squarer, adder, square-root unit, and divider, which collectively consume between four and six DSP slices and introduce variable latency. On the other hand, a 24-bit input lookup table requires approximately $2^{24} \times 24 = 50$ MB, which would consume a large amount of FPGA memory and logic resources. A Taylor expansion demands several multiplications and additions per evaluation to ensure a reliable result.

This issue is resolved by reformulating (5.7) using trigonometric identities suitable for the Coordinate Rotation Digital Computer (CORDIC) evaluation. Starting from the definition of $g(v)$ and using $v = \text{sign}(v) |v|$, the numerator can be rewritten as:

$$g(v) = \text{sign}(v) \frac{|v|}{\sqrt{v^2 + 1}} \quad (5.8)$$

Let $\Phi = \arctan\left(\frac{1}{|v|}\right)$. Applying the identity $\cos(\arctan(u)) = \frac{1}{\sqrt{1+u^2}}$, it follows that:

$$\cos \Phi = \cos\left(\arctan \frac{1}{|v|}\right) = \frac{1}{\sqrt{1 + \frac{1}{v^2}}} = \frac{|v|}{\sqrt{v^2 + 1}} \quad (5.9)$$

Substituting (5.9) into (5.8):

$$g(v) = \text{sign}(v) \cos\left(\arctan \frac{1}{|v|}\right) \quad (5.10)$$

This identity decomposes $g(v)$ into two sequential operations: computing $\Phi = \arctan(1/|v|)$ and evaluating $\cos \Phi$, followed by a sign correction. Both trigonometric operations can be implemented using CORDIC cores operating in different modes, which eliminates the need for hardware multipliers.

CORDIC Algorithm Principle

The CORDIC algorithm computes vector rotations through a sequence of N pseudo-rotations by angles $\alpha_i = \arctan(2^{-i})$, whose tangents are exact inverse powers of two. The three variable update equations at each iteration are [61]:

$$\begin{cases} x_{i+1} = x_i - d_i \cdot 2^{-i} \cdot y_i \\ y_{i+1} = y_i + d_i \cdot 2^{-i} \cdot x_i \\ z_{i+1} = z_i - d_i \cdot \arctan(2^{-i}) \end{cases} \quad (5.11)$$

Because $\tan(\alpha_i) = 2^{-i}$, every multiplication reduces to a right arithmetic shift by i positions. The rotation direction is determined by a negative feedback: $d_i = +1$ if $z_i > 0$ (rotation mode) or $y_i < 0$ (vectoring mode) and $d_i = -1$ otherwise, driving $z_N \rightarrow 0$ or $y_N \rightarrow 0$, respectively.

Table 5.4 lists the first ten micro-rotation angles. The rapid decrease in α_i halves the angular resolution in each iteration, so 28 iterations achieve a precision of $\arctan(2^{-27}) \approx 4.3 \times 10^{-9}$ radians. This is more than adequate for 24-bit audio.

Table 5.4: First ten CORDIC micro-rotation angles $\alpha_i = \arctan(2^{-i})$.

Iteration i	0	1	2	3	4	5	6	7	8	9
2^{-i}	1.00000	0.50000	0.25000	0.12500	0.06250	0.03125	0.01562	0.00781	0.00390	0.00195
α_i	45.000°	26.565°	14.036°	7.125°	3.576°	1.790°	0.895°	0.448°	0.224°	0.112°

Omitting the term $\cos(\alpha_i)$ from each pseudo-rotation stretches the vector by $1/\cos(\alpha_i)$, as illustrated in the left panel of Figure 5.10. After N iterations, the output vector is stretched by the product of all such factors, introducing a cumulative gain whose product converges to a constant independent of the target angle [61]:

$$K = \prod_{i=0}^{N-1} \cos(\arctan(2^{-i})) \approx 0.6073, \quad N \geq 12 \quad (5.12)$$

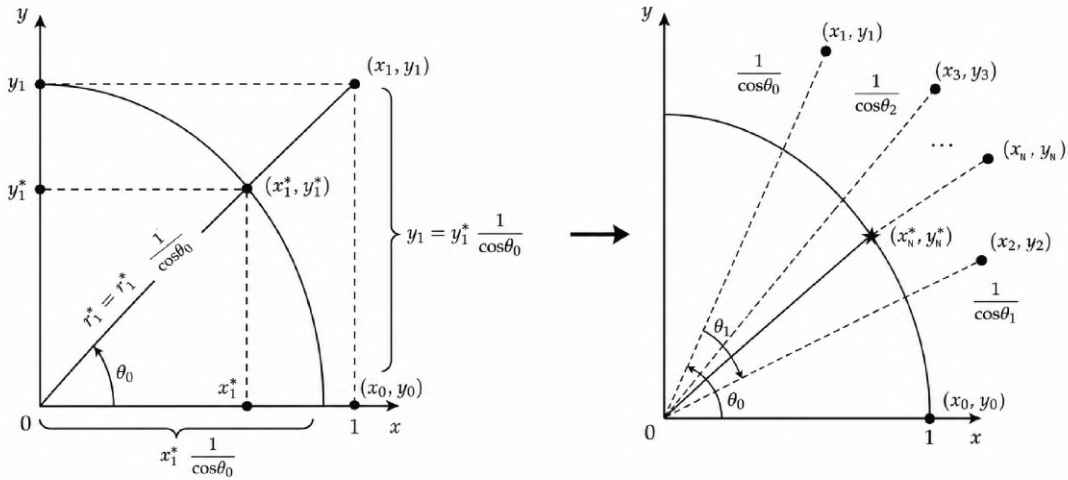


Figure 5.10: CORDIC scaling factor and magnitude convergence.

The right panel of Figure 5.10 shows how this accumulated stretching pushes intermediate points outside the unit circle.

Thus, the final result is given by:

$$\begin{cases} x_{final} = K \cdot x_N \\ y_{final} = K \cdot y_N \end{cases} \quad (5.13)$$

Rotation mode and vectoring mode

The CORDIC algorithm operates in two main modes [61], depending on the quantity minimized by the z register:

- **Rotation mode:** The register is initialized to a desired angle $z_0 = \theta_N$ and the feedback drives $z_N \rightarrow 0$. The vector rotates by the target angle and converges to:

$$x_N \rightarrow \cos \theta_N, \quad y_N \rightarrow \sin \theta_N \quad (5.14)$$

This mode evaluates sine and cosine of a known phase, corresponding to the Xilinx *Sin* and *Cos* functional configuration.

- **Vectoring mode:** The feedback instead drives $y_N \rightarrow 0$, accumulating the rotation needed to align the input vector with the x -axis. The phase register converges to:

$$z_N \rightarrow \arctan\left(\frac{y_0}{x_0}\right) \quad (5.15)$$

This mode computes arc-tangent, using the Xilinx *Translate* functional configuration.

Two CORDIC Cascade

A single CORDIC instance cannot simultaneously compute arctan and evaluate cos, because each operation requires a different feedback target and initialization. Two Xilinx CORDIC IP cores were implemented in cascade, each configured for a signed 24-bit fraction format and 28 pipeline stages per core, as shown in Figure 5.11.

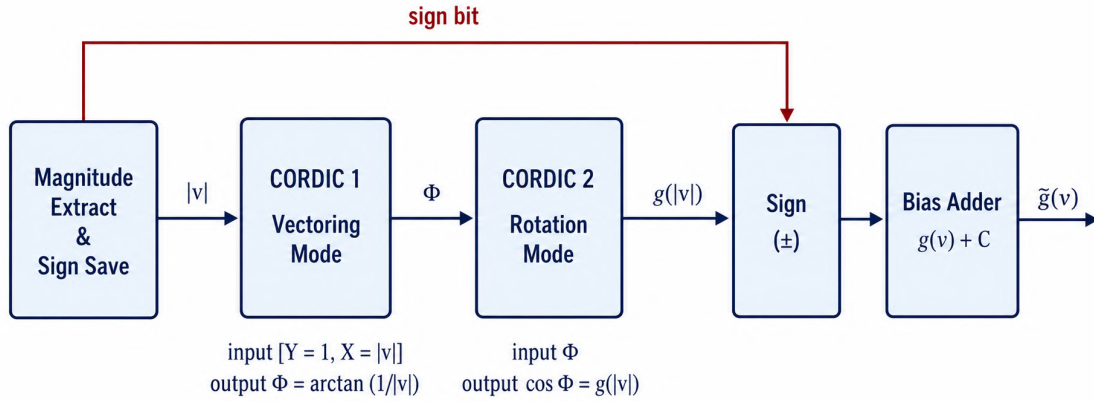


Figure 5.11: Two CORDIC cascade chain.

The overall signal flow is as follows:

1. **Magnitude extraction (1 cycle):** The sign bit of the 24-bit input v is captured in a 1 bit register. The two's complement absolute value $|v|$ is computed with an overflow guard.
2. **CORDIC 1 in vectoring mode (28 cycles):** The input vector is set to $(X, Y) = (|v|, 1)$. Over 28 iterations, the core drives $y_N \rightarrow 0$, accumulating Φ in the 24-bit phase register.
3. **CORDIC 2 in rotation mode (28 cycles):** The 24-bit Φ feeds the phase input. The core rotates the pre-scaled unit vector by Φ and produces $g(|v|)$ on its output.
4. **Sign recovery (1 cycle):** Dedicated 1-bit register carries the saved sign bit. Once the CORDIC 2 output is valid, the sign bit is applied to recover the full signed result.

The total pipeline latency of the non-linear function module is therefore 58 clock cycles.

Bias Augmentation

The base function satisfies $g(0) = 0$, which results in a deterministic ciphertext value when the plaintext sample is zero. While this does not enable signal recovery, it leaks voice activity information by allowing an observer to identify silent intervals. To ensure continuous ciphertext activity, a constant offset is added to the CORDIC output before the encryption stage.

$$\tilde{g}(v) = g(v) + C_{bias}, \quad C_{bias} = 0.1 \quad (5.16)$$

5.5.4 Orthogonal Projection and Coefficient Generation

Within the encryption framework, a noise injection vector $B = [b_1, b_2]^T$ must be projected into the orthogonal complement of the information vector $A = [a_1, a_2]^T$. To guarantee perfect cancellation during the receiver's decryption phase, the resulting coefficient vector $C = [c_1, c_2]^T$ must maintain strict geometric orthogonality with A , which implies:

$$A^T C = a_1 c_1 + a_2 c_2 = 0. \quad (5.17)$$

Direct Matrix Approach and Quantization Leakage

Standard mathematical implementations derive C via the 2×2 orthogonal projection matrix:

$$C = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = P_A^\perp B = \left(I_2 - \frac{A A^T}{A^T A} \right) B = \begin{bmatrix} p_{11} b_1 + p_{12} b_2 \\ p_{21} b_1 + p_{22} b_2 \end{bmatrix} = \begin{bmatrix} \frac{a_2^2}{a_1^2 + a_2^2} & -\frac{a_1 a_2}{a_1^2 + a_2^2} \\ -\frac{a_1 a_2}{a_1^2 + a_2^2} & \frac{a_1^2}{a_1^2 + a_2^2} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (5.18)$$

In a fixed-point hardware environment, computing the individual matrix elements of $P_A^\perp$ requires independent integer divisions, and each hardware division introduces an error ϵ_{ij} . The computed matrix elements become $\hat{p}_{ij} = p_{ij} + \epsilon_{ij}$. This leads to the following dynamic coefficients:

$$\begin{cases} \hat{c}_1 = \hat{p}_{11} b_1 + \hat{p}_{12} b_2 \\ \hat{c}_2 = \hat{p}_{21} b_1 + \hat{p}_{22} b_2 \end{cases} \quad (5.19)$$

Substituting these hardware approximated coefficients into (5.17) reveals how leakage is introduced:

$$a_1 \hat{c}_1 + a_2 \hat{c}_2 = b_1(a_1 \epsilon_{11} + a_2 \epsilon_{12}) + b_2(a_1 \epsilon_{21} + a_2 \epsilon_{22}) = \Delta_{\text{error}} \neq 0 \quad (5.20)$$

Because ϵ_{ij} originates from independent division remainders, they are fundamentally uncorrelated. As a result, the residual Δ_{error} remains non-zero. This violation prevents perfect decryption, and the incomplete cancellation introduces a background noise floor that degrades the reconstructed audio signal.

Algebraic Reformulation for Zero Leakage

To guarantee absolute orthogonality, the matrix representation can be bypassed entirely. Any vector orthogonal to A is strictly proportional to $A^\perp = [a_2, -a_1]^T$. By expanding (5.18), a single scalar multiplier M can be isolated:

$$M = \frac{a_2 b_1 - a_1 b_2}{a_1^2 + a_2^2} \quad (5.21)$$

The projection coefficients are then directly assigned as scaled orthogonal pairs:

$$c_1 = a_2 \cdot M, \quad c_2 = -a_1 \cdot M \quad (5.22)$$

If the hardware division produces an error $\hat{M} = M + \epsilon_M$, the orthogonality constraint (5.17) remains mathematically unbroken:

$$a_1(a_2 \hat{M}) + a_2(-a_1 \hat{M}) = \hat{M}(a_1 a_2 - a_1 a_2) = 0 \quad (5.23)$$

By linking c_1 and c_2 through a shared quantization state ($\hat{M}$), the residual is forced to exactly 0. The latter protects the decryption stream against fixed-point errors, eliminating the associated chaotic background noise.

Optimized FPGA Architecture

The algebraic reformulation optimizes the RTL architecture and reduces DSP resource utilization. The architecture employs an iterative Finite State Machine (FSM) orchestrating one restoring division, four DSP multiplications, and three adders.

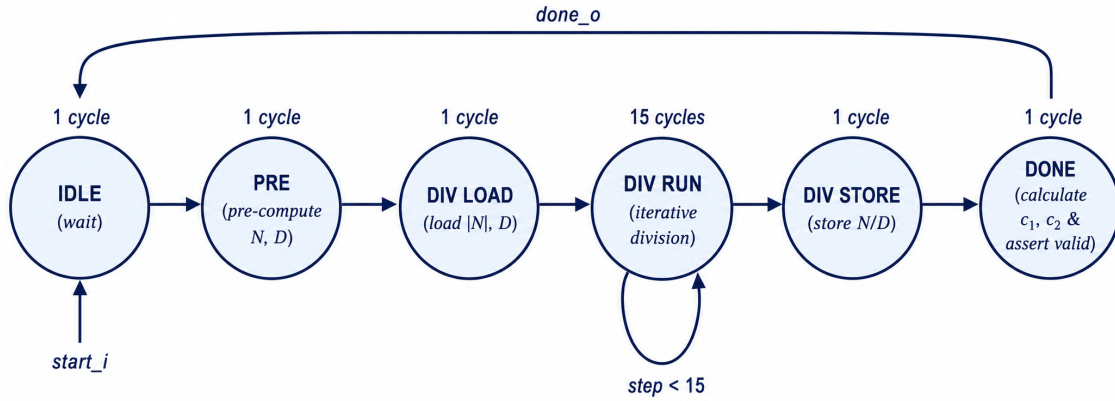


Figure 5.12: FSM of the orthogonal projection coefficient calculator.

The proposed architecture calculates (5.21) and (5.22) using an optimized 6-state FSM operating on 16-bit Q1.15 signed inputs as shown in Figure 5.12. It can be summarized as:

- **Pre-computation (1 cycle):** The numerator $N = a_2 b_1 - a_1 b_2$ and denominator $D = a_1^2 + a_2^2$ are computed via DSP slices and padded to 33-bit registers to prevent overflow conditions before division.
- **Division Core (17 cycles):** A unified binary restoring divider calculates the magnitude $|N/D|$ iteratively, functioning as a hardware realization of Euclidean long division.

Across 15 clock cycles, the partial remainder is left shifted and conditionally subtracted from the divisor, generating exactly one quotient bit per step. Including a 1-cycle state used to load the absolute operands and a 1-cycle state to store the result, the division phase consumes 17 clock cycles.

- **Coefficients Generation (1 cycle):** Two dedicated DSP multipliers evaluate (5.22) in parallel. The resulting coefficients c_1 and c_2 are registered as 32-bit Q2.30 vectors. On this stage, the module's valid flag is asserted, and the FSM synchronously resets to the idle state, waiting for the next parameter update.

This methodology reduces the computational latency to just 20 clock cycles while achieving mathematical perfection in noise suppression.

5.5.5 The Encryption Arithmetic Pipeline

The encryption pipeline had initial design issues that prevented reliable hardware operation, that were successfully resolved, resulting in a finalized and validated implementation.

Implementation Timing Issues and Pipeline Expansion

The hardware synthesis of real-time applications within FPGA fabrics requires a balance between mathematical accuracy and propagation delays. In the initial design phase, the arithmetic SBE core was mapped into a four-stage pipeline. However, physical implementation resulted in timing violations, indicated by negative Worst Negative Slack (WNS) and Total Negative Slack (TNS) values. As shown in the Figure 5.13, the system clock frequency establishes a temporal window or deadline of $T_{clk} = 8$ ns for data to travel from one register to the next.

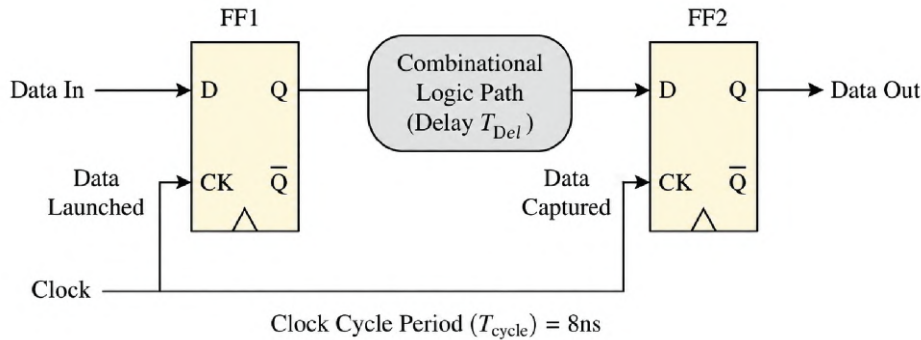


Figure 5.13: Data transfer between registers within timing constraints

In digital design, **slack** is the amount of time remaining in that 8 ns window after the data has successfully arrived at its destination. If the combinational logic (mathematical operations) takes too long to process the data, missing the clock edge deadline. This results in a negative slack value, leading to structural metastability and computational errors.

In the four-stage architecture, the critical bottleneck was localized within the projection and noise scaling phases due to the massive 24×16 bit and 32×32 bit multiplications. Because there were not enough pipeline registers (memory buffers) between these operations, the hardware had to route the data across multiple hardened DSP columns in a single clock cycle, which can lead to false output arithmetical values captured by the register.

To achieve clean timing closure with a positive slack margin, the problem can be eliminated by refactoring the arithmetic architecture into a balanced, eight-stage topology. By inserting pure register breaks at critical logic boundaries, the long operations were divided into smaller, manageable chunks that easily complete within the 8 ns window.

The Eight Stage Arithmetic Pipeline

Fixed-point arithmetic requires careful management of the dynamic range to prevent arithmetic underflow and overflow. To clarify the data flow, the eight-stage encryption pipeline can be divided into four distinct functional phases as detailed from Figure 5.14 to 5.17.

Phase 1: Input Conditioning (Stages 1 & 2)

The data first arrives from the dual-core CORDIC cascade as an absolute magnitude vector. In **Stage 1**, the hardware reconstructs the true two's complement value using a synchronized sign bit. Simultaneously, to eliminate the silent interval previously discussed, a fixed deterministic bias (C_{bias}) is injected into the stream. The hardware dynamically clamps the value to its maximum safe boundary if there is an overflow. Once conditioned, the data passes into **Stage 2**, which acts entirely as a register break. This stage simply buffers the data to physically isolate the routing networks of the CORDIC outputs from the multiplication ahead.

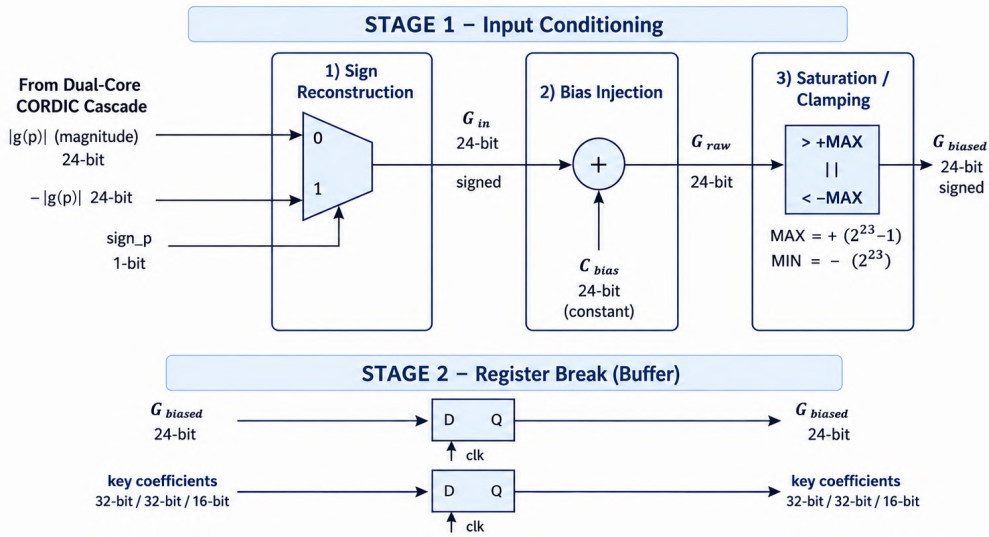


Figure 5.14: Encryption architecture: Phase 1 schematics.

Phase 2: True Random Noise Generation (Stages 3 & 4)

With the data stabilized, **Stage 3** multiplies $\tilde{g}(t)$ by the pseudo-random key $k(t)$ sourced from the TRNG, generating a 40-bit product. To maintain implementation efficiency while preserving cryptographic entropy, the lowest fractional 8 bits that fall below the standard audio noise floor are discarded, leaving a 32-bit scaled noise vector. **Stage 4** is introduced as another register break. This structural buffer catches the newly formed noise vector and stabilizes it before it is fed into the projection cores.

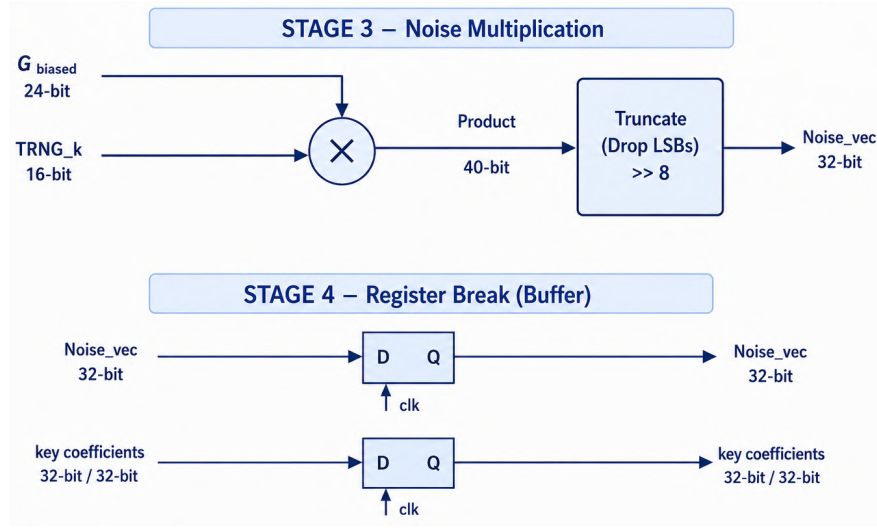


Figure 5.15: Encryption architecture: Phase 2 schematics.

Phase 3: Parallel Geometric Projection (Stages 5 & 6)

This phase executes the fundamental geometric encryption transformations. In **Stage 5**, the hardware fires up a parallel layout of hardened DSP slices to execute two operations simultaneously:

1. The plaintext audio sample $p(t)$ is mapped to its subspace coordinates, producing 48-bit information channels.
2. The noise vector is orthogonally projected using the zero-leakage geometric coefficients $c_1(t)$ and $c_2(t)$ mentioned in Section 5.5.4, 64-bits for each.

To prevent the immense routing delay of these arrays from crashing the timing closure, **Stage 6** immediately captures the output vectors in a register break.

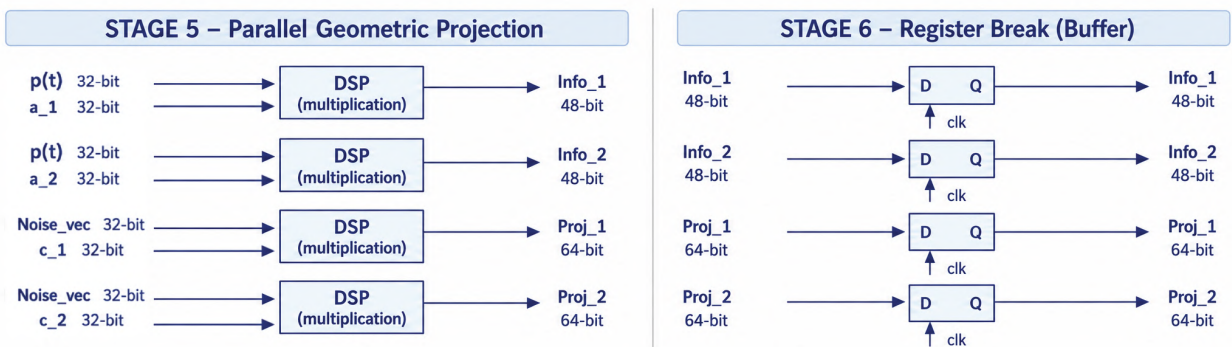


Figure 5.16: Encryption architecture: Phase 3 schematics.

Phase 4: Final Reconstruction and Clamping (Stages 7 & 8)

In **Stage 7**, the 64-bit projection vectors are interpolated by the fixed 8-bit noise intensity parameter (β), resulting in a high precision 72-bit fixed-point noise array. Finally, **Stage 8** resolves the core encryption equation by combining the information and noise streams.

However, directly truncating these vectors back down to a 32-bit output in $Q_{8.24}$ (by removing

the fractional part) would introduce a negative DC bias that decreases the audio's Signal-to-Noise Ratio (SNR). To solve this, Stage 8 implements a symmetric rounding technique. A half-LSB offset is mathematically added into both streams before truncation:

$$\text{info}_{i,\text{rnd}} = \text{info}_i + 2^{14}, \quad \text{noise}_{i,\text{rnd}} = \text{noise}_i + 2^{36}$$

Following this offset injection, the vectors are arithmetically right-shifted to align their binary fractional points perfectly:

$$\text{info}_{i,\text{aligned}} = \text{info}_{i,\text{rnd}} \gg 15, \quad \text{noise}_{i,\text{aligned}} = \text{noise}_{i,\text{rnd}} \gg 37$$

An addition logic combines these aligned values with a saturation logic that overrides the stream, securely clamping the output to standard 32-bit integer limits. The finalized encrypted values are then packed into a 64-bit AXI4-Stream, prepared for Ethernet transmission.

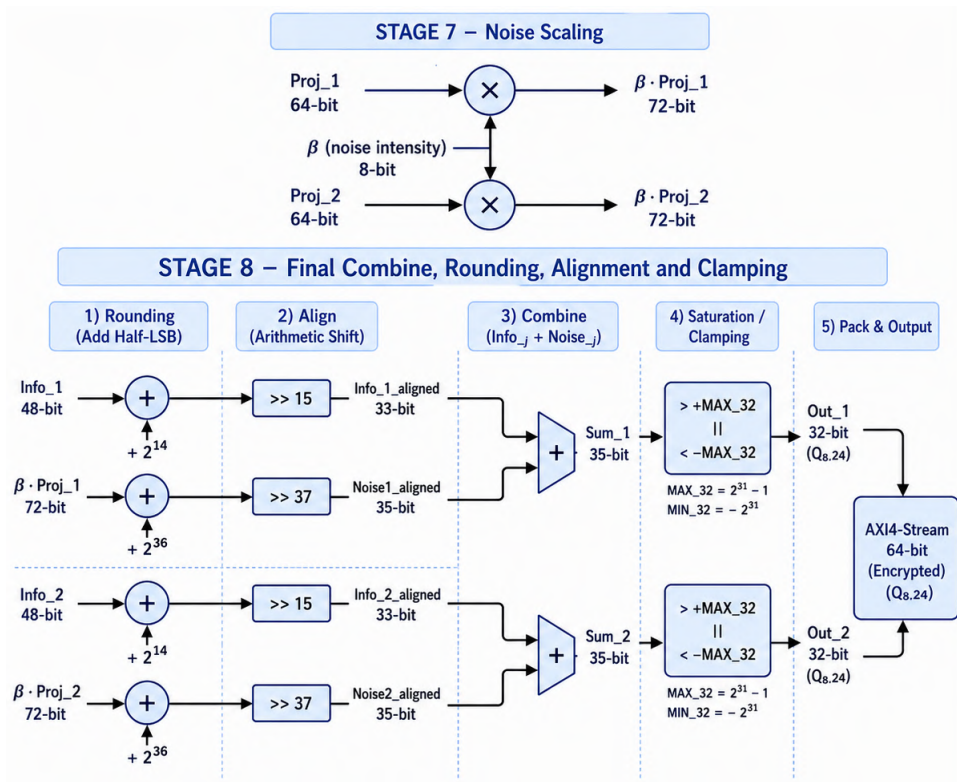


Figure 5.17: Encryption architecture: Phase 4 schematics.

5.6 Decryption Architecture

While the transmitter balances complex orthogonal projections and non-linear transformations, the receiver's primary objective is signal recovery through exact algebraic cancellation and synchronization between the two FPGA boards.

5.6.1 Mathematical Basis

As established in previous chapters, applying the pseudo-inverse $A^\#$ to the ciphertext recovers the plaintext:

$$\hat{p}(t) = A^\# \mathbf{x}(t) = w_1 x_1(t) + w_2 x_2(t), \quad w_i = \frac{a_i}{a_1^2 + a_2^2}, \quad i = 1, 2 \quad (5.24)$$

By expanding this product, the noise term geometrically collapses to exactly zero due to the orthogonality constraint ($a_1 c_1 + a_2 c_2 = 0$). The decryption architecture is a weighted product of the two ciphertext channels. It requires only the time-varying key direction vector $A(t) = [a_1(t), a_2(t)]^T$ without any knowledge of the transmitter's internal chaotic states.

5.6.2 Top-Level Decryption System

The top-level decryption architecture routes the incoming AXI4-Stream data between computational and bypass paths based on the received values.

As shown in Figure 5.18, the architecture is divided into a compute path and a bypass path, dynamically governed by incoming data and an external decryption enable signal.

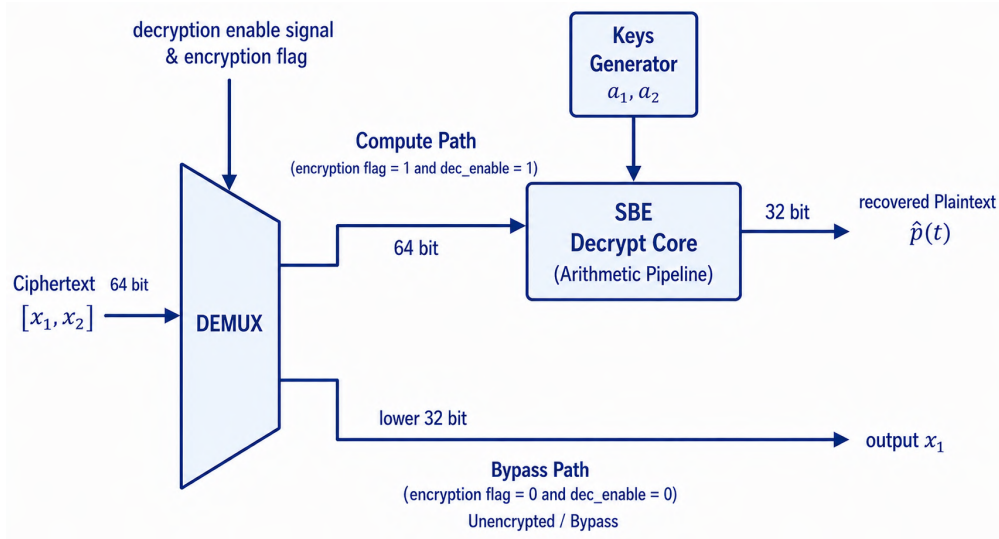


Figure 5.18: Top-level of the decryption architecture.

- **Decryption Path:** Activated when an encrypted frame is detected (the 32 upper bits of the incoming data are non-zero values). It drives the 64-bit ciphertext along with the current key set through the arithmetic core.
- **Bypass Path:** Engaged for unencrypted payloads or when the decryption enable signal is de-asserted. The system routes the lower 32 bits directly to the output interface without arithmetic latency.

5.6.3 Decryption Arithmetic Stages

Similar to the encryption module, the evaluation of equation (5.24) requires the management of combinational delays and fixed-point data ranges. The system is divided into an arithmetic pipeline to ensure a positive timing slack.

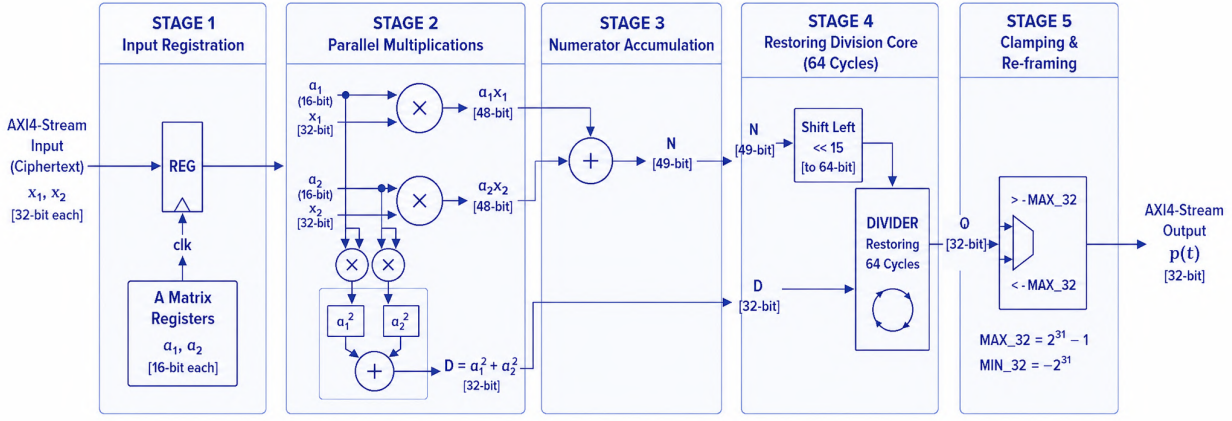


Figure 5.19: Data flow diagram of the decryption system.

1. **Stage 1: Input Registration:** The ciphertext components x_1, x_2 are captured from the incoming AXI frame, and the matrix A parameters required for decryption are read from their holding registers.
2. **Stage 2: Parallel Multiplications:** A parallel arithmetic operation used to compute the partial products. The operations a_1x_1 and a_2x_2 expand into 48-bit signed vectors, and the squared magnitude of the keys ($a_1^2 + a_2^2$) is calculated and buffered into a 32-bit unsigned register, serving as the divisor D for the next stage.
3. **Stage 3: Numerator Accumulation:** The 48-bit partial products are sign extended and added to form a 49-bit signed numerator N .
4. **Restoring Division Core (64 Cycles):** Instead of heavily violating timing with combinational logic, a multi-cycle binary restoring divider evaluates N/D . The 49-bit numerator is formatted into a 64-bit vector (shifted left by 15 bits to preserve fractional precision). Over 64 clock cycles, the operation iteratively evaluates the partial remainders to achieve reliable precision for better audio signal recovery.
5. **Clamping and Re-framing:** Because the fractional division output is highly scaled, the quotient is subjected to boundary checking. The recovered 32-bit signal $\hat{p}(t)$ is finally packed into the outgoing AXI-Stream bus.

5.7 Full-Duplex System Architecture

The preceding sections described each subsystem in isolation. This section presents how they are composed into a unified, full-duplex cryptographic platform operating across two peer-to-peer Zybo Z7-20 boards.

System-Level Overview

Both boards are architecturally identical: each node runs the same firmware and instantiates the complete encryption and decryption pipelines simultaneously. The roles of encryptor and decryptor are therefore relative to the direction of a given audio stream. For a stream originating from Board A, Board A encrypts and Board B decrypts; for the simultaneous stream originating from Board B, the roles are reversed. The complete IP block design is

shown in Figure 5.21.

Outgoing Datapath

Each board continuously captures audio from its local SSM2603 codec at $f_s = 48$ kHz via the I^2S interface. Each 24-bit PCM sample is packed into a 32-bit AXI4-Stream word and fed into the eight-stage SBE encryption pipeline, which produces a 64-bit ciphertext frame within the 1302-cycle budget. The ciphertext, together with the 96-bit nonce and the 1-bit encryption flag, is assembled into a 160-bit five-word block, transferred to the DDR buffer via the MM2S DMA channel, and dispatched as a single encapsulated network packet over the GbE link.

Incoming Datapath

Concurrently, each board receives the transport-layer payload over the GbE link from its peer. The S2MM DMA channel writes the incoming payload into the DDR buffer, from which the PS extracts the nonce and ciphertext. If the encryption flag is asserted, the nonce is written into the local ChaCha20 engine to regenerate the matching $A(t)$ vector, and the ciphertext is forwarded to the SBE decryption core to recover $\hat{p}(t)$. The recovered sample is then serialized over I^2S to the local DAC for playback. If the flag is deasserted, the payload is routed directly to the DAC, and the ChaCha20 state is held, preserving phase alignment for the next encrypted frame.

Path Independence and Synchronization

The two datapaths share no combinational logic. The encryption pipeline is driven by the local LRCLK, while the decryption pipeline processes frames arriving asynchronously from the network. DDR memory is the only shared resource, and the ping-pong buffering scheme ensures that simultaneous DMA accesses in both directions never contend.

The physical realization of the dual-board system is shown in Figure 5.20, where both boards are connected via a standard Cat-5e UTP Ethernet cable, with each board's Line-In and headphone outputs connected to the audio source and playback device, respectively.

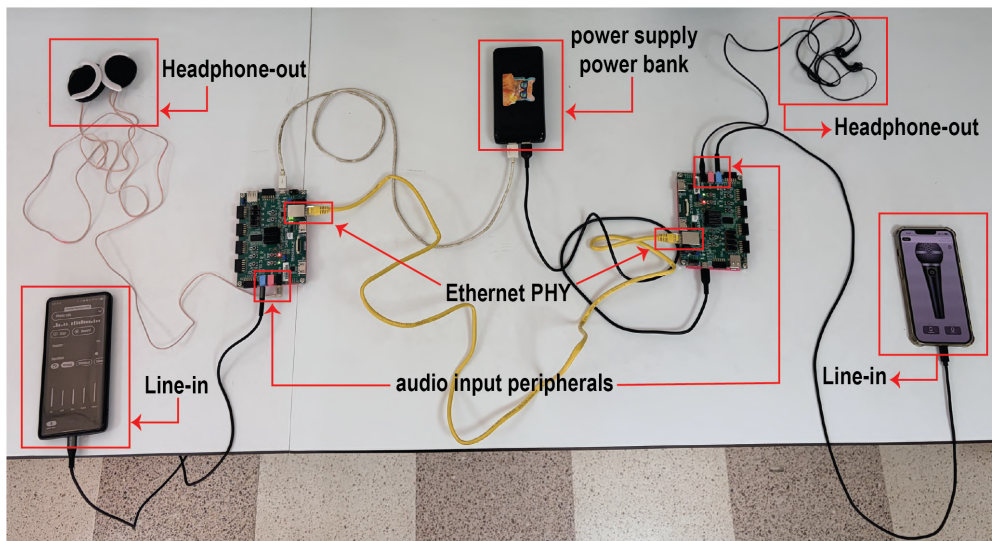


Figure 5.20: Physical implementation of the dual-board full-duplex secure audio communications system.

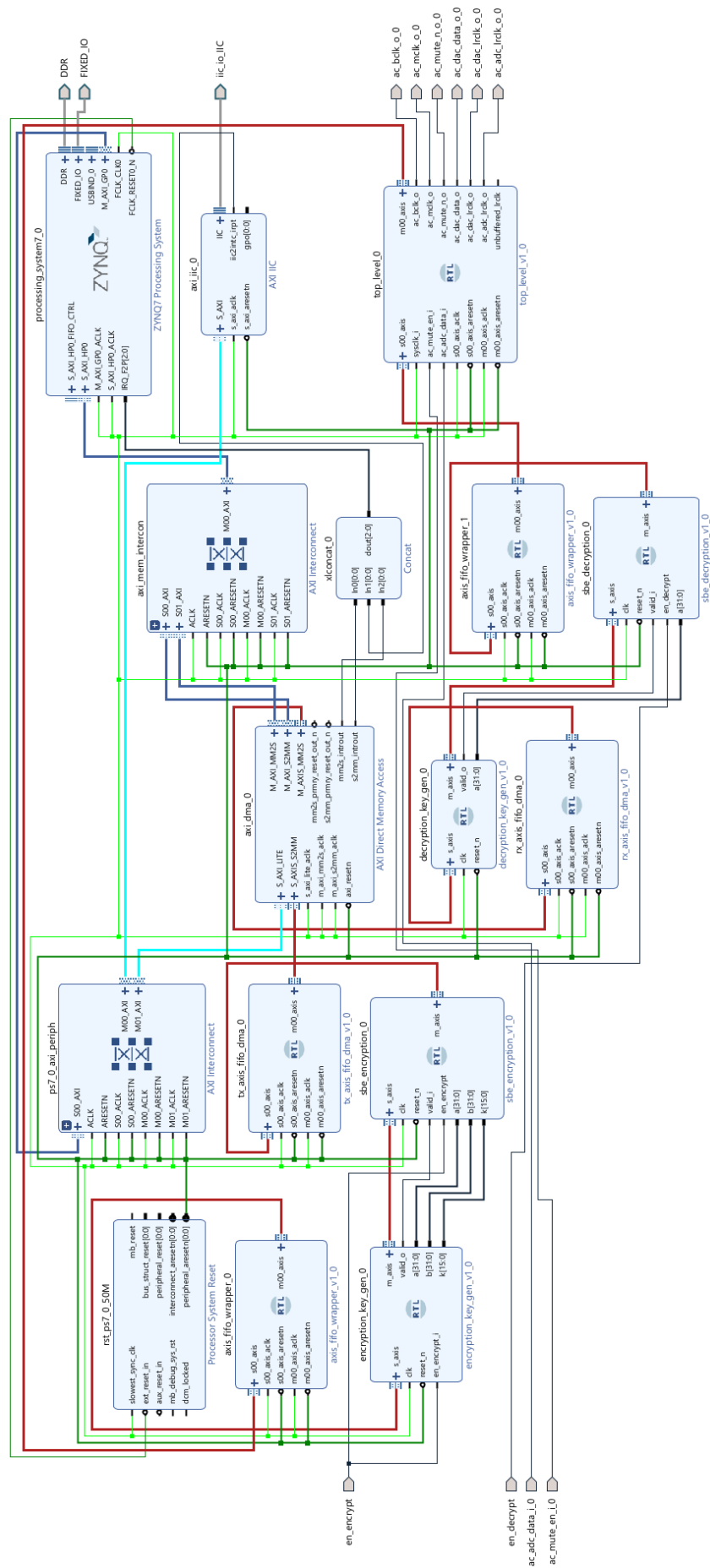


Figure 5.21: Complete IP block design of the full-duplex cryptographic system.

Conclusion

This chapter presented the complete FPGA implementation of the SBE audio encryption system on the Zybo Z7-20 platform. Our proposed architecture successfully meets the real-time constraint of 1302 clock cycles per audio sample, achieving timing closure across all pipeline stages. The non-linear function is evaluated through a dual-CORDIC cascade without any multiplier resources, the orthogonal projection is resolved through an algebraic reformulation that guarantees exact fixed-point orthogonality, and the full encryption path is organized as a balanced eight-stage arithmetic pipeline. On the receiver side, the pseudo-inverse decryption engine recovers the plaintext audio, and the synchronization mechanism maintains strict phase alignment between the two boards. Collectively, these design choices demonstrate that the SBE algorithm can be realized as a robust hardware system capable of operating in real-time applications.

Results and Analysis

This chapter evaluates the proposed secure audio architecture implemented on the Xilinx Zynq-7020 SoC across two complementary perspectives: hardware implementation results and cryptographic security analysis. The hardware evaluation covers resource utilization, timing closure, power consumption, and pipeline latency across four system configurations. The security analysis examines the encrypted audio streams using time-domain, frequency-domain, and statistical metrics and stress-tests the system against blind source separation and subspace-based attacks. The chapter concludes with a comparison against hardware-based audio encryption systems from the literature.

The remainder of this chapter is organized as follows:

- Section 6.1 presents the hardware implementation results, covering resource utilization, timing analysis, power estimation, and latency measurements.
- Section 6.2 develops the cryptographic security analysis through waveform inspection, spectral evaluation, SNR and correlation measurements, and NIST randomness testing.
- Section 6.3 assesses the robustness of the system against blind signal recovery attacks based on blind source separation and subspace-based methods, including SOBI, PCA, MUSIC, Capon/MVDR, and the Bartlett beamformer.
- Section 6.4 positions the proposed system within the existing literature through a structured comparison with related hardware audio encryption architectures.

6.1 Hardware Implementation Results

The implementation results were extracted directly from the Xilinx Vivado ¹ and Vitis ² 2023.1 software. Four project configurations were evaluated:

1. **Standalone Transmitter (Tx):** Contains only the audio capture, SBE encryption pipeline, and TCP/UDP-based data transmission.
2. **Standalone Receiver (Rx):** Contains only the TCP/UDP-based data reception,

¹Vivado is a design tool used for FPGA hardware configurations [60].

²Vitis is a design tool used for FPGA software configurations [60].

decryption pipeline, and audio playback.

3. **Full Duplex (Fixed Nonce):** Integrates both Tx and Rx paths simultaneously, utilizing a fixed nonce for key generation.
4. **Full Duplex (Shared Nonce):** The complete system featuring dynamic key generation, where the RO-TRNG-generated nonce is exchanged between both boards.

6.1.1 Resource Utilization

Table 6.1 summarizes the utilization of implementation resources in the programmable logic fabric. The metrics include Lookup Tables (LUTs), LUTRAM resources, and Flip Flops (FFs), which reflect the logic and storage requirements of the design. In addition to the DSP slices for arithmetic operations, Block RAM (BRAM) resources for data buffering and storage, Input/Output (IO) pins for external interfacing, BUFG resources for global clock distribution, and Mixed-Mode Clock Managers (MMCMs) for clock generation and management.

Table 6.1: Implementation resource utilization on the Zybo Z7-20.

Configuration	LUT	LUTRAM	FF	BRAM	DSP	IO	BUFG	MMCM
Available on Zynq-7020	53,200	17,400	106,400	140	220	125	32	4
Tx Only	11770 (22.12%)	2079 (11.95%)	11822 (11.11%)	1.5 (1.07%)	29 (13.18%)	11 (8.8%)	4 (12.5%)	1 (25%)
Rx Only	6639 (12.48%)	1999 (11.49%)	3329 (3.13%)	1.5 (1.07%)	6 (2.73%)	10 (8%)	4 (12.5%)	1 (25%)
Full Duplex (Fixed Nonce)	19338 (36.35%)	4181 (24.03%)	16756 (15.75%)	3.5 (2.5%)	33 (15%)	12 (9.6%)	4 (12.5%)	1 (25%)
Full Duplex (Shared Nonce)	15569 (29.27%)	2268 (13.03%)	14461 (13.59%)	2.5 (1.79%)	33 (15%)	12 (9.6%)	4 (12.5%)	1 (25%)

Looking at Table 6.1, the standalone Tx demands more resources than the Rx, specifically using 22.12% vs 12.48% LUTs and 29 vs 6 DSPs because it hosts the heavy cryptographic pipelines and multi-stage CORDIC engines.

Moreover, the full duplex (Shared Nonce) configuration actually uses fewer resources than the fixed nonce baseline, which leads to the savings of roughly 5.25% in LUTs and 6.58% in LUTRAM. This all comes down to how the TRNG handles post-processing. The standalone modules and the fixed nonce design use the traditional von Neumann corrector (Section 3.5.2), which consumes extra tracking logic and distributed memory to manage its unpredictable bit-discarding. In contrast, the shared nonce version utilizes a simplified and optimized TRNG key generation. As detailed in Section 3.7, this optimization successfully passes all NIST tests while cutting down the hardware overhead.

6.1.2 Timing Analysis

The overall design timing can be evaluated by the following two metrics. Table 6.2 shows the timing quality for each:

- **Worst Negative Slack (WNS):** It represents the single most design-delayed path (the weakest link of the circuit). A WNS of -3.5 ns means that the slowest operation in the pipeline took 11.5 ns to complete, missing the 8 ns deadline by a significant margin.
- **Total Negative Slack (TNS):** It is the sum of the negative slack of all failing paths in the design. A highly negative TNS means the timing failure is not a routing issue, but a widespread architectural problem.

Table 6.2: Implementation timing quality [3].

Quality Classification	WNS (ps)	TNS (ns)
Excellent	+100 to -100	0 to -10
Good	-100 to -300	-10 to -100
Fair	-300 to -600	-100 to -1,000
Poor	-600 and >	-1,000 and >

Positive Worst Negative Slack (WNS) values confirm that the critical paths within the overall architecture successfully meet the timing requirements without setup or hold violations. Table 6.3 presents the Worst Negative Slack and Total Negative Slack (TNS) for the setup times across all configurations.

Table 6.3: Timing analysis results.

Configuration	WNS (ns)	TNS (ns)
Tx Only	+0.226	0.000
Rx Only	+0.306	0.000
Full Duplex (Fixed Nonce)	+0.222	0.000
Full Duplex (Shared Nonce)	+0.206	0.000

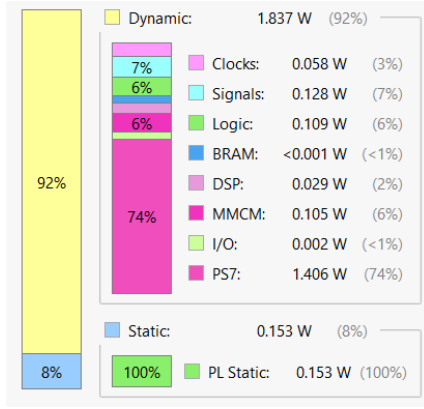
The zero-valued TNS across all configurations confirms successful timing closure without any setup violations. While the simpler Rx path yields the highest margin (+0.306 ns), the WNS tightens to +0.206 ns in the shared-nonce configuration due to the increased routing tracks in the dynamic RO-TRNG loop.

6.1.3 Power Consumption

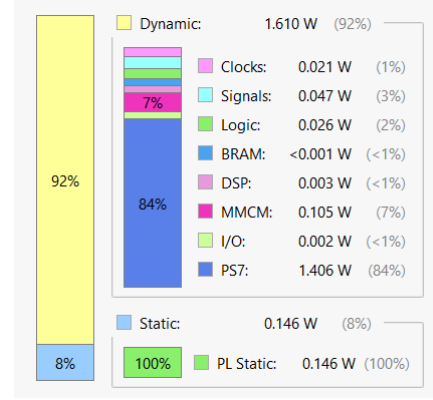
Power consumption was evaluated using the Vivado Power Analysis tool after implementation, categorized into dynamic, static, and total power. Dynamic power corresponds to the energy consumed by switching activities within the PL, including clock networks, logic resources, memories, DSP blocks, and interconnect signals. Static power represents the power required to maintain the device in an operational state, even in the absence of switching activity. The total power is the sum of both previous contributions. Figure 6.1 presents the power estimation reports for the four evaluated configurations.

Power reports indicate that overall consumption is strongly dominated by the Zynq-7020 PS (consistently 1.406 W), while PL contributes minimally. This is fundamentally architectural: the dual-core ARM Cortex-A9 operates at high clock frequency (667 MHz) and results in a continuous switching from instruction fetch-decode cycles, memory controllers, and active memory caches. However, the PL operates at 125 MHz, and the operations are executed directly through hardware pathways (such as logic, DSP) that only consume power during active data transitions. Table 6.4 summarizes the power consumption for all configurations:

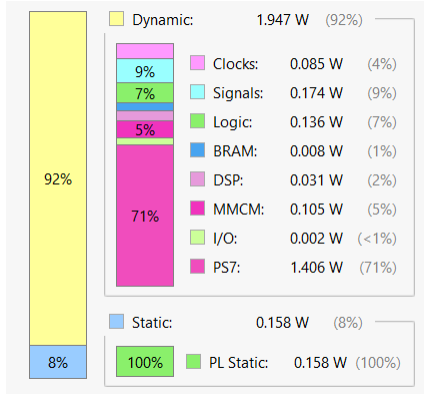
Because static power remains nearly constant (ranging from 0.146 W to 0.158 W) due to identical physical conditions, total power variations depend entirely on dynamic active modules. Notably, the data reveals that the RO-TRNG-based shared nonce configuration consumes slightly less dynamic power compared to the fixed nonce baseline. This reduction aligns directly with the hardware optimizations discussed in Section 6.1.1.



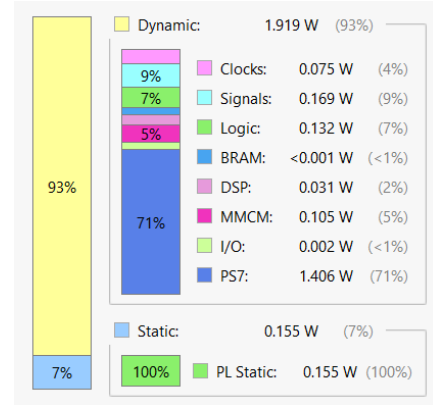
(a) Tx only.



(b) Rx only.



(c) Full-Duplex (fixed nonce).



(d) Full-Duplex (shared nonce).

Figure 6.1: Power analysis for the evaluated system configurations.

Table 6.4: Implementation resource utilization on the Zybo Z7-20.

Configuration	PS Dynamic Power (W)	PL Dynamic Power (W)	Static Power (W)
Tx Only	1.406	0.431	0.153
Rx Only	1.406	0.204	0.146
Full Duplex (Fixed Nonce)	1.406	0.541	0.158
Full Duplex (Shared Nonce)	1.406	0.513	0.155

6.1.4 Pipeline Latency

Pipeline latency is defined as the number of clock cycles elapsed from the moment an audio sample enters the cryptographic pipeline until the corresponding processed frame is asserted at the AXI4-Stream output interface. Table 6.5 breaks down the latency contributions of each hardware sub-module.

To do so, the latency of each stage was measured in real-time on the hardware using the Integrated Logic Analyzer (ILA) ³. By probing ⁴ the AXI4-Stream TVALID signal at both the input and output of the target IP block, and the exact number of system clock cycles between active data transitions, was directly captured.

³ILA is an embedded soft-IP core for real-time debugging that probes internal FPGA signals and buffers the data into BRAM for offline analysis via JTAG.

⁴Probing is used in FPGA debugging. It consists of connecting selected internal signals to the ILA core so that their behavior can be monitored and recorded in real-time.

Table 6.5: Project latency assessment.

Module	Latency Per-Audio Sample
Encryption key derivation (RO-TRNG for dynamic nonce)	263 cycles / 2,096 μ s
Encryption arithmetic pipeline	66 cycles / 0.528 μ s
Decryption key derivation	165 cycles / 1.328 μ s
Decryption arithmetic pipeline	71 cycles / 0.568 μ s

The total latency falls well within the 1302 cycle budget established by the 48 kHz sampling rate. This timing margin leaves sufficient margin for the AXI4-Stream handshake overhead and the Ethernet packetization logic.

6.2 Security Analysis

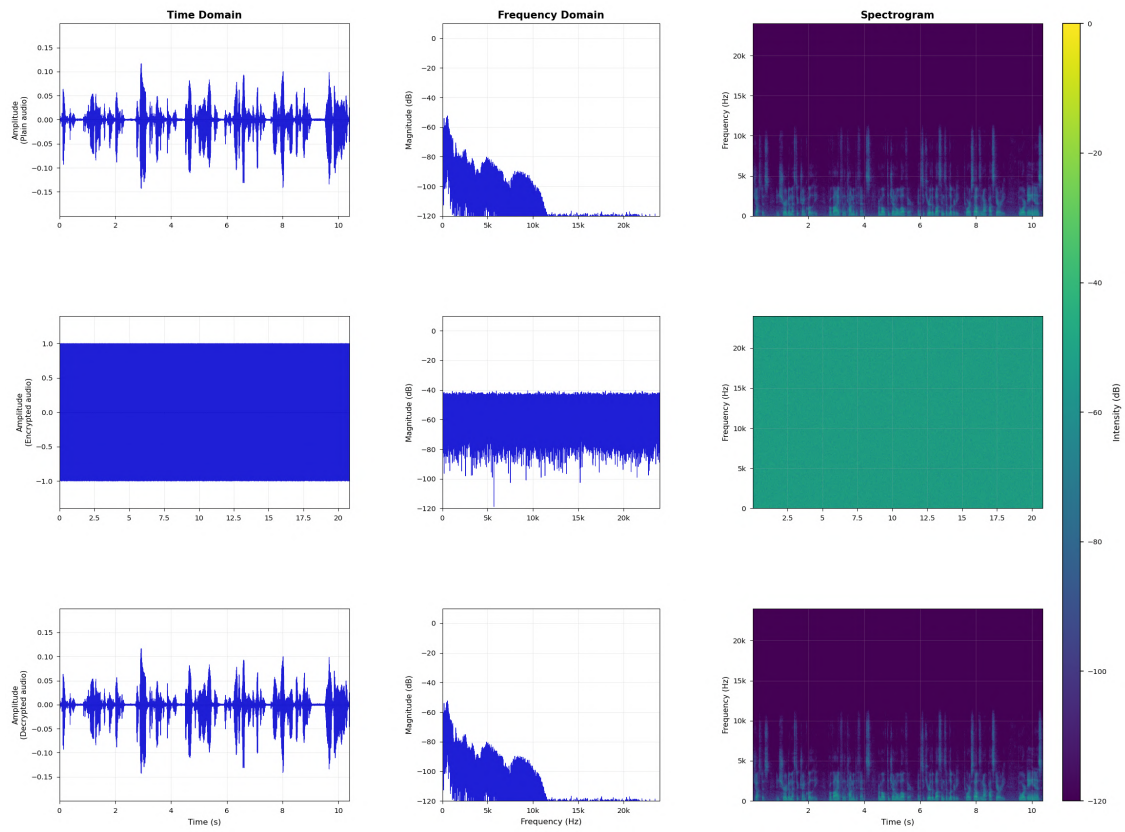
To evaluate cryptographic strength, a unified single-board configuration was deployed where audio is encrypted and immediately decrypted on the same FPGA, ensuring sample-accurate alignment between the plaintext, ciphertext, and recovered plaintext. This bypasses the ILA depth limitation of 131 072 samples, enabling the capture of 480 000 samples per channel at 48 kHz (10 seconds) via DMA to DDR memory, exported as a binary file for offline Python analysis covering time-domain, frequency-domain, entropy, and NIST SP 800-22 statistical metrics.

6.2.1 Time and Frequency Domain Analysis

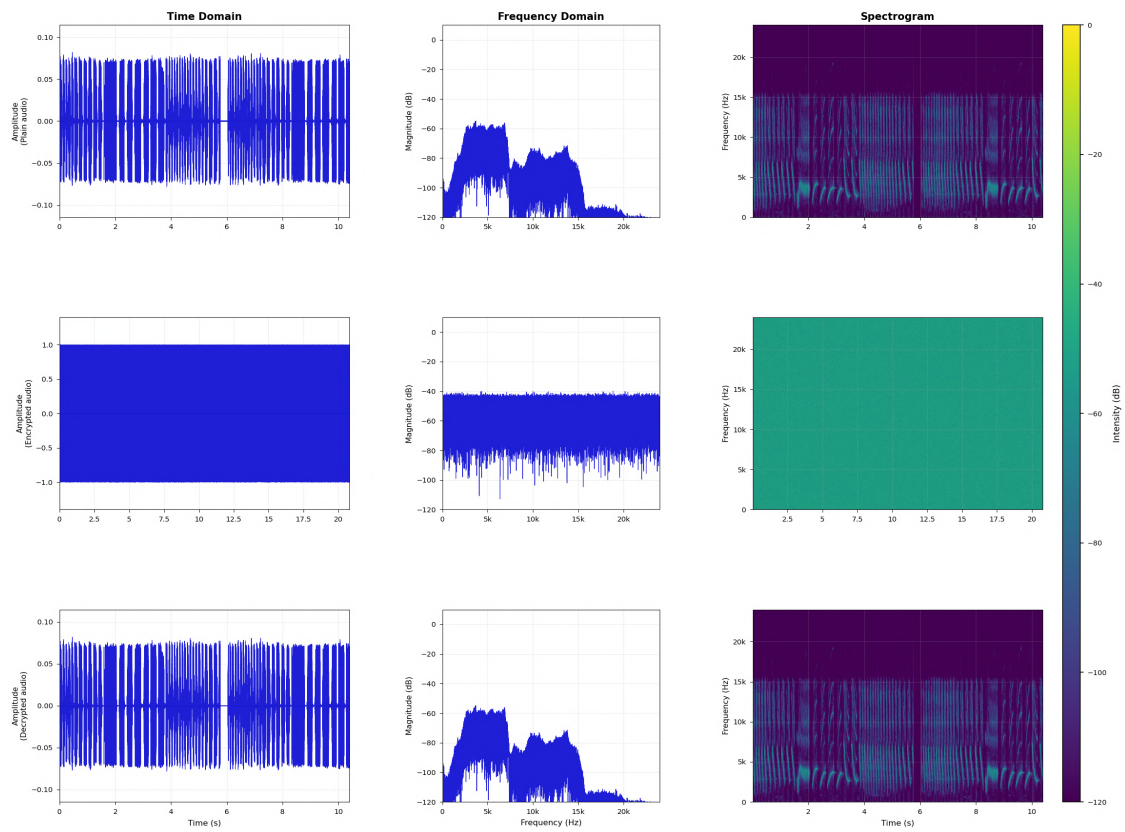
A fundamental requirement of any encryption scheme is that the ciphertext must exhibit no visual or structural resemblance to the original plaintext that destroys any recognizable patterns. As demonstrated in Figure 6.2, this complete examination is achieved across different audio streams.

In the time domain, while the plaintext signals show clear variations corresponding to speech or music, the encrypted signals appear as uniform white noise, masking all rhythmic and structural characteristics. The decrypted signal matches the original one perfectly.

Moreover, the energy of the original audio is concentrated in specific frequency bands typical of human hearing. However, as shown in the spectrograms and frequency plots, the encrypted audio's energy was flattened into an ideal white noise spectrum leveled around -40 dB across the entire 24 kHz bandwidth. This prevents any frequency-based pattern recognition or cryptanalysis.



(a) Signal Analysis - Speech Audio



(b) Signal Analysis - Bird Sound

Figure 6.2: Time, frequency domains, and spectrogram analysis comparing all audio streams.

6.2.2 Signal-to-Noise Ratio

The SNR measures the degree of distortion introduced by the encryption process. It confirms the fidelity of the decryption process and reflects the effect of numerical errors. It is computed as:

$$\text{SNR} = 10 \log_{10} \frac{\sum_n p[n]^2}{\sum_n (\hat{p}[n] - p[n])^2} \quad (6.1)$$

Where $p[n]$ and $\hat{p}[n]$ represent the original and recovered samples, respectively. Table 6.6 and Table 6.7 reports the measured SNR.

A high SNR (87 dB to 109 dB) in both mono and stereo cases confirms that the orthogonality guaranteed by the reformulated projection calculator propagates cleanly through the fixed-point arithmetic pipeline with no measurable accumulation of rounding error.

6.2.3 Statistical Analysis

Two classical statistical metrics were used to assess the degree to which the ciphertext resembles a uniformly random sequence: the cross-correlation between the plaintext and the ciphertext, and the NIST statistical test of the ciphertext samples.

Correlation

The normalized cross-correlation between a 1D plaintext sequence $p[n]$ and a ciphertext sequence $x[n]$ is defined as:

$$\rho = \frac{\sum_n (p[n] - \mu_p)(x[n] - \mu_x)}{\sqrt{\sum_n (p[n] - \mu_p)^2 \sum_n (x[n] - \mu_x)^2}} \quad (6.2)$$

where:

$$\mu_p = \frac{1}{N} \sum_{n=0}^{N-1} p[n], \quad \mu_x = \frac{1}{N} \sum_{n=0}^{N-1} x[n]$$

Since the proposed encryption architecture generates two cipher-text outputs for every single plain-text input sample, equation (6.2) must be adapted into a vector format.

Let $\mathbf{P}_n = [p[n], p[n]]^T$ represent the duplicated plaintext vector at sample index n and $\mathbf{X}_n = [x_1[n], x_2[n]]^T$ denote the corresponding ciphertext output pair. The generalized vector correlation is thus expressed as:

$$\rho = \frac{\sum_n (\mathbf{P}_n - \boldsymbol{\mu}_p)^T (\mathbf{X}_n - \boldsymbol{\mu}_x)}{\sqrt{\sum_n \|\mathbf{P}_n - \boldsymbol{\mu}_p\|^2 \sum_n \|\mathbf{X}_n - \boldsymbol{\mu}_x\|^2}} \quad (6.3)$$

where:

$$\boldsymbol{\mu}_p = \begin{bmatrix} \mu_p \\ \mu_p \end{bmatrix}, \quad \boldsymbol{\mu}_x = \begin{bmatrix} \mu_{x1} \\ \mu_{x2} \end{bmatrix}$$

are the mean vectors of the respective sequences, and $\|\cdot\|$ denotes the Euclidean norm.

A value of $\rho \approx 0$ indicates statistical independence between the two signals. Table 6.6 and Table 6.7 report the computed correlation values

Table 6.6: SNR and correlation analysis for mono audio configuration.

Audio Configuration	Decrypted SNR (dB)		Correlation Coefficient	
	Left	Right	Left	Right
Speech Signal 1	89.49	88.50	-0.000610	+0.000661
Bird Sound	99.08	101.54	+0.000158	-0.000269
Instrumental Sound	93.27	93.43	+0.000559	+0.000903
Song	95.58	94.11	-0.000247	-0.000112

Table 6.7: SNR and correlation analysis for stereo audio configuration.

Audio Configuration		Decrypted SNR (dB)		Correlation Coefficient	
Left Channel	Right Channel	Left	Right	Left	Right
Speech Signal 1	Instrumental Sound	89.38	90.69	-0.000336	-0.001898
Preamble10	Speech Signal 2	87.38	96.81	+0.000945	+0.000841
Speech Signal 3	Bird Sound	90.84	91.97	+0.000385	+0.000846
Song	Instrumental Sound	109.25	104.91	+0.000877	+0.000410

All correlation coefficients hover around zero. This low linear dependence proves that the encrypted audio streams are statistically independent of the plaintext, which demonstrates excellent resistance to statistical cryptanalysis.

NIST Randomness Test Suite

To assess the randomness of the generated ciphertext, the encrypted bitstream was subjected to the NIST test suite. The tests verify whether the sequence exhibits properties expected from a truly random sequence as explained in Section 3.7. Table 6.8 shows that the encrypted stream successfully passed the evaluated tests using 100 bitstreams that contain 10^6 bits for each. This confirms its cryptographic resistance to statistical attacks.

Table 6.8: NIST randomness test results for the encrypted samples.

Statistical Test		100 Bitstreams	
		<i>p</i> -value	Status
Frequency (Monobit)		0.534146	PASS
Block Frequency		0.867692	PASS
Cumulative Sums (Forward)		0.366918	PASS
Cumulative Sums (Reverse)		0.366918	PASS
Runs		0.834308	PASS
Longest Run of Ones		0.066882	PASS
Binary Matrix Rank		0.595549	PASS
DFT (Spectral)		0.534146	PASS
Non-overlapping Template Matching		0.0005–0.9988	PASS
Overlapping Template Matching		0.719747	PASS
Maurer’s Universal Statistical		0.289667	PASS
Approximate Entropy		0.534146	PASS
Serial	Test 1	0.637119	PASS
	Test 2	0.122325	PASS
Linear Complexity		0.474986	PASS
Random Excursions		0.1296–0.7197	PASS
Random Excursions Variant		0.0110–0.7197	PASS

6.3 Subspace and Array Processing Attacks

The SBE encryption scheme maps each plaintext sample $p(t)$ into a two-dimensional observation vector:

$$x(t) = Ap(t) + n(t) \quad (6.4)$$

where A is the secret mixing matrix and $n(t)$ is a composite masking noise term formed by a scaling factor β , a mixing matrix $B(t)$, a random vector $k(t)$, and a non-linear function coupled with the plaintext. This structure is formally identical to the narrowband array signal model used in Direction-of-Arrival (DoA) estimation, which motivates testing the system against representative blind recovery methods.

Attack Methodology

All four attacks are applied to the sample covariance matrix:

$$\hat{R} = \frac{1}{N} \sum_{t=1}^N x(t) x(t)^\top \quad (6.5)$$

estimated from the ciphertext stream, without any knowledge of $A(t)$, $B(t)$, or $k(t)$.

- **Second-Order Blind Identification (SOBI)** [62]: Jointly diagonalizes time-lagged covariance matrices to separate sources via temporal autocorrelation.
- **Principal Component Analysis (PCA)**: Projects observations onto the dominant eigenvector of $\hat{R}$, capturing the direction of maximum variance.
- **Multiple Signal Classification (MUSIC)**: Decomposes $\hat{R}$ into signal and noise subspaces and projects $\mathbf{x}(t)$ onto the estimated signal subspace.

- **Bartlett:** Applies a fixed matched filter using the dominant eigenvector of $\hat{R}$; the simplest non-adaptive subspace projection attack.
- **Capon/MVDR:** Computes an adaptive steering vector minimizing output power while maintaining unit gain, offering finer resolution than Bartlett.

In all cases, the recovered signal is amplitude-scaled to minimize mean-squared error against the plaintext reference, giving each attack the best possible chance of success.

Results and Analysis

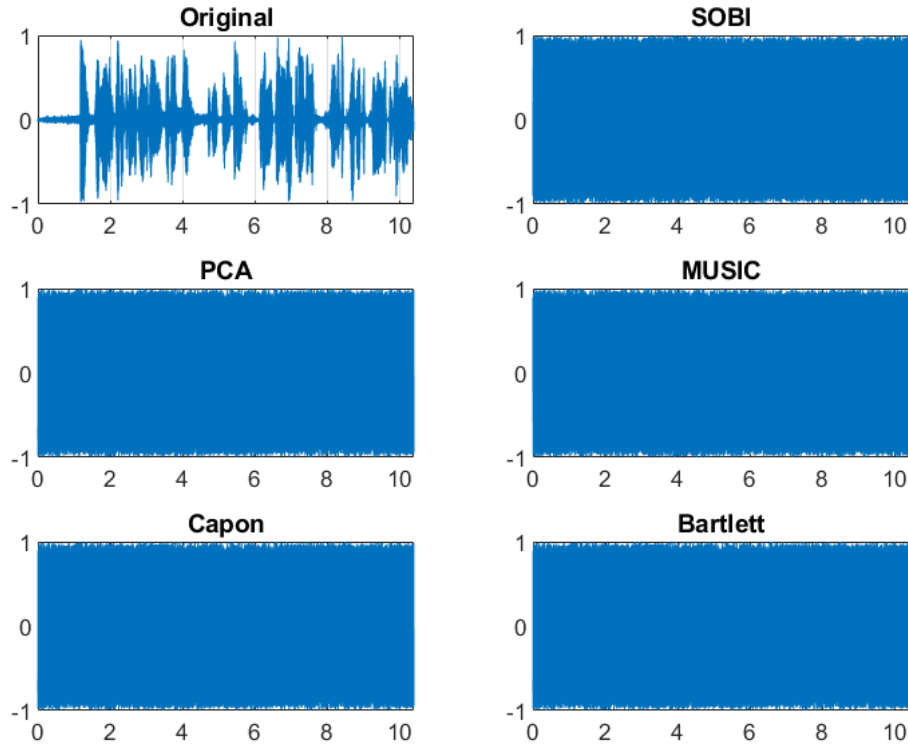


Figure 6.3: Plot of original and blind-attack-recovered signals.

As shown in Figure 6.3, all attack outputs are saturated at the ± 1 amplitude boundary with no observable structure, confirming that none of the five methods extracted meaningful information from the ciphertext. The saturation indicates that projection outputs are dominated by the masking noise, which, being statistically independent of $p(t)$, whitens $\hat{R}$ and destroys the low-rank structure subspace methods rely on. Notably, SOBI, the most statistically powerful attack evaluated, also fails, confirming that the temporal correlation of $p(t)$ is not preserved in the encrypted observations.

Combined with the cryptographic analysis established in Table 2.1 of Section 2.2.3, these results collectively confirm the robustness of our proposed architecture against subspace-based attacks.

6.4 Comparison with Related Work

Our proposed architecture is evaluated against recent hardware-based audio encryption implementations in the literature. Table 6.9 compares key performance indicators, including hardware platform, algorithmic approach, resource consumption, and maximum operating frequency.

Table 6.9: Comparison with existing FPGA-Based Audio encryption architectures.

Reference	Algorithm	Platform	LUTs	DSP	Frequency (MHz)	Channel Mode
[29]	Chaos-Based + XOR	NEXYS A7-100T	13697	120	254.38	Full-Duplex + Wifi
[4]	ANN-Hyperchaotic + XOR	Artix-7	5584	100	105	PRNG + Encryption Core
Proposed Work	ChaCha20/TRNG + SBE	Zybo Z7-20	15569	33	125	Full-Duplex + Ethernet

Compared to the designs mentioned in [29] and [4], our proposed architecture uses fewer DSP slices, whereas both reference models rely on DSP-heavy chaotic or ANN-based key generators. This advantage comes at the cost of a larger LUT footprint, since the CORDIC cascades and the cryptographic key generation pipeline are implemented as combinational logic rather than arithmetic units. The differences in operating frequency are primarily related to the architectural characteristics of each implementation. While [29] achieves the highest clock frequency of 254.38 MHz, the proposed system operates at 125 MHz and [4] at 105 MHz.

In terms of architecture, [29] and [4] both apply bit-level confusion through XOR operations driven by chaotic or learned PRNGs. The proposed system embeds confidentiality directly into the signal's structure. Finally, while the work of [29] establishes full-duplex communications over an XBEE S6B WiFi module, the proposed work targets a wired Gigabit Ethernet link, trading wireless flexibility for higher transmission bandwidth.

Table 6.10 compares the correlation coefficients reported in article [4] with our proposed work for the same audio samples:

Table 6.10: Correlation coefficient comparison with [4].

Audio File	[4]	Proposed Work
gettysburg10.wav	+0.0018	+0.000900
preamble10.wav	+0.0051	+0.000945
StarWars3.wav	+0.0008	−0.000646

Although [4] reports values ranging from 0.0008 to 0.0051, our proposed system consistently achieves a coefficient below 0.001 in all three available samples, nearly an order of magnitude closer to zero. This confirms a lower linear relationship between plaintext and ciphertext, consistent with the non-linear function enforced by the SBE scheme.

Conclusion

This chapter provided an evaluation of our proposed FPGA audio encryption system from both a physical hardware and cryptographic standpoint. The implementation results demonstrate that our design is highly resource-efficient, meeting all timing constraints within a 125 MHz system clock. The transition to a dynamic, shared nonce full-duplex system was achieved with minimal power and area overhead. On the security front, time-frequency visual diagnostics, alongside correlation, SNR, and NIST test metrics, proved that the encrypted audio stream is statistically indistinguishable from white noise. In addition, our proposed architecture establishes itself as a robust solution for secure real-time audio communications when compared with the existing literature.

Conclusion

This work explored the design, implementation, and validation of a complete real-time, full-duplex secure audio communications system based on Subspace-Based Encryption (SBE), deployed on two Digilent Zybo Z7-20 FPGA boards. The work progressed from theoretical cryptographic foundations to a fully functional hardware system, addressing both the algorithm’s mathematical guarantees and the practical constraints of embedded real-time implementation.

The study began by situating SBE within the broader cryptographic landscape, reviewing classical, modern, and BSS-based schemes. This highlighted the linearity flaw of earlier BSS ciphers and motivated the non-linear SBE formulation, where noise confined to the orthogonal subspace of the signal defeats differential and known-plaintext attacks while preserving exact recovery. The oblique variant was also presented, decoupling the signal and null subspaces for enhanced security.

The second phase focused on cryptographic key generation. An initial LFSR served as a hardware baseline but was shown to be insecure under the Berlekamp-Massey attack, motivating a transition to a ChaCha20 keystream generator and a Ring-Oscillator TRNG for physical entropy. The resulting hybrid architecture, ChaCha20, deterministically derives $A(t)$, while RO-TRNG independently supplies $B(t)$ and $k(t)$, passing the entire NIST statistical test suite.

Building on this, the communications channel architecture linked the SSM2603 codec, the AXI4-Stream protocol, and a Gigabit Ethernet link, with sampling rate, bit depth, clocking, and frame structure derived from signal processing and information theory considerations, and a DMA-based double-buffering scheme decoupling audio timing from network jitter.

The core contribution lies in the FPGA implementation, where the SBE equations were translated into a fixed-point, eight-stage pipeline meeting timing closure at 125 MHz. The non-linear coupling was realized through a multiplier-free dual-CORDIC cascade, and an algebraic reformulation of the orthogonal projection eliminated quantization leakage. A synchronization mechanism maintained phase alignment between boards, forming a full-duplex transceiver.

The final evaluation confirmed the design’s practical effectiveness. The system uses a reasonable amount of resources (around 31% LUTs, 33 DSP slices), meets timing closure with positive slack, and shows power dominated by the processing system rather than the cryptographic logic. Security analysis demonstrated a ciphertext indistinguishable from white noise, SNR values between 87–109 dB, and near-zero plaintext-ciphertext correlation.

The final system is a fully pipelined, full-duplex implementation capable of:

- Encrypting and decrypting live 24-bit, 48 kHz audio within the 1302 cycle real-time budget.
- Generating key matrices $B(t)$, $k(t)$, and regenerating $A(t)$ identically on both boards without key transmission.
- Evaluating the non-linear function through a multiplier-free dual-CORDIC cascade.
- Computing the orthogonal projection coefficients with zero quantization leakage via an algebraic reformulation.
- Maintaining synchronization between boards through a data-driven mechanism requiring no side-channel.
- Operating as a symmetric full-duplex transceiver over a Gigabit Ethernet link.

Future work may extend this platform in several directions:

- Adaptation of the communications channel to wireless links (Wi-Fi or RF transceivers).
- Hardware implementation of the oblique SBE variant to fully decouple the signal and null subspaces.
- Implementation of Anti-Replay Attack Mechanisms: To prevent attackers from capturing and retransmitting valid messages in order to gain unauthorized access or disrupt communications.
- Mutual Authentication Between Devices: To ensure that both communicating devices verify each other's identity before exchanging sensitive information.
- Data Integrity Mechanisms: To guarantee that transmitted data has not been modified, altered, or corrupted during transmission through the use of integrity verification techniques such as cryptographic hash functions or Message Authentication Codes (MACs).

Overall, this work demonstrates that Subspace-Based Encryption, previously confined to theoretical and proof-of-concept studies [14], can be realized as a robust, real-time, full-duplex cryptographic system on FPGA hardware.

Bibliography

- [1] K. D. Sharma, H. Saini, and S. Johari, “Mathematical insights into cryptographic algorithms: Recent analysis and future directions,” in *2024 IEEE North Karnataka Subsection Flagship International Conference (NKCon)*, 2024, pp. 1–6.
- [2] P. Sharma, Anjali, R. Kumari, and P. Bansal, “Cryptography: A comparative analysis of symmetric and asymmetric techniques,” in *2024 IEEE 3rd World Conference on Applied Intelligence and Computing (AIC)*, 2024, pp. 1343–1348.
- [3] AMD/Xilinx, *A Methodology for Repeatable and Reliable Timing Closure (WP480)*, August 2016, [Online]. Available: <https://docs.amd.com/v/u/en-US/wp480-timing-closure-methodology>.
- [4] M. S. Azzaz, Y. Alloun, A. Kifouche, A. Kadir, B. Madani, and R. Kaibou, “Fpga-based ann-hyperchaotic prng for secure audio applications,” in *2025 17th International Conference on Advanced Computational Intelligence (ICACI)*, 2025, pp. 79–86.
- [5] A. Mermoul and A. Belouchrani, “A subspace-based method for speech encryption,” in *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, 2010, pp. 538–541.
- [6] R. Serrano, C. Duran, M. Sarmiento, C.-K. Pham, and T.-T. Hoang, “Chacha20–poly1305 authenticated encryption with additional data for transport layer security 1.3,” *Cryptography*, vol. 6, p. 30, 06 2022.
- [7] QSFPTek. (2026) Half-duplex vs full-duplex vs simplex transmission mode. [Accessed: May 16, 2026]. [Online]. Available: <https://www.qsfptek.com/qt-news/half-duplex-vs-full-duplex-vs-simplex-transmission-mode.html>
- [8] Analog Devices, *Low Power Audio Codec SSM2603 Datasheet*, Analog Devices, Inc., 2013, rev. C. [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/ssm2603.pdf>
- [9] Advanced Micro Devices, Inc., *Zynq 7000 SoC Technical Reference Manual (UG585)*, 1st ed., Advanced Micro Devices, Inc., February 2026. [Online]. Available: <https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM>
- [10] I. Winandy, A. Dion, F. Manni, P.-L. Garoche, D. B. Khalifa, and M. Martel, “Automated fixed-point precision optimization for fpga synthesis,” *IEEE Open Journal of Circuits and Systems*, vol. 6, pp. 192–204, 2025.

- [11] A. Mermoul and A. Belouchrani, "Subspace-based technique for speech encryption," *Digital Signal Processing*, vol. 22, no. 2, pp. 298–303, 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S105120041100159X>
- [12] A. Mermoul, "An iterative speech encryption scheme based on subspace technique," in *International Workshop on Systems, Signal Processing and their Applications, WOSSPA*, 2011, pp. 361–364.
- [13] *Proceedings of the 7th International Workshop on Systems, Signal Processing and their Applications (WoSSPA)*. Tipaza, Algeria: IEEE, May 2011.
- [14] A. Mermoul, "Subspace based encryption: Concepts and robustness study," Ph.D. Dissertation, École Nationale Polytechnique (ENP), Algiers, Algeria, 2013. [Online]. Available: <http://repository.enp.edu.dz/xmlui/handle/123456789/423>
- [15] H. M. Ammari, Ed., *Mission-Oriented Sensor Networks and Systems: Art and Science -Volume 1: Foundations*. Springer, 2019, vol. 163. [Online]. Available: <https://doi.org/10.1007/978-3-319-91146-5>
- [16] H. Saif, G. Gharib, and M. Almousa, "A mathematical proposed model for public key encryption algorithms in cybersecurity," *Advances in Mathematics: Scientific Journal*, vol. 10, pp. 3063–3092, 09 2021.
- [17] R. Kikani, K. Verma, R. Navalakhe, G. Shrivastava, and V. Shrivastava, "Cryptography: Recent research trends of encrypting mathematics," *Materials Today: Proceedings*, vol. 56, 10 2021.
- [18] J. Sanchez, R. Correa, H. Buenaño, S. Arias, and H. Gomez, "Encryption techniques: A theoretical overview and future proposals," in *2016 Third International Conference on eDemocracy eGovernment (ICEDEG)*, 2016, pp. 60–64.
- [19] A. Al-Sabaawi, "Cryptanalysis of classic ciphers: Methods implementation survey," in *2021 International Conference on Intelligent Technologies (CONIT)*, 2021, pp. 1–6.
- [20] M. SaberiKamarposhti, A. Ghorbani, and M. Yadollahi, "A comprehensive survey on image encryption: Taxonomy, challenges, and future directions," *Chaos, Solitons Fractals*, vol. 178, p. 114361, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0960077923012638>
- [21] R. Gupta, P. Gupta, and J. Singh, "14 - security and cryptography," in *Software Engineering for Embedded Systems (Second Edition)*, second edition ed., R. Oshana and M. Kraeling, Eds. Newnes, 2019, pp. 501–547. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B978012809448800014X>
- [22] C. Matt and U. Maurer, "The one-time pad revisited," in *2013 IEEE International Symposium on Information Theory*, 2013, pp. 2706–2710.
- [23] E. I. Zafir, A. Akter, M. Islam, S. A. Hasib, T. Islam, S. K. Sarker, and S. Mueeen, "Enhancing security of internet of robotic things: A review of recent trends, practices, and recommendations with encryption and blockchain techniques," *Internet of Things*, vol. 28, p. 101357, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660524002981>

- [24] J. Rosenberg, "Chapter e6 - embedded security," in *Rugged Embedded Systems*, A. Vega, P. Bose, and A. Buyuktosunoglu, Eds. Boston: Morgan Kaufmann, 2017, pp. e1–e74. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128024591000117>
- [25] R. Majumder, S. Datta, and M. Roy, "An enhanced cryptosystem based on modified classical ciphers," in *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, 2022, pp. 692–696.
- [26] M. El Maraghy, S. Hesham, and M. A. Abd El Ghany, "Real-time efficient fpga implementation of aes algorithm," in *2013 IEEE International SOC Conference*, 2013, pp. 203–208.
- [27] S. E. Ghrare and H. A. Bargi, "Design and implementation of encryption and decryption algorithm based on encrypted hidden private key with logical shift and xor operations," in *2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, 2023, pp. 291–294.
- [28] E. A. Albahrani, "A new audio encryption algorithm based on chaotic block cipher," in *2017 Annual Conference on New Trends in Information Communications Technology Applications (NTICT)*, 2017, pp. 22–27.
- [29] A. Kifouche, M. S. Azzaz, Y. Alloun, and W. Merdaci, "Design and implementation on fpga of a real-time secure audio communication system," in *2024 6th International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*, 2024, pp. 1–6.
- [30] R. Nadaf, N. Sumangala, M. Mandi, and A. Konnur, "Symmetric and asymmetric cryptographic approach based security protocol for key exchange," in *2023 International Conference on Applied Intelligence and Sustainable Computing (ICAISC)*, 2023, pp. 1–6.
- [31] E. T. Algareeb and V. O. Osipyan, "A mathematical model of asymmetric encryption based on linear diophantine equations," in *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2024, pp. 1–6.
- [32] Q.-H. Lin, F.-L. Yin, T.-M. Mei, and H. Liang, "A blind source separation-based method for multiple images encryption," *Image and Vision Computing*, vol. 26, no. 6, pp. 788–798, 2008. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0262885607001503>
- [33] N. Doukas and N. Karadimas, "A blind source separation based cryptography scheme for mobile military communication applications," *WSEAS Transactions on Communications*, vol. 7, pp. 1235–1245, 04 2008.
- [34] S. Li, C. Li, K.-T. Lo, and G. Chen, "Cryptanalyzing an encryption scheme based on blind source separation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 4, pp. 1055–1063, 2008.
- [35] A. Kumar, R. Rhouma, Y. Wang, N. Sklavos, and M. Ghose, "Comments on, "a blind source separation-based method for multiple images encryption"," *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 3, pp. 1675–1686, 2011. [Online].

Available: <https://www.sciencedirect.com/science/article/pii/S1007570410003369>

- [36] A. A. M. Khalaf, M. S. A. El-Karim, and H. F. A. Hamed, "A triple hill cipher algorithm proposed to increase the security of encrypted binary data and its implementation using fpga," in *2016 18th International Conference on Advanced Communication Technology (ICACT)*, 2016, pp. 1–1.
- [37] P. Alfke, "Efficient shift registers, lfsr counters, and long pseudo-random sequence generators," Xilinx, Inc., Application Note XAPP052, jul 1996.
- [38] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of applied cryptography, chapter 6: Linear feedback shift registers and stream ciphers," 1996. [Online]. Available: <https://cacr.uwaterloo.ca/hac/about/chap6.pdf>
- [39] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," IETF, Tech. Rep. RFC 7539, 2015. [Online]. Available: <https://tools.ietf.org/html/rfc7539>
- [40] J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger, "New features of latin dances: Analysis of Salsa, ChaCha, and Rumba," in *Fast Software Encryption (FSE 2008)*, ser. LNCS, vol. 5086, 2008, pp. 470–488.
- [41] S. Maitra, "Chosen iv cryptanalysis on reduced round chacha and salsa," *Discrete Applied Mathematics*, vol. 208, pp. 88–97, 2016, first appeared as IACR ePrint in 2015.
- [42] M. Coutinho and T. Neto, "New multi-bit differentials to improve attacks against chacha," IACR Cryptol. ePrint Arch. 2020, 350, 2020.
- [43] S. Dey, H. K. Garai, S. Sarkar, and N. K. Sharma, "Revamped differential-linear cryptanalysis on reduced round ChaCha," in *Advances in Cryptology – EUROCRYPT 2022*, ser. LNCS, vol. 13277, 2022, pp. 86–114.
- [44] S. Dey and H. K. Garai, "Cryptanalysis of reduced round ChaCha – new attack and implementation," IACR ePrint Archive, Tech. Rep. 2023/134, 2023. [Online]. Available: <https://eprint.iacr.org/2023/134.pdf>
- [45] H. K. Garai and S. Dey, "A multi-step key recovery attack on reduced round salsa and chacha," *Cryptologia*, vol. 49, no. 3, pp. 252–267, 2024.
- [46] M. Battagliola, R. Longo, A. Rinaldo, and M. Sala, "Rotational cryptanalysis on ChaCha stream cipher," *Symmetry*, vol. 14, no. 6, p. 1087, 2022.
- [47] KDDI Research, Inc., "Security analysis of ChaCha20-Poly1305 AEAD," CRYPTREC, Tech. Rep., 2017. [Online]. Available: <https://www.cryptrec.go.jp/exreport/cryptrec-ex-2601-2016.pdf>
- [48] "Generating true random numbers with FPGA," <https://www.controlpaths.com/2025/07/13/generating-true-random-numbers-with-fpga/>, 2025, accessed: July 2025.
- [49] M. Hiller *et al.*, "A high-entropy true random number generator with Keccak conditioning for FPGA," *Sensors*, vol. 23, 2023.
- [50] Z. Lu, S. Yang, J. Liu, X. Wang, and Y. Li, "Efficient fpga implementation of high-speed true random number generator," *Review of Scientific Instruments*, vol. 92, no. 2, p.

- 024706, 02 2021. [Online]. Available: <https://doi.org/10.1063/5.0035519>
- [51] J. von Neumann, “Various techniques used in connection with random digits,” in *Applied Mathematics Series*. U.S. National Bureau of Standards, 1951, vol. 12, pp. 36–38.
- [52] N. V. Thanh, “Statistical tests on the ginar random number generator,” Infinity Blockchain Labs, Tech. Rep., October 2018. [Online]. Available: thanhnv@blockchainlabs.asia
- [53] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, and L. E. Bassham, “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. SP 800-22 Rev. 1a, 2010.
- [54] M. Varchola *et al.*, “Revisiting multiple ring oscillator-based true random generators to achieve compact implementations on FPGAs for cryptographic applications,” *Cryptography*, vol. 7, no. 2, p. 26, 2023.
- [55] H. Nyquist, “Certain topics in telegraph transmission theory,” *Transactions of the American Institute of Electrical Engineers*, vol. 47, no. 2, pp. 617–644, 1928.
- [56] C. Shannon, “Communication in the presence of noise,” *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.
- [57] J. G. Proakis and D. G. Manolakis, “Digital signal processing: Principles, algorithms, and applications,” *Pearson Prentice Hall*, 2007.
- [58] Digilent Inc., *Zybo Z7 Reference Manual*, Digilent Inc., 2018, rev. B. [Online]. Available: <https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual>
- [59] ARM Limited, *AMBA 4 AXI4-Stream Protocol Specification*, ARM, 2010. [Online]. Available: <https://developer.arm.com/documentation/ih0051/latest/>
- [60] Advanced Micro Devices, Inc., “Amd vivado design suite,” 2026, accessed: June 21, 2026. [Online]. Available: <https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html>
- [61] S. Arar. An introduction to the CORDIC algorithm. [Online]. Available: <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/>
- [62] A. Belouchrani, K. Abed-Meraim, J.-F. Cardoso, and E. Moulines, “A blind source separation technique using second-order statistics,” *IEEE Transactions on Signal Processing*, vol. 45, no. 2, pp. 434–444, 1997.

Appendix

ChaCha20 (Fixed Nonce, 100 Bitstream)

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <data/chacha20_test3.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
7	11	11	8	11	6	11	16	10	9	0.637119	99/100	Frequency
9	12	9	6	13	11	14	9	12	5	0.554420	98/100	BlockFrequency
7	12	12	6	10	14	6	12	10	11	0.637119	99/100	CumulativeSums
10	6	13	7	10	9	17	9	10	9	0.474986	100/100	CumulativeSums
7	4	13	15	8	7	13	9	14	10	0.224821	100/100	Runs
11	9	11	6	9	11	11	10	8	14	0.897763	97/100	LongestRun
12	16	11	12	10	8	7	13	4	7	0.262249	99/100	Rank
12	14	6	9	10	7	13	5	13	11	0.437274	99/100	FFT
8	10	11	10	11	9	9	7	13	12	0.964295	97/100	NonOverlappingTemplate
8	9	10	14	12	8	11	10	11	7	0.911413	98/100	NonOverlappingTemplate
15	9	8	12	10	8	10	11	10	7	0.851383	99/100	NonOverlappingTemplate
10	12	7	9	13	13	8	11	11	6	0.798139	98/100	NonOverlappingTemplate
17	14	9	6	14	12	8	9	6	5	0.096578	98/100	NonOverlappingTemplate
10	12	9	11	12	8	10	13	9	6	0.911413	98/100	NonOverlappingTemplate
3	5	16	13	8	14	12	5	13	11	0.037566	100/100	NonOverlappingTemplate
10	14	11	7	11	12	7	9	10	9	0.897763	98/100	NonOverlappingTemplate
7	10	11	3	8	10	14	10	14	13	0.319084	98/100	NonOverlappingTemplate
14	12	6	19	8	11	8	5	6	11	0.051942	100/100	NonOverlappingTemplate
8	10	11	15	12	14	6	7	11	6	0.419021	100/100	NonOverlappingTemplate
13	11	5	6	10	16	10	7	11	11	0.366918	99/100	NonOverlappingTemplate
10	10	7	12	12	8	8	11	13	9	0.935716	99/100	NonOverlappingTemplate
9	9	9	7	11	13	8	12	13	9	0.911413	100/100	NonOverlappingTemplate
10	10	13	11	13	6	9	11	10	7	0.867692	99/100	NonOverlappingTemplate
10	8	11	5	6	11	11	10	14	14	0.534146	100/100	NonOverlappingTemplate
17	7	8	8	12	6	10	12	11	9	0.419021	98/100	NonOverlappingTemplate
15	4	10	14	5	9	10	9	17	7	0.062821	100/100	NonOverlappingTemplate
9	9	13	8	10	9	13	13	10	6	0.834308	99/100	NonOverlappingTemplate
10	8	6	15	5	7	13	14	12	10	0.289667	97/100	NonOverlappingTemplate
7	13	12	9	12	18	8	7	6	8	0.191687	100/100	NonOverlappingTemplate
10	10	13	9	13	16	4	7	12	6	0.213309	98/100	NonOverlappingTemplate
10	3	8	14	2	16	11	12	15	9	0.017912	99/100	NonOverlappingTemplate
10	10	9	12	9	7	10	14	4	15	0.419021	100/100	NonOverlappingTemplate
12	11	4	7	11	11	12	11	6	15	0.366918	98/100	NonOverlappingTemplate
12	8	7	9	14	17	10	7	9	7	0.334538	99/100	NonOverlappingTemplate

12	11	6	11	6	9	17	12	7	9	0.334538	97/100	NonOverlappingTemplate
8	12	10	13	10	9	10	7	12	9	0.955835	100/100	NonOverlappingTemplate
12	10	13	5	14	4	18	8	13	3	0.010237	98/100	NonOverlappingTemplate
9	6	17	5	11	12	6	9	16	9	0.090936	99/100	NonOverlappingTemplate
8	10	6	9	11	13	9	7	13	14	0.678686	98/100	NonOverlappingTemplate
7	9	9	10	14	10	11	10	9	11	0.964295	98/100	NonOverlappingTemplate
11	9	14	15	6	13	12	2	8	10	0.122325	99/100	NonOverlappingTemplate
12	7	15	7	13	14	8	12	5	7	0.249284	98/100	NonOverlappingTemplate
8	8	9	12	5	12	10	13	12	11	0.779188	99/100	NonOverlappingTemplate
5	9	16	8	9	8	13	12	11	9	0.474986	100/100	NonOverlappingTemplate
8	8	12	15	12	9	8	12	9	7	0.739918	100/100	NonOverlappingTemplate
12	8	8	11	8	12	14	6	11	10	0.798139	99/100	NonOverlappingTemplate
8	13	6	15	12	8	12	9	11	6	0.494392	100/100	NonOverlappingTemplate
6	8	8	7	9	14	6	10	11	21	0.026948	100/100	NonOverlappingTemplate
9	7	8	6	10	9	12	11	16	12	0.574903	99/100	NonOverlappingTemplate
8	9	7	12	10	11	10	8	10	15	0.851383	98/100	NonOverlappingTemplate
9	10	9	9	13	12	8	10	12	8	0.971699	97/100	NonOverlappingTemplate
12	10	8	7	11	8	7	13	14	10	0.779188	97/100	NonOverlappingTemplate
5	16	6	7	11	16	11	10	9	9	0.181557	100/100	NonOverlappingTemplate
7	13	9	11	10	9	10	9	11	11	0.983453	98/100	NonOverlappingTemplate
13	4	11	12	7	9	11	11	14	8	0.514124	99/100	NonOverlappingTemplate
5	16	11	8	7	10	9	15	10	9	0.334538	99/100	NonOverlappingTemplate
13	9	13	8	10	10	11	9	6	11	0.897763	97/100	NonOverlappingTemplate
13	9	6	10	8	6	7	15	11	15	0.304126	98/100	NonOverlappingTemplate
16	13	6	10	10	9	12	7	6	11	0.419021	99/100	NonOverlappingTemplate
8	10	13	12	9	10	14	10	7	7	0.816537	98/100	NonOverlappingTemplate
5	9	8	15	10	9	10	13	8	13	0.554420	100/100	NonOverlappingTemplate
6	8	18	12	8	11	5	13	8	11	0.153763	99/100	NonOverlappingTemplate
9	10	8	8	10	11	15	6	11	12	0.779188	100/100	NonOverlappingTemplate
14	11	8	12	7	5	8	7	12	16	0.262249	100/100	NonOverlappingTemplate
12	8	13	5	15	7	12	8	11	9	0.474986	98/100	NonOverlappingTemplate
13	10	7	15	9	11	9	4	8	14	0.334538	97/100	NonOverlappingTemplate
9	12	11	5	13	11	9	13	9	8	0.779188	99/100	NonOverlappingTemplate
8	10	9	6	8	14	14	10	9	12	0.719747	100/100	NonOverlappingTemplate
13	9	13	11	7	9	9	11	7	11	0.897763	100/100	NonOverlappingTemplate
9	10	7	10	14	9	7	8	13	13	0.759756	99/100	NonOverlappingTemplate
10	6	13	10	11	6	9	15	9	11	0.637119	99/100	NonOverlappingTemplate
7	12	9	8	13	9	15	11	9	7	0.699313	100/100	NonOverlappingTemplate
15	8	12	18	9	6	6	7	13	6	0.058984	96/100	NonOverlappingTemplate
8	10	8	7	12	10	8	13	11	13	0.883171	100/100	NonOverlappingTemplate
10	13	9	8	6	9	16	12	8	9	0.574903	99/100	NonOverlappingTemplate
11	15	12	5	11	11	9	7	12	7	0.534146	98/100	NonOverlappingTemplate
15	5	8	13	8	9	7	10	12	13	0.437274	98/100	NonOverlappingTemplate
13	7	11	9	6	11	16	11	9	7	0.494392	98/100	NonOverlappingTemplate
10	9	9	15	8	7	7	16	7	12	0.366918	99/100	NonOverlappingTemplate
7	13	7	7	14	15	4	11	11	11	0.236810	100/100	NonOverlappingTemplate
5	8	15	6	8	15	13	11	10	9	0.275709	99/100	NonOverlappingTemplate
10	10	5	10	10	11	13	11	11	9	0.924076	99/100	NonOverlappingTemplate
8	10	11	10	11	9	9	7	13	12	0.964295	97/100	NonOverlappingTemplate
6	8	14	13	8	15	14	5	6	11	0.153763	100/100	NonOverlappingTemplate
10	12	7	9	15	8	10	12	6	11	0.699313	98/100	NonOverlappingTemplate
12	14	10	8	8	7	11	5	13	12	0.574903	100/100	NonOverlappingTemplate
9	8	7	14	11	17	12	6	9	7	0.275709	99/100	NonOverlappingTemplate
8	11	10	8	8	9	14	13	12	7	0.816537	100/100	NonOverlappingTemplate
10	10	8	7	15	3	13	11	10	13	0.304126	100/100	NonOverlappingTemplate
12	5	13	7	12	9	7	10	14	11	0.554420	99/100	NonOverlappingTemplate
14	6	12	4	12	9	14	11	5	13	0.171867	97/100	NonOverlappingTemplate
6	10	10	2	21	13	6	12	9	11	0.005762	99/100	NonOverlappingTemplate
8	11	11	17	6	10	9	9	8	11	0.554420	100/100	NonOverlappingTemplate

5	13	9	12	10	8	11	9	15	8	0.595549	100/100	NonOverlappingTemplate
5	10	13	8	10	8	13	8	15	10	0.534146	99/100	NonOverlappingTemplate
11	11	8	15	13	11	11	5	9	6	0.494392	98/100	NonOverlappingTemplate
5	7	9	7	12	13	12	11	11	13	0.616305	100/100	NonOverlappingTemplate
10	10	13	13	11	10	7	9	6	11	0.867692	98/100	NonOverlappingTemplate
10	13	3	14	7	4	9	19	10	11	0.016717	99/100	NonOverlappingTemplate
10	8	10	7	8	18	6	13	12	8	0.249284	100/100	NonOverlappingTemplate
7	10	12	7	16	8	9	6	18	7	0.085587	100/100	NonOverlappingTemplate
7	13	10	7	10	5	16	10	9	13	0.366918	100/100	NonOverlappingTemplate
15	5	10	8	4	7	13	11	11	16	0.102526	99/100	NonOverlappingTemplate
11	13	9	9	12	7	15	10	9	5	0.574903	100/100	NonOverlappingTemplate
12	11	8	3	15	10	18	5	10	8	0.040108	99/100	NonOverlappingTemplate
10	12	8	13	6	11	6	16	10	8	0.437274	100/100	NonOverlappingTemplate
11	10	7	5	10	10	17	8	13	9	0.366918	100/100	NonOverlappingTemplate
6	12	8	9	15	11	8	16	5	10	0.236810	100/100	NonOverlappingTemplate
4	9	13	11	6	11	9	12	7	18	0.115387	99/100	NonOverlappingTemplate
9	15	10	7	7	8	12	12	11	9	0.759756	100/100	NonOverlappingTemplate
16	9	7	9	12	12	8	13	7	7	0.474986	98/100	NonOverlappingTemplate
8	10	9	8	11	8	10	13	8	15	0.816537	100/100	NonOverlappingTemplate
5	14	9	11	14	7	6	13	9	12	0.366918	100/100	NonOverlappingTemplate
13	11	9	9	12	10	9	9	8	10	0.987896	98/100	NonOverlappingTemplate
10	12	6	17	6	14	9	7	10	9	0.262249	99/100	NonOverlappingTemplate
14	9	7	7	12	8	12	13	5	13	0.437274	99/100	NonOverlappingTemplate
10	8	9	8	7	18	9	12	9	10	0.455937	100/100	NonOverlappingTemplate
9	12	7	8	16	10	8	11	9	10	0.739918	99/100	NonOverlappingTemplate
11	15	7	6	7	9	9	14	13	9	0.455937	96/100	NonOverlappingTemplate
7	11	12	7	8	15	13	13	6	8	0.437274	100/100	NonOverlappingTemplate
12	3	15	13	12	12	7	10	3	13	0.062821	98/100	NonOverlappingTemplate
9	8	11	12	16	10	7	6	15	6	0.262249	99/100	NonOverlappingTemplate
12	9	7	9	10	8	14	10	9	12	0.911413	99/100	NonOverlappingTemplate
11	6	8	10	8	15	10	10	10	12	0.798139	98/100	NonOverlappingTemplate
9	11	9	15	6	8	11	8	13	10	0.719747	99/100	NonOverlappingTemplate
6	14	9	12	11	4	10	11	8	15	0.319084	99/100	NonOverlappingTemplate
7	13	11	9	8	8	17	8	11	8	0.474986	99/100	NonOverlappingTemplate
10	8	12	10	9	6	12	10	12	11	0.946308	99/100	NonOverlappingTemplate
12	11	9	4	10	14	13	7	12	8	0.494392	100/100	NonOverlappingTemplate
15	12	7	8	9	10	7	14	8	10	0.616305	98/100	NonOverlappingTemplate
13	10	11	12	5	6	10	10	12	11	0.739918	99/100	NonOverlappingTemplate
7	6	8	9	9	10	11	12	14	14	0.657933	100/100	NonOverlappingTemplate
9	8	8	14	11	9	11	17	5	8	0.304126	97/100	NonOverlappingTemplate
6	15	15	10	9	8	9	13	7	8	0.401199	99/100	NonOverlappingTemplate
12	9	8	9	8	12	11	12	9	10	0.983453	99/100	NonOverlappingTemplate
6	1	14	9	8	11	19	11	8	13	0.010988	100/100	NonOverlappingTemplate
10	11	10	13	12	7	12	12	6	7	0.779188	100/100	NonOverlappingTemplate
10	7	10	12	9	7	12	13	13	7	0.798139	99/100	NonOverlappingTemplate
7	8	14	10	16	12	9	9	11	4	0.289667	100/100	NonOverlappingTemplate
13	11	11	9	14	10	6	10	5	11	0.637119	100/100	NonOverlappingTemplate
5	9	21	5	4	7	11	12	9	17	0.001296	100/100	NonOverlappingTemplate
4	8	14	8	11	13	10	10	11	11	0.616305	100/100	NonOverlappingTemplate
11	8	5	5	9	11	14	15	8	14	0.224821	99/100	NonOverlappingTemplate
14	12	3	9	14	12	10	8	11	7	0.319084	98/100	NonOverlappingTemplate
20	13	9	7	15	8	10	5	5	8	0.016717	100/100	NonOverlappingTemplate
3	10	8	11	15	7	12	14	10	10	0.289667	100/100	NonOverlappingTemplate
9	9	7	12	7	12	10	13	8	13	0.834308	98/100	NonOverlappingTemplate
13	6	7	10	7	15	9	10	14	9	0.474986	98/100	NonOverlappingTemplate
11	6	7	11	9	10	18	11	10	7	0.334538	100/100	NonOverlappingTemplate
3	8	11	16	10	11	12	7	15	7	0.129620	100/100	NonOverlappingTemplate
11	12	12	12	8	9	8	14	4	10	0.595549	98/100	NonOverlappingTemplate
9	10	6	10	17	11	8	13	8	8	0.455937	99/100	NonOverlappingTemplate

11	8	18	12	12	4	9	7	9	10	0.191687	98/100	NonOverlappingTemplate
5	10	8	13	10	8	12	9	13	12	0.739918	100/100	NonOverlappingTemplate
6	7	8	11	13	9	9	8	17	12	0.366918	99/100	NonOverlappingTemplate
10	9	6	10	11	10	13	11	11	9	0.964295	99/100	NonOverlappingTemplate
8	11	11	12	12	8	10	9	10	9	0.991468	99/100	OverlappingTemplate
14	4	18	7	12	5	9	9	10	12	0.066882	99/100	Universal
8	7	16	6	8	11	13	8	9	14	0.350485	97/100	ApproximateEntropy
2	9	9	1	6	8	8	7	5	6	0.242986	61/61	RandomExcursions
3	6	7	7	5	9	7	7	7	3	0.788728	60/61	RandomExcursions
6	4	4	6	5	7	10	5	5	9	0.723129	60/61	RandomExcursions
5	5	8	5	6	3	8	6	10	5	0.723129	60/61	RandomExcursions
4	5	6	4	10	8	5	8	6	5	0.756476	61/61	RandomExcursions
5	5	4	7	4	7	11	7	6	5	0.689019	61/61	RandomExcursions
4	9	4	5	7	5	5	9	8	5	0.756476	61/61	RandomExcursions
7	9	5	6	3	5	9	5	7	5	0.788728	60/61	RandomExcursions
9	3	6	9	8	6	4	3	7	6	0.585209	60/61	RandomExcursionsVariant
6	6	5	5	14	4	3	6	6	6	0.155209	60/61	RandomExcursionsVariant
5	5	6	4	10	3	7	6	9	6	0.654467	61/61	RandomExcursionsVariant
3	7	4	3	6	12	6	6	6	8	0.311542	61/61	RandomExcursionsVariant
2	5	6	7	5	4	10	5	9	8	0.452799	61/61	RandomExcursionsVariant
2	4	7	8	9	4	5	6	7	9	0.517442	61/61	RandomExcursionsVariant
2	4	7	6	7	8	4	7	7	9	0.654467	61/61	RandomExcursionsVariant
6	5	7	6	5	2	3	13	9	5	0.105618	61/61	RandomExcursionsVariant
6	4	5	9	6	11	4	4	6	6	0.551026	61/61	RandomExcursionsVariant
4	9	2	7	6	5	9	8	3	8	0.392456	61/61	RandomExcursionsVariant
4	5	9	5	6	7	4	7	5	9	0.819544	61/61	RandomExcursionsVariant
4	5	8	9	5	6	5	8	4	7	0.848588	61/61	RandomExcursionsVariant
2	5	10	6	8	7	8	3	8	4	0.364146	61/61	RandomExcursionsVariant
3	3	12	4	6	5	10	4	7	7	0.141256	61/61	RandomExcursionsVariant
1	7	6	6	3	9	7	11	8	3	0.128379	61/61	RandomExcursionsVariant
1	7	4	4	9	10	9	7	5	5	0.222869	61/61	RandomExcursionsVariant
3	5	2	7	8	7	8	8	7	6	0.654467	61/61	RandomExcursionsVariant
4	4	2	7	7	10	12	7	3	5	0.095617	61/61	RandomExcursionsVariant
7	15	7	9	13	10	10	5	7	17	0.137282	99/100	Serial
9	9	12	4	10	17	9	9	10	11	0.401199	99/100	Serial
11	10	10	9	12	11	10	7	10	10	0.996335	100/100	LinearComplexity

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 96 for a sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately = 58 for a sample size = 61 binary sequences.

For further guidelines construct a probability table using the MAPLE program provided in the addendum section of the documentation.

RO-TRNG (100 Bitstream)

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <data/trng_16R0.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
10	9	10	8	6	18	9	8	10	12	0.401199	99/100	Frequency
9	15	9	13	6	11	12	9	5	11	0.494392	100/100	BlockFrequency
11	9	13	16	7	10	12	6	9	7	0.474986	99/100	CumulativeSums
6	11	11	10	9	11	17	12	6	7	0.366918	99/100	CumulativeSums
13	12	8	11	7	7	10	8	16	8	0.534146	98/100	Runs
11	8	10	8	11	9	10	12	9	12	0.991468	99/100	LongestRun
9	11	11	10	11	7	7	8	10	16	0.719747	98/100	Rank
6	8	20	5	6	13	7	10	13	12	0.023545	99/100	FFT
10	10	10	11	7	15	8	15	3	11	0.249284	99/100	NonOverlappingTemplate
8	9	10	8	8	13	17	7	10	10	0.534146	99/100	NonOverlappingTemplate
12	11	13	7	11	7	11	13	7	8	0.779188	99/100	NonOverlappingTemplate
3	8	15	10	13	12	6	10	9	14	0.191687	100/100	NonOverlappingTemplate
12	9	11	10	9	14	7	14	8	6	0.657933	100/100	NonOverlappingTemplate
7	15	9	14	11	8	6	8	12	10	0.534146	97/100	NonOverlappingTemplate
6	13	10	9	7	13	7	14	8	13	0.514124	99/100	NonOverlappingTemplate
10	6	13	11	10	11	7	7	12	13	0.759756	97/100	NonOverlappingTemplate
15	10	6	11	11	7	8	5	12	15	0.275709	97/100	NonOverlappingTemplate
7	10	11	12	13	11	6	10	9	11	0.897763	100/100	NonOverlappingTemplate
10	9	9	12	6	12	6	11	13	12	0.779188	100/100	NonOverlappingTemplate
14	3	10	9	16	10	8	10	11	9	0.289667	99/100	NonOverlappingTemplate
7	12	12	9	9	11	13	6	13	8	0.759756	99/100	NonOverlappingTemplate
14	7	13	13	8	10	8	10	10	7	0.739918	98/100	NonOverlappingTemplate
10	6	11	15	16	11	6	12	6	7	0.191687	100/100	NonOverlappingTemplate
10	7	8	11	3	7	8	7	14	25	0.000157	98/100	NonOverlappingTemplate
13	14	6	13	8	10	11	5	13	7	0.366918	98/100	NonOverlappingTemplate
11	12	9	10	7	10	8	10	12	11	0.983453	98/100	NonOverlappingTemplate
7	5	12	16	10	11	13	10	10	6	0.350485	98/100	NonOverlappingTemplate
11	14	4	15	15	9	12	10	4	6	0.066882	97/100	NonOverlappingTemplate
10	9	10	7	15	8	8	11	6	16	0.383827	99/100	NonOverlappingTemplate
12	7	12	8	9	12	7	10	5	18	0.191687	99/100	NonOverlappingTemplate
8	13	9	6	12	6	13	13	14	6	0.350485	99/100	NonOverlappingTemplate
7	11	15	8	9	13	11	15	9	2	0.122325	100/100	NonOverlappingTemplate
6	14	17	8	9	8	10	6	13	9	0.236810	99/100	NonOverlappingTemplate
13	5	10	14	10	15	9	4	7	13	0.162606	99/100	NonOverlappingTemplate
7	14	9	8	13	16	10	11	7	5	0.275709	100/100	NonOverlappingTemplate
8	13	7	11	11	9	9	12	9	11	0.955835	98/100	NonOverlappingTemplate
14	9	13	7	6	9	7	13	12	10	0.595549	99/100	NonOverlappingTemplate
11	7	9	11	9	12	10	9	10	12	0.987896	99/100	NonOverlappingTemplate
10	10	16	12	10	5	10	9	12	6	0.474986	98/100	NonOverlappingTemplate
9	13	12	10	10	5	9	10	11	11	0.897763	99/100	NonOverlappingTemplate
18	12	9	7	9	9	15	10	6	5	0.102526	99/100	NonOverlappingTemplate
8	11	6	12	7	11	5	16	17	7	0.080519	99/100	NonOverlappingTemplate
6	14	11	8	6	8	11	12	14	10	0.554420	99/100	NonOverlappingTemplate
11	13	12	8	8	7	13	10	13	5	0.595549	100/100	NonOverlappingTemplate
13	12	12	10	6	6	10	10	11	10	0.834308	100/100	NonOverlappingTemplate
10	7	11	4	10	11	14	9	15	9	0.437274	97/100	NonOverlappingTemplate
10	15	12	10	8	7	5	9	12	12	0.574903	99/100	NonOverlappingTemplate
15	11	7	9	12	11	6	7	10	12	0.637119	99/100	NonOverlappingTemplate
8	11	9	14	15	9	7	10	7	10	0.678686	99/100	NonOverlappingTemplate
12	10	9	10	11	10	11	6	12	9	0.971699	99/100	NonOverlappingTemplate
12	9	13	10	5	9	11	9	9	13	0.816537	98/100	NonOverlappingTemplate
8	6	11	10	7	12	13	16	8	9	0.494392	99/100	NonOverlappingTemplate
14	9	7	11	5	11	11	7	8	17	0.236810	99/100	NonOverlappingTemplate
6	18	13	8	11	9	9	9	9	8	0.334538	99/100	NonOverlappingTemplate
15	7	10	11	13	7	15	5	9	8	0.289667	98/100	NonOverlappingTemplate
12	7	10	18	11	7	9	7	6	13	0.202268	98/100	NonOverlappingTemplate
8	12	9	12	11	9	8	9	13	9	0.964295	99/100	NonOverlappingTemplate

10	10	13	9	7	9	12	12	7	11	0.924076	99/100	NonOverlappingTemplate
6	8	8	14	12	10	10	12	11	9	0.834308	100/100	NonOverlappingTemplate
16	7	7	6	16	8	13	11	6	10	0.137282	98/100	NonOverlappingTemplate
7	8	9	10	13	13	11	9	11	9	0.935716	99/100	NonOverlappingTemplate
12	14	8	8	13	8	7	10	9	11	0.816537	98/100	NonOverlappingTemplate
12	14	14	8	9	9	12	8	6	8	0.637119	99/100	NonOverlappingTemplate
13	5	10	10	7	11	9	14	10	11	0.719747	100/100	NonOverlappingTemplate
8	17	6	12	8	9	11	7	11	11	0.437274	97/100	NonOverlappingTemplate
12	9	9	13	10	10	12	4	11	10	0.779188	98/100	NonOverlappingTemplate
12	6	11	14	12	12	5	14	6	8	0.304126	99/100	NonOverlappingTemplate
7	9	10	10	11	8	11	10	7	17	0.595549	99/100	NonOverlappingTemplate
13	15	3	9	12	10	5	11	8	14	0.145326	100/100	NonOverlappingTemplate
10	3	8	10	14	15	8	9	12	11	0.319084	100/100	NonOverlappingTemplate
13	8	12	6	9	13	13	7	10	9	0.719747	98/100	NonOverlappingTemplate
8	9	7	13	9	13	5	11	11	14	0.574903	100/100	NonOverlappingTemplate
12	7	7	12	14	12	4	9	9	14	0.350485	99/100	NonOverlappingTemplate
9	13	9	11	10	12	10	6	11	9	0.946308	100/100	NonOverlappingTemplate
9	12	14	4	9	7	12	9	10	14	0.455937	99/100	NonOverlappingTemplate
10	8	10	12	11	8	8	10	12	11	0.987896	98/100	NonOverlappingTemplate
7	9	9	7	14	12	7	12	16	7	0.366918	100/100	NonOverlappingTemplate
13	8	12	15	11	4	8	11	8	10	0.455937	100/100	NonOverlappingTemplate
15	6	12	5	12	8	8	9	13	12	0.383827	100/100	NonOverlappingTemplate
11	14	8	10	11	6	14	10	9	7	0.699313	98/100	NonOverlappingTemplate
8	18	11	14	10	2	7	11	8	11	0.058984	98/100	NonOverlappingTemplate
6	9	6	12	8	12	9	14	13	11	0.616305	98/100	NonOverlappingTemplate
10	10	10	11	7	15	8	15	3	11	0.249284	99/100	NonOverlappingTemplate
10	12	12	5	10	12	7	14	9	9	0.699313	99/100	NonOverlappingTemplate
12	15	6	11	8	7	9	10	8	14	0.534146	99/100	NonOverlappingTemplate
8	9	11	16	11	8	9	12	8	8	0.739918	100/100	NonOverlappingTemplate
10	12	9	4	12	11	10	13	10	9	0.779188	99/100	NonOverlappingTemplate
14	9	10	11	6	9	13	15	6	7	0.401199	99/100	NonOverlappingTemplate
11	9	8	9	8	14	14	11	7	9	0.798139	100/100	NonOverlappingTemplate
12	16	11	10	7	10	10	7	8	9	0.699313	98/100	NonOverlappingTemplate
8	5	9	7	22	5	12	7	10	15	0.003447	100/100	NonOverlappingTemplate
10	11	12	11	14	12	10	7	6	7	0.739918	100/100	NonOverlappingTemplate
3	9	9	14	14	12	11	7	13	8	0.275709	100/100	NonOverlappingTemplate
7	16	5	7	11	11	9	12	10	12	0.437274	99/100	NonOverlappingTemplate
10	6	13	11	8	7	15	9	14	7	0.437274	99/100	NonOverlappingTemplate
7	14	9	5	8	8	14	10	9	16	0.262249	100/100	NonOverlappingTemplate
9	14	7	8	10	6	8	18	11	9	0.236810	98/100	NonOverlappingTemplate
12	9	11	3	9	15	10	11	9	11	0.494392	98/100	NonOverlappingTemplate
10	18	9	7	5	11	8	8	10	14	0.191687	99/100	NonOverlappingTemplate
3	7	7	10	15	21	7	14	9	7	0.003201	99/100	NonOverlappingTemplate
7	14	11	11	9	9	12	14	4	9	0.474986	99/100	NonOverlappingTemplate
10	8	7	8	14	10	7	12	10	14	0.719747	99/100	NonOverlappingTemplate
11	7	12	8	11	13	10	10	10	8	0.955835	98/100	NonOverlappingTemplate
14	7	6	10	17	7	10	13	8	8	0.236810	100/100	NonOverlappingTemplate
15	15	8	9	8	7	7	12	8	11	0.474986	98/100	NonOverlappingTemplate
5	12	6	6	14	12	18	5	8	14	0.025193	100/100	NonOverlappingTemplate
10	13	9	12	6	9	10	14	6	11	0.699313	100/100	NonOverlappingTemplate
11	6	13	8	14	6	13	13	11	5	0.304126	99/100	NonOverlappingTemplate
6	7	10	11	12	13	9	12	16	4	0.236810	99/100	NonOverlappingTemplate
19	8	11	11	11	8	8	7	10	7	0.249284	97/100	NonOverlappingTemplate
9	7	8	11	10	12	11	11	10	11	0.987896	98/100	NonOverlappingTemplate
10	8	8	9	11	13	6	15	10	10	0.739918	100/100	NonOverlappingTemplate
10	12	8	13	11	13	12	7	4	10	0.574903	97/100	NonOverlappingTemplate
8	5	12	13	15	11	10	9	12	5	0.366918	98/100	NonOverlappingTemplate
7	13	10	10	5	7	9	7	13	19	0.085587	100/100	NonOverlappingTemplate
5	21	14	8	10	17	5	8	6	6	0.001112	100/100	NonOverlappingTemplate

10	12	9	13	6	8	11	13	8	10	0.851383	100/100	NonOverlappingTemplate
8	8	11	14	12	15	9	4	10	9	0.419021	98/100	NonOverlappingTemplate
8	14	10	7	9	8	9	14	7	14	0.574903	99/100	NonOverlappingTemplate
14	6	8	11	8	10	17	10	9	7	0.350485	99/100	NonOverlappingTemplate
3	10	11	8	13	7	7	16	16	9	0.080519	100/100	NonOverlappingTemplate
8	4	12	12	12	11	9	8	9	15	0.494392	97/100	NonOverlappingTemplate
12	15	12	8	6	11	12	5	10	9	0.494392	100/100	NonOverlappingTemplate
10	10	9	9	13	8	13	7	8	13	0.867692	100/100	NonOverlappingTemplate
9	6	11	14	7	11	8	15	10	9	0.595549	98/100	NonOverlappingTemplate
13	7	13	8	13	8	13	10	5	10	0.554420	99/100	NonOverlappingTemplate
7	15	11	15	10	11	9	8	6	8	0.474986	100/100	NonOverlappingTemplate
15	13	14	7	5	12	9	9	7	9	0.350485	99/100	NonOverlappingTemplate
7	11	10	8	8	10	12	15	6	13	0.616305	100/100	NonOverlappingTemplate
12	5	9	7	5	15	8	11	13	15	0.171867	99/100	NonOverlappingTemplate
13	8	16	14	5	8	16	9	2	9	0.020548	99/100	NonOverlappingTemplate
13	10	6	15	4	10	13	11	11	7	0.304126	100/100	NonOverlappingTemplate
16	7	8	8	11	9	13	10	10	8	0.657933	98/100	NonOverlappingTemplate
13	12	4	11	10	11	9	12	11	7	0.678686	98/100	NonOverlappingTemplate
13	10	12	12	7	5	7	10	9	15	0.474986	98/100	NonOverlappingTemplate
11	8	13	6	9	10	7	12	15	9	0.637119	98/100	NonOverlappingTemplate
11	10	10	11	13	7	9	7	11	11	0.955835	100/100	NonOverlappingTemplate
6	5	7	13	12	11	8	12	15	11	0.366918	100/100	NonOverlappingTemplate
13	9	10	12	8	7	9	11	7	14	0.798139	100/100	NonOverlappingTemplate
14	11	9	12	7	11	12	8	7	9	0.834308	99/100	NonOverlappingTemplate
13	14	10	8	10	7	9	11	10	8	0.883171	98/100	NonOverlappingTemplate
13	6	10	8	13	10	4	15	12	9	0.319084	96/100	NonOverlappingTemplate
5	12	10	7	10	11	10	20	4	11	0.040108	99/100	NonOverlappingTemplate
7	11	14	13	10	6	14	8	11	6	0.455937	100/100	NonOverlappingTemplate
8	12	9	8	16	13	9	11	8	6	0.534146	99/100	NonOverlappingTemplate
11	11	15	9	7	9	6	9	8	15	0.494392	100/100	NonOverlappingTemplate
5	6	9	16	9	13	13	10	12	7	0.275709	100/100	NonOverlappingTemplate
6	7	13	12	6	12	11	9	10	14	0.574903	100/100	NonOverlappingTemplate
13	12	13	4	11	14	11	8	7	7	0.366918	98/100	NonOverlappingTemplate
10	13	8	9	9	12	8	10	9	12	0.971699	98/100	NonOverlappingTemplate
7	10	9	10	12	9	10	15	5	13	0.595549	100/100	NonOverlappingTemplate
13	15	12	6	7	11	14	6	11	5	0.202268	100/100	NonOverlappingTemplate
11	11	8	7	10	12	10	9	11	11	0.987896	97/100	NonOverlappingTemplate
10	11	4	10	8	11	11	3	15	17	0.055361	99/100	NonOverlappingTemplate
12	9	6	10	14	14	4	9	12	10	0.401199	100/100	NonOverlappingTemplate
6	9	6	12	8	12	9	14	13	11	0.616305	98/100	NonOverlappingTemplate
9	2	15	15	8	12	11	12	11	5	0.080519	99/100	OverlappingTemplate
8	12	12	11	12	9	6	9	11	10	0.935716	97/100	Universal
14	11	9	5	11	11	8	12	11	8	0.759756	98/100	ApproximateEntropy
9	5	9	4	4	6	3	4	6	11	0.287306	60/61	RandomExcursions
6	5	5	3	6	10	4	7	7	8	0.723129	61/61	RandomExcursions
8	6	3	7	11	3	8	7	4	4	0.337162	59/61	RandomExcursions
10	9	4	8	8	5	6	3	5	3	0.392456	59/61	RandomExcursions
7	5	6	6	6	3	8	5	7	8	0.941144	58/61	RandomExcursions
7	7	2	5	6	5	9	6	8	6	0.788728	61/61	RandomExcursions
8	6	2	9	9	1	9	6	6	5	0.204076	61/61	RandomExcursions
7	4	4	12	6	10	4	4	2	8	0.095617	61/61	RandomExcursions
6	2	6	11	6	3	5	7	8	7	0.392456	59/61	RandomExcursionsVariant
4	6	5	5	6	8	4	6	12	5	0.484646	59/61	RandomExcursionsVariant
7	3	6	2	11	7	6	4	7	8	0.337162	59/61	RandomExcursionsVariant
7	5	4	8	9	6	5	6	4	7	0.900104	59/61	RandomExcursionsVariant
7	6	4	8	8	8	5	7	3	5	0.848588	59/61	RandomExcursionsVariant
9	2	8	5	4	2	11	6	6	8	0.155209	60/61	RandomExcursionsVariant
8	5	3	4	5	5	5	13	5	8	0.186566	60/61	RandomExcursionsVariant
8	4	9	4	5	3	3	11	8	6	0.242986	59/61	RandomExcursionsVariant

5	6	8	5	7	9	4	4	8	5	0.848588	58/61	RandomExcursionsVariant
6	6	7	8	5	6	5	5	4	9	0.941144	61/61	RandomExcursionsVariant
8	5	9	4	5	4	9	4	4	9	0.517442	61/61	RandomExcursionsVariant
9	8	4	3	5	5	9	5	9	4	0.484646	60/61	RandomExcursionsVariant
7	4	11	3	3	7	10	6	7	3	0.186566	61/61	RandomExcursionsVariant
9	0	4	7	6	7	9	8	7	4	0.242986	61/61	RandomExcursionsVariant
8	5	3	5	5	10	6	7	6	6	0.788728	61/61	RandomExcursionsVariant
6	6	8	6	5	7	2	14	1	6	0.029796	61/61	RandomExcursionsVariant
6	8	6	5	5	4	8	3	6	10	0.689019	61/61	RandomExcursionsVariant
6	8	5	5	7	2	11	5	6	6	0.517442	61/61	RandomExcursionsVariant
12	8	9	11	9	10	7	14	5	15	0.474986	99/100	Serial
9	12	10	8	8	6	15	9	13	10	0.699313	100/100	Serial
9	9	15	11	7	13	7	7	10	12	0.657933	100/100	LinearComplexity

The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 58 for a sample size = 61 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.

Hybrid ChaCha20 + RO-TRNG (100 Bitstream)

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <data/final_data.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
11	9	8	9	10	17	10	5	12	9	0.474986	99/100	Frequency
9	9	14	5	14	12	6	11	10	10	0.534146	99/100	BlockFrequency
7	13	6	3	12	20	9	9	10	11	0.025193	99/100	CumulativeSums
11	14	7	6	15	8	15	6	9	9	0.249284	100/100	CumulativeSums
15	9	18	9	8	8	7	11	7	8	0.202268	99/100	Runs
8	13	4	13	9	8	5	14	13	13	0.202268	98/100	LongestRun
7	8	17	10	13	5	10	12	10	8	0.319084	99/100	Rank
8	7	17	13	8	8	9	9	15	6	0.202268	99/100	FFT
14	6	12	7	13	10	10	9	11	8	0.739918	99/100	NonOverlappingTemplate
11	10	6	15	11	11	9	11	7	9	0.779188	96/100	NonOverlappingTemplate
7	10	8	11	10	15	7	10	11	11	0.834308	100/100	NonOverlappingTemplate
13	7	12	11	9	6	13	7	9	13	0.657933	97/100	NonOverlappingTemplate
10	9	7	10	18	10	7	5	11	13	0.224821	99/100	NonOverlappingTemplate
12	3	10	10	15	4	9	16	11	10	0.085587	98/100	NonOverlappingTemplate
9	13	17	7	10	8	9	13	9	5	0.289667	99/100	NonOverlappingTemplate
9	12	11	9	12	10	9	8	7	13	0.946308	99/100	NonOverlappingTemplate
8	15	12	17	5	13	7	6	8	9	0.102526	100/100	NonOverlappingTemplate
9	7	10	8	8	6	14	14	14	10	0.514124	100/100	NonOverlappingTemplate
9	11	15	12	7	7	10	9	12	8	0.759756	99/100	NonOverlappingTemplate
11	5	11	9	6	11	15	11	11	10	0.616305	98/100	NonOverlappingTemplate
8	9	8	6	13	14	9	14	6	13	0.419021	100/100	NonOverlappingTemplate

5	8	11	13	10	11	12	8	15	7	0.514124	100/100	NonOverlappingTemplate
7	12	7	11	10	18	9	10	10	6	0.319084	100/100	NonOverlappingTemplate
17	13	10	5	6	7	8	8	17	9	0.055361	99/100	NonOverlappingTemplate
9	7	12	4	6	15	15	13	8	11	0.162606	100/100	NonOverlappingTemplate
5	9	12	12	13	12	8	9	11	9	0.798139	100/100	NonOverlappingTemplate
14	9	8	8	10	11	11	11	10	8	0.955835	98/100	NonOverlappingTemplate
6	11	12	13	8	5	12	6	12	15	0.289667	100/100	NonOverlappingTemplate
12	9	14	5	8	10	11	11	9	11	0.798139	98/100	NonOverlappingTemplate
9	14	5	10	10	14	11	6	8	13	0.455937	100/100	NonOverlappingTemplate
8	8	8	15	10	17	7	3	11	13	0.080519	100/100	NonOverlappingTemplate
7	11	8	12	3	12	15	13	9	10	0.304126	99/100	NonOverlappingTemplate
15	8	7	13	8	10	9	9	12	9	0.759756	98/100	NonOverlappingTemplate
9	8	13	6	14	9	11	12	14	4	0.319084	99/100	NonOverlappingTemplate
14	12	10	8	9	7	13	11	8	8	0.816537	99/100	NonOverlappingTemplate
11	12	13	11	14	11	8	5	8	7	0.595549	99/100	NonOverlappingTemplate
10	6	13	8	9	6	16	9	11	12	0.455937	97/100	NonOverlappingTemplate
14	15	13	9	14	11	7	3	8	6	0.102526	97/100	NonOverlappingTemplate
10	5	10	10	13	7	8	8	16	13	0.383827	99/100	NonOverlappingTemplate
6	8	15	4	18	5	16	7	12	9	0.008879	100/100	NonOverlappingTemplate
16	10	7	7	9	13	12	8	8	10	0.574903	99/100	NonOverlappingTemplate
10	6	16	3	9	8	11	12	12	13	0.191687	99/100	NonOverlappingTemplate
8	14	12	6	12	7	8	13	11	9	0.657933	100/100	NonOverlappingTemplate
10	14	9	8	5	12	11	10	14	7	0.574903	100/100	NonOverlappingTemplate
11	9	9	8	14	12	12	9	11	5	0.759756	100/100	NonOverlappingTemplate
20	13	6	5	9	5	11	11	7	13	0.020548	99/100	NonOverlappingTemplate
5	7	11	9	8	18	9	9	13	11	0.236810	99/100	NonOverlappingTemplate
11	6	5	8	8	11	14	10	14	13	0.419021	99/100	NonOverlappingTemplate
12	9	14	11	11	10	10	8	8	7	0.911413	98/100	NonOverlappingTemplate
9	13	13	8	12	12	13	9	3	8	0.401199	96/100	NonOverlappingTemplate
6	13	12	10	10	6	7	13	12	11	0.657933	98/100	NonOverlappingTemplate
14	8	7	14	8	11	10	12	7	9	0.699313	99/100	NonOverlappingTemplate
6	18	15	8	12	7	8	7	12	7	0.096578	99/100	NonOverlappingTemplate
8	9	7	9	9	7	13	11	14	13	0.739918	100/100	NonOverlappingTemplate
7	11	12	13	10	9	12	12	9	5	0.759756	99/100	NonOverlappingTemplate
11	9	7	12	10	5	13	7	14	12	0.554420	99/100	NonOverlappingTemplate
11	13	10	10	14	7	8	4	10	13	0.494392	96/100	NonOverlappingTemplate
11	9	12	16	9	8	12	6	10	7	0.574903	99/100	NonOverlappingTemplate
14	17	11	9	8	10	7	10	8	6	0.350485	99/100	NonOverlappingTemplate
20	12	7	11	7	6	9	9	8	11	0.102526	98/100	NonOverlappingTemplate
4	13	16	7	14	9	9	7	10	11	0.224821	99/100	NonOverlappingTemplate
14	4	8	7	7	13	16	7	8	16	0.051942	99/100	NonOverlappingTemplate
4	9	13	12	9	18	10	6	11	8	0.137282	99/100	NonOverlappingTemplate
10	5	12	12	9	9	8	9	17	9	0.437274	100/100	NonOverlappingTemplate
10	4	13	7	15	14	12	10	7	8	0.262249	98/100	NonOverlappingTemplate
8	10	12	13	7	15	9	7	9	10	0.719747	98/100	NonOverlappingTemplate
10	8	8	10	10	9	15	8	9	13	0.851383	100/100	NonOverlappingTemplate
10	10	8	6	8	8	9	16	13	12	0.554420	98/100	NonOverlappingTemplate
15	10	9	10	9	10	11	11	8	7	0.897763	100/100	NonOverlappingTemplate
7	12	9	8	9	11	7	7	15	15	0.455937	99/100	NonOverlappingTemplate
9	12	13	15	12	12	7	7	9	4	0.334538	97/100	NonOverlappingTemplate
12	18	5	5	5	9	6	10	17	13	0.009535	98/100	NonOverlappingTemplate
10	4	9	13	14	12	13	7	10	8	0.455937	100/100	NonOverlappingTemplate
6	8	16	11	8	10	6	12	15	8	0.275709	100/100	NonOverlappingTemplate
6	10	10	10	9	9	12	13	9	12	0.935716	99/100	NonOverlappingTemplate
6	14	11	8	10	10	7	8	13	13	0.657933	100/100	NonOverlappingTemplate
17	10	8	12	12	7	12	6	10	6	0.304126	100/100	NonOverlappingTemplate
13	17	8	11	6	7	11	7	6	14	0.162606	96/100	NonOverlappingTemplate
7	14	9	11	11	13	8	10	8	9	0.867692	100/100	NonOverlappingTemplate
11	12	11	11	1	11	6	9	11	17	0.075719	99/100	NonOverlappingTemplate

8	9	15	10	6	8	8	17	9	10	0.319084	100/100	NonOverlappingTemplate
14	9	8	10	13	7	13	10	7	9	0.759756	99/100	NonOverlappingTemplate
14	6	12	7	13	10	10	10	10	8	0.759756	99/100	NonOverlappingTemplate
13	9	11	10	11	9	10	8	12	7	0.964295	97/100	NonOverlappingTemplate
9	6	14	9	7	14	10	9	11	11	0.719747	99/100	NonOverlappingTemplate
11	9	10	9	16	10	13	10	5	7	0.514124	99/100	NonOverlappingTemplate
12	15	6	7	8	15	12	7	9	9	0.366918	100/100	NonOverlappingTemplate
6	5	9	13	13	10	12	10	14	8	0.494392	100/100	NonOverlappingTemplate
14	11	7	8	10	10	12	7	7	14	0.657933	100/100	NonOverlappingTemplate
7	9	15	12	12	9	9	6	12	9	0.678686	99/100	NonOverlappingTemplate
5	11	8	11	7	15	10	11	12	10	0.637119	99/100	NonOverlappingTemplate
7	6	14	10	5	12	15	12	7	12	0.262249	100/100	NonOverlappingTemplate
8	12	6	13	3	19	14	9	8	8	0.026948	100/100	NonOverlappingTemplate
12	5	12	5	13	12	9	11	11	10	0.595549	98/100	NonOverlappingTemplate
8	10	7	10	10	16	10	12	10	7	0.719747	99/100	NonOverlappingTemplate
10	8	6	12	12	11	12	7	12	10	0.867692	100/100	NonOverlappingTemplate
5	9	9	13	10	11	8	10	12	13	0.798139	100/100	NonOverlappingTemplate
12	7	8	10	13	10	11	7	12	10	0.911413	97/100	NonOverlappingTemplate
7	8	11	14	7	7	11	7	16	12	0.366918	100/100	NonOverlappingTemplate
9	15	14	5	10	5	6	14	10	12	0.171867	100/100	NonOverlappingTemplate
7	9	9	14	8	10	9	10	12	12	0.911413	99/100	NonOverlappingTemplate
11	10	8	8	8	13	6	9	13	14	0.699313	99/100	NonOverlappingTemplate
14	11	17	8	11	10	5	7	9	8	0.275709	100/100	NonOverlappingTemplate
11	13	6	6	12	11	11	9	6	15	0.437274	98/100	NonOverlappingTemplate
10	8	11	12	11	10	12	12	8	6	0.924076	100/100	NonOverlappingTemplate
10	8	7	13	16	9	10	8	8	11	0.657933	98/100	NonOverlappingTemplate
10	9	8	13	14	7	11	11	10	7	0.834308	99/100	NonOverlappingTemplate
12	8	9	11	14	11	11	6	9	9	0.867692	98/100	NonOverlappingTemplate
11	12	11	7	15	10	9	11	7	7	0.739918	97/100	NonOverlappingTemplate
11	11	8	16	8	9	3	12	13	9	0.275709	100/100	NonOverlappingTemplate
10	14	8	8	14	5	7	10	14	10	0.437274	100/100	NonOverlappingTemplate
14	9	14	9	11	7	9	9	10	8	0.834308	98/100	NonOverlappingTemplate
10	15	3	11	11	8	10	12	6	14	0.236810	99/100	NonOverlappingTemplate
5	9	6	9	12	10	12	10	13	14	0.574903	100/100	NonOverlappingTemplate
9	12	13	8	12	7	7	10	14	8	0.739918	99/100	NonOverlappingTemplate
9	10	7	5	15	14	14	7	7	12	0.249284	100/100	NonOverlappingTemplate
12	9	9	9	10	9	10	7	14	11	0.946308	100/100	NonOverlappingTemplate
9	9	8	12	6	12	7	12	8	17	0.383827	100/100	NonOverlappingTemplate
8	9	6	15	12	9	8	8	11	14	0.574903	99/100	NonOverlappingTemplate
11	8	3	8	8	13	8	13	10	18	0.096578	100/100	NonOverlappingTemplate
10	7	6	13	7	9	9	14	13	12	0.595549	100/100	NonOverlappingTemplate
8	15	10	5	8	16	13	10	3	12	0.075719	100/100	NonOverlappingTemplate
9	7	13	6	9	13	6	14	10	13	0.474986	99/100	NonOverlappingTemplate
14	11	9	10	6	9	11	6	9	15	0.554420	99/100	NonOverlappingTemplate
8	18	13	10	12	8	2	11	10	8	0.080519	100/100	NonOverlappingTemplate
11	9	9	7	13	15	6	15	6	9	0.319084	100/100	NonOverlappingTemplate
6	16	11	10	11	4	12	10	11	9	0.383827	99/100	NonOverlappingTemplate
13	8	10	6	11	6	14	10	11	11	0.699313	98/100	NonOverlappingTemplate
9	9	6	14	13	12	6	11	5	15	0.249284	99/100	NonOverlappingTemplate
8	10	12	15	9	7	12	9	9	9	0.834308	100/100	NonOverlappingTemplate
15	11	6	10	5	8	8	8	19	10	0.066882	98/100	NonOverlappingTemplate
13	8	10	9	9	7	15	6	11	12	0.637119	98/100	NonOverlappingTemplate
5	11	10	12	8	13	11	9	12	9	0.834308	99/100	NonOverlappingTemplate
10	10	14	10	11	10	10	8	11	6	0.924076	98/100	NonOverlappingTemplate
11	7	7	11	9	8	15	9	15	8	0.534146	98/100	NonOverlappingTemplate
7	9	11	9	7	8	17	16	9	7	0.213309	99/100	NonOverlappingTemplate
15	6	6	8	11	14	11	16	10	3	0.058984	98/100	NonOverlappingTemplate
8	9	11	10	12	9	9	13	10	9	0.987896	98/100	NonOverlappingTemplate
11	10	11	13	9	8	11	13	6	8	0.867692	97/100	NonOverlappingTemplate

11	11	14	7	8	8	9	6	13	13	0.637119	99/100	NonOverlappingTemplate
12	6	6	7	12	8	12	12	14	11	0.554420	100/100	NonOverlappingTemplate
11	12	9	8	9	11	14	7	9	10	0.924076	97/100	NonOverlappingTemplate
10	10	11	7	10	6	15	8	15	8	0.494392	98/100	NonOverlappingTemplate
8	6	11	15	13	12	5	11	13	6	0.275709	99/100	NonOverlappingTemplate
14	13	12	8	7	7	8	12	7	12	0.616305	97/100	NonOverlappingTemplate
5	7	10	7	11	12	10	13	11	14	0.595549	100/100	NonOverlappingTemplate
10	13	10	7	11	11	10	7	10	11	0.964295	99/100	NonOverlappingTemplate
7	6	16	7	10	11	10	10	15	8	0.350485	99/100	NonOverlappingTemplate
14	10	14	12	5	11	10	5	16	3	0.045675	97/100	NonOverlappingTemplate
12	9	14	7	11	2	12	17	8	8	0.075719	98/100	NonOverlappingTemplate
12	8	11	5	11	12	13	12	8	8	0.739918	98/100	NonOverlappingTemplate
18	15	3	7	12	8	12	7	10	8	0.045675	100/100	NonOverlappingTemplate
15	8	11	12	8	11	9	10	9	7	0.834308	99/100	NonOverlappingTemplate
5	14	9	14	11	8	9	10	8	12	0.616305	100/100	NonOverlappingTemplate
9	14	12	9	7	12	10	11	9	7	0.867692	100/100	NonOverlappingTemplate
14	9	8	10	13	8	12	10	7	9	0.851383	99/100	NonOverlappingTemplate
11	7	12	12	8	9	12	9	12	8	0.935716	99/100	OverlappingTemplate
10	5	5	11	17	12	9	8	9	14	0.181557	99/100	Universal
7	10	14	7	11	8	14	13	5	11	0.437274	99/100	ApproximateEntropy
7	4	6	11	5	5	7	5	6	5	0.756476	61/61	RandomExcursions
13	8	8	6	4	4	7	6	5	0	0.046169	58/61	RandomExcursions
12	5	6	8	6	10	2	1	6	5	0.057146	58/61	RandomExcursions
9	8	7	4	3	7	7	7	6	3	0.689019	61/61	RandomExcursions
6	5	6	8	6	3	7	6	6	8	0.957319	60/61	RandomExcursions
4	9	5	6	4	4	6	4	9	10	0.484646	60/61	RandomExcursions
2	7	6	5	2	9	6	6	10	8	0.311542	61/61	RandomExcursions
8	4	6	5	5	3	8	6	6	10	0.689019	61/61	RandomExcursions
7	10	7	3	9	5	4	5	6	5	0.619772	60/61	RandomExcursionsVariant
7	8	9	9	4	2	7	6	4	5	0.517442	60/61	RandomExcursionsVariant
8	12	7	4	6	6	4	5	7	2	0.264458	61/61	RandomExcursionsVariant
10	4	6	6	12	3	4	7	4	5	0.186566	61/61	RandomExcursionsVariant
8	8	4	8	4	5	14	2	6	2	0.021262	61/61	RandomExcursionsVariant
9	9	9	0	8	6	9	2	6	3	0.051391	60/61	RandomExcursionsVariant
10	10	5	2	4	5	5	9	5	6	0.287306	59/61	RandomExcursionsVariant
7	8	6	8	3	9	8	3	6	3	0.517442	60/61	RandomExcursionsVariant
9	6	8	9	5	6	3	5	4	6	0.723129	60/61	RandomExcursionsVariant
7	5	3	11	4	8	5	2	11	5	0.105618	60/61	RandomExcursionsVariant
4	6	7	4	12	8	6	4	2	8	0.204076	60/61	RandomExcursionsVariant
5	6	3	6	11	6	7	3	7	7	0.551026	60/61	RandomExcursionsVariant
8	5	5	5	9	6	6	6	5	6	0.970538	59/61	RandomExcursionsVariant
7	7	6	4	8	3	7	5	9	5	0.819544	60/61	RandomExcursionsVariant
8	5	6	3	2	5	4	14	6	8	0.046169	60/61	RandomExcursionsVariant
8	6	3	3	5	5	6	8	9	8	0.654467	60/61	RandomExcursionsVariant
7	6	4	3	7	6	5	10	8	5	0.723129	60/61	RandomExcursionsVariant
6	5	4	6	6	3	6	6	12	7	0.484646	60/61	RandomExcursionsVariant
9	12	6	8	13	9	10	11	10	12	0.911413	100/100	Serial
7	12	13	10	4	10	15	6	9	14	0.236810	100/100	Serial
7	7	10	15	8	9	8	12	14	10	0.616305	99/100	LinearComplexity

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 96 for a sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately = 58 for a sample size = 61 binary sequences.

For further guidelines construct a probability table using the MAPLE program provided in the addendum section of the documentation.

Hybrid ChaCha20 + RO-TRNG (1000 Bitstream)

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <data/data_final_1000.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
106	95	105	97	99	112	108	89	106	83	0.585209	987/1000	Frequency
90	98	101	101	100	120	103	99	96	92	0.743915	988/1000	BlockFrequency
97	113	106	92	107	87	99	96	99	104	0.807412	990/1000	CumulativeSums
111	101	85	97	95	110	94	99	113	95	0.624627	985/1000	CumulativeSums
103	98	119	82	105	106	97	91	90	109	0.326749	987/1000	Runs
87	115	70	93	105	111	104	116	100	99	0.039847	985/1000	LongestRun
94	98	105	99	102	94	104	106	103	95	0.992670	995/1000	Rank
109	99	102	119	78	92	100	104	117	80	0.048716	992/1000	FFT
97	84	100	95	97	101	101	111	116	98	0.655854	991/1000	NonOverlappingTemplate
118	108	80	94	101	80	107	112	86	114	0.034031	985/1000	NonOverlappingTemplate
92	88	95	110	92	122	105	97	94	105	0.370262	993/1000	NonOverlappingTemplate
106	110	97	93	106	86	106	83	106	107	0.498313	987/1000	NonOverlappingTemplate
104	106	98	89	105	73	88	109	126	102	0.031219	987/1000	NonOverlappingTemplate
89	98	112	86	124	101	95	97	97	101	0.285427	989/1000	NonOverlappingTemplate
99	111	115	88	101	103	88	96	107	92	0.560545	988/1000	NonOverlappingTemplate
84	103	91	101	101	103	105	105	96	111	0.794391	991/1000	NonOverlappingTemplate
94	116	98	108	86	110	86	103	101	98	0.469232	988/1000	NonOverlappingTemplate
78	99	108	106	88	104	112	99	116	90	0.188601	990/1000	NonOverlappingTemplate
96	112	114	101	95	89	101	88	97	107	0.630872	989/1000	NonOverlappingTemplate
100	106	106	94	98	83	113	102	102	96	0.745908	991/1000	NonOverlappingTemplate
106	86	96	88	108	104	87	130	71	124	0.000517	990/1000	NonOverlappingTemplate
114	99	99	91	101	104	100	88	104	100	0.870856	990/1000	NonOverlappingTemplate
102	110	74	97	108	94	109	117	103	86	0.101311	995/1000	NonOverlappingTemplate
104	92	106	90	104	93	98	89	122	102	0.442831	989/1000	NonOverlappingTemplate
90	110	98	93	96	105	103	118	90	97	0.599693	992/1000	NonOverlappingTemplate
106	104	109	102	112	103	102	93	100	69	0.160805	990/1000	NonOverlappingTemplate
106	107	104	96	90	98	94	108	109	88	0.792508	984/1000	NonOverlappingTemplate
104	90	118	107	96	83	86	87	105	124	0.040108	995/1000	NonOverlappingTemplate
113	89	91	95	89	110	87	109	114	103	0.295391	990/1000	NonOverlappingTemplate
99	102	98	92	112	104	105	107	95	86	0.809249	991/1000	NonOverlappingTemplate
91	104	87	114	104	92	85	112	96	115	0.241741	997/1000	NonOverlappingTemplate
97	105	106	102	91	98	114	87	94	106	0.743915	989/1000	NonOverlappingTemplate
108	90	89	98	99	117	102	98	105	94	0.691081	990/1000	NonOverlappingTemplate
94	94	100	98	104	90	92	119	106	103	0.676615	986/1000	NonOverlappingTemplate
108	87	99	89	103	99	101	106	89	119	0.452173	994/1000	NonOverlappingTemplate
103	89	109	95	100	104	112	91	90	107	0.713641	991/1000	NonOverlappingTemplate
103	105	86	94	102	88	116	113	88	105	0.344048	988/1000	NonOverlappingTemplate
90	103	102	94	100	113	98	96	102	102	0.943242	992/1000	NonOverlappingTemplate
99	89	101	103	110	108	85	91	116	98	0.473064	981/1000	NonOverlappingTemplate
79	92	116	82	119	94	108	103	108	99	0.062821	995/1000	NonOverlappingTemplate
122	98	94	98	120	106	89	104	89	80	0.062427	994/1000	NonOverlappingTemplate
100	94	122	102	112	94	93	81	97	105	0.244236	986/1000	NonOverlappingTemplate
82	108	85	109	120	91	98	111	97	99	0.158133	994/1000	NonOverlappingTemplate
108	90	115	103	98	86	90	111	85	114	0.191687	989/1000	NonOverlappingTemplate

92	90	91	98	120	121	102	98	97	91	0.220159	995/1000	NonOverlappingTemplate
109	91	82	113	115	95	89	93	102	111	0.213309	989/1000	NonOverlappingTemplate
106	109	99	98	94	101	97	98	90	108	0.948298	990/1000	NonOverlappingTemplate
108	105	95	115	83	115	103	94	92	90	0.302657	987/1000	NonOverlappingTemplate
110	100	91	108	100	106	101	90	98	96	0.910091	990/1000	NonOverlappingTemplate
88	108	102	110	97	101	112	81	99	102	0.502247	989/1000	NonOverlappingTemplate
96	102	114	99	92	101	99	91	95	111	0.825505	993/1000	NonOverlappingTemplate
94	103	106	107	99	118	94	90	99	90	0.645448	983/1000	NonOverlappingTemplate
93	103	99	106	98	94	113	94	116	84	0.482707	996/1000	NonOverlappingTemplate
87	102	113	96	81	95	107	106	103	110	0.402962	993/1000	NonOverlappingTemplate
107	100	106	118	96	103	100	104	80	86	0.314544	989/1000	NonOverlappingTemplate
106	95	85	112	105	95	119	91	95	97	0.387264	990/1000	NonOverlappingTemplate
96	105	112	97	116	97	80	84	102	111	0.191687	988/1000	NonOverlappingTemplate
92	107	95	112	91	91	111	113	95	93	0.526105	992/1000	NonOverlappingTemplate
101	105	92	116	97	104	102	98	101	84	0.703417	991/1000	NonOverlappingTemplate
106	108	96	93	92	94	94	117	101	99	0.747898	987/1000	NonOverlappingTemplate
93	93	103	103	107	98	94	123	92	94	0.500279	994/1000	NonOverlappingTemplate
94	108	101	105	95	104	88	88	109	108	0.739918	994/1000	NonOverlappingTemplate
94	103	108	133	89	103	99	90	95	86	0.057146	987/1000	NonOverlappingTemplate
91	82	109	100	106	97	103	103	110	99	0.689019	990/1000	NonOverlappingTemplate
102	101	102	92	122	97	87	98	90	109	0.419021	990/1000	NonOverlappingTemplate
94	93	99	111	105	104	111	81	98	104	0.585209	991/1000	NonOverlappingTemplate
119	96	82	102	103	119	97	79	93	110	0.052947	992/1000	NonOverlappingTemplate
110	87	103	100	76	106	100	108	121	89	0.086634	985/1000	NonOverlappingTemplate
100	92	84	106	100	98	114	117	111	78	0.105618	990/1000	NonOverlappingTemplate
96	106	78	97	97	103	96	103	107	117	0.413628	990/1000	NonOverlappingTemplate
102	91	101	108	106	80	107	97	110	98	0.587274	990/1000	NonOverlappingTemplate
107	95	105	90	87	90	101	104	101	120	0.450297	990/1000	NonOverlappingTemplate
89	95	104	100	113	97	108	100	85	109	0.626709	993/1000	NonOverlappingTemplate
103	103	108	113	111	88	97	86	100	91	0.532132	988/1000	NonOverlappingTemplate
110	92	108	94	92	108	97	107	87	105	0.695200	988/1000	NonOverlappingTemplate
86	100	110	105	92	98	97	107	102	103	0.867692	991/1000	NonOverlappingTemplate
97	95	105	107	100	122	88	88	94	104	0.408275	991/1000	NonOverlappingTemplate
114	108	100	110	103	94	87	93	86	105	0.490483	986/1000	NonOverlappingTemplate
112	99	113	75	103	109	97	97	93	102	0.275709	990/1000	NonOverlappingTemplate
103	109	94	96	103	98	98	108	99	92	0.968863	998/1000	NonOverlappingTemplate
90	92	136	110	79	110	87	111	98	87	0.002339	989/1000	NonOverlappingTemplate
95	97	110	80	106	99	93	99	112	109	0.488534	994/1000	NonOverlappingTemplate
97	84	99	96	96	102	102	111	115	98	0.682823	991/1000	NonOverlappingTemplate
121	98	96	111	104	102	92	98	91	87	0.419021	987/1000	NonOverlappingTemplate
95	98	93	91	94	111	107	97	100	114	0.769527	991/1000	NonOverlappingTemplate
102	103	98	104	98	105	99	103	84	104	0.944274	990/1000	NonOverlappingTemplate
107	90	90	99	89	104	105	112	95	109	0.676615	992/1000	NonOverlappingTemplate
96	76	108	106	99	106	99	102	108	100	0.536163	993/1000	NonOverlappingTemplate
90	105	88	95	98	93	113	112	99	107	0.626709	995/1000	NonOverlappingTemplate
80	113	108	110	105	88	98	113	97	88	0.197981	993/1000	NonOverlappingTemplate
101	114	98	103	84	102	86	99	117	96	0.373625	993/1000	NonOverlappingTemplate
93	85	102	100	97	113	108	99	103	100	0.807412	995/1000	NonOverlappingTemplate
108	103	104	114	97	110	95	87	86	96	0.534146	987/1000	NonOverlappingTemplate
89	103	95	90	115	99	89	108	102	110	0.564639	994/1000	NonOverlappingTemplate
97	102	113	98	98	104	104	96	97	91	0.952152	994/1000	NonOverlappingTemplate
105	104	99	99	106	107	95	95	98	92	0.981940	988/1000	NonOverlappingTemplate
101	90	97	98	90	91	104	119	96	114	0.452173	986/1000	NonOverlappingTemplate
89	103	109	91	101	87	109	113	97	101	0.614226	988/1000	NonOverlappingTemplate
89	99	104	115	92	93	104	111	93	100	0.676615	994/1000	NonOverlappingTemplate
111	102	108	86	101	85	95	96	113	103	0.504219	984/1000	NonOverlappingTemplate
100	112	85	95	123	101	81	108	92	103	0.114712	994/1000	NonOverlappingTemplate
105	91	88	96	108	98	93	107	113	101	0.737915	988/1000	NonOverlappingTemplate
108	94	119	103	98	95	103	96	100	84	0.554420	990/1000	NonOverlappingTemplate

93	97	87	101	97	111	113	94	97	110	0.666245	986/1000	NonOverlappingTemplate
105	90	94	101	95	98	106	116	105	90	0.731886	985/1000	NonOverlappingTemplate
83	116	94	99	100	121	91	103	93	100	0.235589	992/1000	NonOverlappingTemplate
88	101	90	104	104	98	104	107	112	92	0.784927	996/1000	NonOverlappingTemplate
95	104	101	89	88	110	106	96	100	111	0.759756	991/1000	NonOverlappingTemplate
110	125	105	97	109	95	94	78	96	91	0.096000	982/1000	NonOverlappingTemplate
99	101	68	108	103	110	97	104	105	105	0.174728	990/1000	NonOverlappingTemplate
97	91	95	94	93	105	102	114	106	103	0.859637	984/1000	NonOverlappingTemplate
93	85	100	102	106	100	100	108	108	98	0.878618	988/1000	NonOverlappingTemplate
97	108	81	101	102	94	100	97	106	114	0.620465	990/1000	NonOverlappingTemplate
96	94	86	101	97	112	103	101	106	104	0.864494	991/1000	NonOverlappingTemplate
106	103	98	89	123	95	96	91	102	97	0.500279	992/1000	NonOverlappingTemplate
90	103	98	101	99	119	96	106	93	95	0.737915	994/1000	NonOverlappingTemplate
109	84	104	101	103	90	99	105	89	116	0.469232	993/1000	NonOverlappingTemplate
105	91	111	97	96	118	105	87	81	109	0.206629	992/1000	NonOverlappingTemplate
96	98	92	102	106	94	96	100	99	117	0.862883	990/1000	NonOverlappingTemplate
73	101	104	109	97	103	96	98	104	115	0.271619	996/1000	NonOverlappingTemplate
102	110	99	94	98	81	94	115	106	101	0.530120	987/1000	NonOverlappingTemplate
101	102	105	95	102	101	105	96	77	116	0.450297	992/1000	NonOverlappingTemplate
99	100	100	102	95	84	88	123	85	124	0.040108	991/1000	NonOverlappingTemplate
94	90	92	104	108	109	115	102	89	97	0.616305	988/1000	NonOverlappingTemplate
96	116	101	104	105	95	92	89	107	95	0.741918	989/1000	NonOverlappingTemplate
102	111	88	89	100	111	105	102	97	95	0.765632	994/1000	NonOverlappingTemplate
84	112	100	95	98	88	100	99	118	106	0.406499	992/1000	NonOverlappingTemplate
96	95	83	92	111	109	110	115	100	89	0.317565	987/1000	NonOverlappingTemplate
102	84	115	101	96	112	90	102	90	108	0.424453	991/1000	NonOverlappingTemplate
89	97	99	121	97	101	86	104	116	90	0.242986	994/1000	NonOverlappingTemplate
102	102	95	105	104	104	89	81	116	102	0.502247	983/1000	NonOverlappingTemplate
87	91	107	108	102	102	104	101	99	99	0.917870	991/1000	NonOverlappingTemplate
94	95	98	97	116	89	94	86	129	102	0.083526	987/1000	NonOverlappingTemplate
105	114	85	106	91	95	97	96	125	86	0.110734	985/1000	NonOverlappingTemplate
106	104	98	98	109	86	92	98	113	96	0.749884	989/1000	NonOverlappingTemplate
102	102	106	89	86	104	107	109	113	82	0.350485	988/1000	NonOverlappingTemplate
104	87	77	96	98	114	112	109	87	116	0.071177	990/1000	NonOverlappingTemplate
116	93	100	101	96	93	93	92	108	108	0.727851	992/1000	NonOverlappingTemplate
104	90	118	100	107	90	104	93	100	94	0.647530	988/1000	NonOverlappingTemplate
93	98	129	93	103	93	99	92	94	106	0.250558	995/1000	NonOverlappingTemplate
99	89	100	89	99	87	102	111	119	105	0.415422	997/1000	NonOverlappingTemplate
96	100	115	98	98	89	110	98	103	93	0.805569	989/1000	NonOverlappingTemplate
89	103	104	90	102	98	92	95	106	121	0.514124	991/1000	NonOverlappingTemplate
105	93	104	108	101	102	101	95	99	92	0.980883	989/1000	NonOverlappingTemplate
102	112	104	78	112	89	107	90	102	104	0.288249	993/1000	NonOverlappingTemplate
83	90	87	91	106	97	108	111	111	116	0.188601	997/1000	NonOverlappingTemplate
104	103	91	98	104	100	101	100	84	115	0.731886	988/1000	NonOverlappingTemplate
108	93	112	98	94	99	96	85	116	99	0.538182	991/1000	NonOverlappingTemplate
87	108	112	99	90	100	101	100	106	97	0.812905	992/1000	NonOverlappingTemplate
105	90	101	91	102	93	118	117	88	95	0.317565	986/1000	NonOverlappingTemplate
102	98	96	97	120	96	102	100	92	97	0.811080	983/1000	NonOverlappingTemplate
98	109	92	105	99	102	103	113	100	79	0.536163	991/1000	NonOverlappingTemplate
105	88	97	98	90	102	97	115	108	100	0.755819	991/1000	NonOverlappingTemplate
91	97	91	117	93	84	108	100	113	106	0.323668	991/1000	NonOverlappingTemplate
94	108	91	96	104	100	100	96	93	118	0.737915	990/1000	NonOverlappingTemplate
95	97	110	80	106	100	92	98	112	110	0.454053	994/1000	NonOverlappingTemplate
113	92	97	97	88	84	112	114	98	105	0.334538	991/1000	OverlappingTemplate
112	113	93	91	105	97	93	90	101	105	0.686955	990/1000	Universal
102	97	90	101	99	106	101	109	92	103	0.961869	991/1000	ApproximateEntropy
68	60	66	68	55	57	72	69	45	56	0.309707	610/616	RandomExcursions
90	64	72	69	47	55	63	52	59	45	0.001739	606/616	RandomExcursions
71	59	58	72	66	63	64	40	62	61	0.230460	607/616	RandomExcursions

76	60	66	57	59	60	64	60	65	49	0.611534	607/616	RandomExcursions
68	59	64	52	64	52	74	57	59	67	0.584363	610/616	RandomExcursions
52	57	61	64	63	69	53	64	69	64	0.812367	609/616	RandomExcursions
58	56	74	67	54	57	60	57	57	76	0.417839	607/616	RandomExcursions
54	66	56	51	75	46	55	77	72	64	0.047448	612/616	RandomExcursions
75	66	61	70	74	59	47	49	58	57	0.130727	609/616	RandomExcursionsVariant
65	70	80	73	50	62	55	57	46	58	0.056116	610/616	RandomExcursionsVariant
66	81	75	55	62	58	60	43	61	55	0.051077	605/616	RandomExcursionsVariant
75	69	72	63	67	55	46	69	57	43	0.037526	604/616	RandomExcursionsVariant
81	66	60	66	58	61	65	56	57	46	0.204224	607/616	RandomExcursionsVariant
83	74	54	53	60	60	63	52	54	63	0.095069	606/616	RandomExcursionsVariant
77	71	59	65	61	59	54	62	58	50	0.420798	609/616	RandomExcursionsVariant
67	70	70	57	74	64	55	58	50	51	0.290600	610/616	RandomExcursionsVariant
80	54	59	49	70	59	69	53	56	67	0.135745	606/616	RandomExcursionsVariant
59	62	62	61	65	51	59	68	79	50	0.337378	610/616	RandomExcursionsVariant
66	50	71	62	68	56	55	66	65	57	0.659297	610/616	RandomExcursionsVariant
60	67	62	59	58	54	64	56	75	61	0.797219	608/616	RandomExcursionsVariant
61	64	52	56	73	49	85	68	50	58	0.025333	610/616	RandomExcursionsVariant
58	72	52	49	60	68	71	60	60	66	0.463387	610/616	RandomExcursionsVariant
69	49	58	61	54	71	67	68	55	64	0.520986	612/616	RandomExcursionsVariant
66	61	49	70	67	68	58	62	61	54	0.693243	610/616	RandomExcursionsVariant
64	54	65	79	61	60	53	71	56	53	0.297665	610/616	RandomExcursionsVariant
58	72	65	61	60	60	69	61	66	44	0.472781	612/616	RandomExcursionsVariant
99	93	105	109	91	100	115	93	92	103	0.755819	990/1000	Serial
92	93	123	118	86	94	98	105	84	107	0.082513	988/1000	Serial
94	83	111	101	97	88	107	105	100	114	0.465415	987/1000	LinearComplexity

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 980 for a sample size = 1000 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately = 602 for a sample size = 616 binary sequences.

For further guidelines construct a probability table using the MAPLE program provided in the addendum section of the documentation.

Ciphertext (100 Bitstream)

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <data\data_samples.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
14	15	11	9	8	8	7	12	6	10	0.534146	99/100	Frequency
10	14	10	7	9	9	13	11	10	7	0.867692	99/100	BlockFrequency
12	14	14	7	14	5	9	9	7	9	0.366918	99/100	CumulativeSums
15	14	11	9	9	4	8	12	11	7	0.366918	99/100	CumulativeSums
7	8	8	9	10	11	9	15	12	11	0.834308	100/100	Runs
18	10	9	11	10	11	6	4	6	15	0.066882	99/100	LongestRun
10	13	6	10	15	7	7	12	11	9	0.595549	100/100	Rank
13	15	4	11	9	11	9	8	11	9	0.534146	98/100	FFT

14	9	6	11	11	6	16	7	10	10	0.383827	98/100	NonOverlappingTemplate
13	8	8	14	13	9	6	12	9	8	0.657933	100/100	NonOverlappingTemplate
10	11	16	9	13	7	6	8	8	12	0.494392	99/100	NonOverlappingTemplate
9	4	9	11	13	9	10	8	12	15	0.514124	98/100	NonOverlappingTemplate
11	10	11	7	6	14	16	8	4	13	0.171867	100/100	NonOverlappingTemplate
8	7	17	13	10	7	10	10	10	8	0.494392	99/100	NonOverlappingTemplate
4	6	16	14	11	6	13	15	5	10	0.035174	100/100	NonOverlappingTemplate
8	12	12	8	11	13	8	7	9	12	0.883171	100/100	NonOverlappingTemplate
12	8	5	24	9	9	9	5	9	10	0.002203	98/100	NonOverlappingTemplate
9	5	10	12	10	8	9	21	7	9	0.055361	100/100	NonOverlappingTemplate
10	13	12	8	7	7	8	13	14	8	0.657933	100/100	NonOverlappingTemplate
5	9	11	11	6	10	16	5	11	16	0.115387	99/100	NonOverlappingTemplate
11	9	13	3	10	12	11	8	14	9	0.474986	100/100	NonOverlappingTemplate
13	11	9	10	7	11	7	14	9	9	0.851383	99/100	NonOverlappingTemplate
10	9	13	9	4	8	15	9	15	8	0.304126	99/100	NonOverlappingTemplate
8	13	12	9	6	9	10	7	12	14	0.699313	100/100	NonOverlappingTemplate
10	6	10	13	8	7	7	14	12	13	0.574903	99/100	NonOverlappingTemplate
9	10	7	13	13	7	10	11	10	10	0.924076	100/100	NonOverlappingTemplate
7	8	10	8	11	13	4	15	11	13	0.366918	100/100	NonOverlappingTemplate
14	5	4	9	8	13	10	16	13	8	0.122325	98/100	NonOverlappingTemplate
7	7	5	13	8	12	14	13	12	9	0.437274	100/100	NonOverlappingTemplate
15	9	6	9	7	14	11	11	9	9	0.616305	99/100	NonOverlappingTemplate
12	7	10	10	15	9	9	11	10	7	0.834308	100/100	NonOverlappingTemplate
11	7	13	8	14	11	9	8	11	8	0.834308	98/100	NonOverlappingTemplate
3	11	11	7	20	18	11	11	4	4	0.000474	100/100	NonOverlappingTemplate
11	7	4	14	11	11	6	18	9	9	0.102526	100/100	NonOverlappingTemplate
7	8	12	17	7	9	10	7	14	9	0.334538	100/100	NonOverlappingTemplate
9	8	10	12	9	6	15	10	8	13	0.699313	100/100	NonOverlappingTemplate
16	11	9	16	14	9	7	7	4	7	0.080519	100/100	NonOverlappingTemplate
7	10	15	11	16	9	6	9	11	6	0.304126	99/100	NonOverlappingTemplate
11	5	12	8	11	12	14	11	12	4	0.383827	100/100	NonOverlappingTemplate
8	10	19	9	4	10	11	9	11	9	0.181557	100/100	NonOverlappingTemplate
10	6	12	12	11	9	15	9	11	5	0.554420	99/100	NonOverlappingTemplate
8	8	7	7	16	9	13	9	11	12	0.554420	100/100	NonOverlappingTemplate
14	10	8	14	9	12	8	9	6	10	0.719747	99/100	NonOverlappingTemplate
11	10	8	7	12	10	8	11	15	8	0.816537	99/100	NonOverlappingTemplate
11	9	8	11	13	7	7	6	15	13	0.494392	97/100	NonOverlappingTemplate
7	10	8	15	9	11	15	10	8	7	0.554420	100/100	NonOverlappingTemplate
10	10	7	8	5	10	9	15	14	12	0.494392	100/100	NonOverlappingTemplate
7	8	17	14	6	8	14	8	14	4	0.048716	100/100	NonOverlappingTemplate
13	9	13	6	11	7	13	7	13	8	0.574903	99/100	NonOverlappingTemplate
8	8	8	14	6	13	8	10	17	8	0.275709	100/100	NonOverlappingTemplate
7	8	11	8	6	13	9	15	17	6	0.145326	99/100	NonOverlappingTemplate
9	12	8	11	6	10	9	13	14	8	0.779188	100/100	NonOverlappingTemplate
10	11	11	12	10	10	8	5	11	12	0.911413	99/100	NonOverlappingTemplate
9	12	9	10	11	14	11	7	11	6	0.834308	100/100	NonOverlappingTemplate
8	13	11	7	12	9	11	5	10	14	0.637119	100/100	NonOverlappingTemplate
12	11	9	6	10	13	11	9	8	11	0.924076	99/100	NonOverlappingTemplate
16	8	6	10	15	9	12	5	12	7	0.191687	99/100	NonOverlappingTemplate
10	10	10	16	7	14	8	5	9	11	0.419021	98/100	NonOverlappingTemplate
8	6	15	10	14	11	11	8	9	8	0.616305	100/100	NonOverlappingTemplate
8	6	9	17	14	8	10	12	3	13	0.085587	99/100	NonOverlappingTemplate
8	9	11	13	5	9	7	14	13	11	0.574903	98/100	NonOverlappingTemplate
15	12	8	9	16	15	5	8	8	4	0.058984	99/100	NonOverlappingTemplate
11	7	10	8	7	17	14	8	8	10	0.383827	98/100	NonOverlappingTemplate
13	13	10	11	9	7	7	10	7	13	0.779188	99/100	NonOverlappingTemplate
5	8	8	10	14	9	15	12	12	7	0.419021	100/100	NonOverlappingTemplate
6	16	5	7	8	14	12	15	8	9	0.122325	100/100	NonOverlappingTemplate
12	12	12	6	11	10	9	8	8	12	0.897763	97/100	NonOverlappingTemplate

9	10	10	14	7	9	9	11	6	15	0.637119	97/100	NonOverlappingTemplate
10	11	8	10	6	12	9	11	14	9	0.883171	100/100	NonOverlappingTemplate
9	14	8	11	7	6	11	9	11	14	0.678686	100/100	NonOverlappingTemplate
10	10	7	10	11	11	9	9	12	11	0.994250	98/100	NonOverlappingTemplate
6	9	15	15	9	9	7	13	7	10	0.383827	100/100	NonOverlappingTemplate
7	11	9	12	6	11	15	7	14	8	0.474986	99/100	NonOverlappingTemplate
8	7	7	14	11	6	11	12	6	18	0.122325	100/100	NonOverlappingTemplate
15	9	8	12	9	9	10	8	12	8	0.851383	97/100	NonOverlappingTemplate
10	11	8	13	12	7	10	10	13	6	0.816537	99/100	NonOverlappingTemplate
11	6	12	3	9	16	15	11	10	7	0.115387	98/100	NonOverlappingTemplate
8	10	6	3	12	12	13	8	12	16	0.162606	100/100	NonOverlappingTemplate
10	11	10	13	9	12	7	5	8	15	0.554420	99/100	NonOverlappingTemplate
11	14	9	8	9	9	11	14	10	5	0.678686	99/100	NonOverlappingTemplate
8	9	7	12	9	10	8	11	14	12	0.883171	97/100	NonOverlappingTemplate
11	8	12	12	12	10	9	7	9	10	0.971699	99/100	NonOverlappingTemplate
14	9	6	11	11	6	16	7	10	10	0.383827	98/100	NonOverlappingTemplate
5	14	11	14	13	8	8	9	12	6	0.383827	99/100	NonOverlappingTemplate
5	7	12	8	8	10	14	13	12	11	0.574903	99/100	NonOverlappingTemplate
12	8	15	5	11	13	11	4	12	9	0.275709	99/100	NonOverlappingTemplate
7	17	11	13	9	9	8	9	10	7	0.494392	99/100	NonOverlappingTemplate
13	5	9	8	5	11	15	8	16	10	0.162606	98/100	NonOverlappingTemplate
9	15	6	15	11	8	9	8	11	8	0.514124	100/100	NonOverlappingTemplate
11	11	12	11	8	7	7	14	6	13	0.637119	100/100	NonOverlappingTemplate
11	10	9	11	9	11	11	14	9	5	0.851383	100/100	NonOverlappingTemplate
12	12	11	9	13	9	5	10	11	8	0.834308	99/100	NonOverlappingTemplate
11	11	18	11	5	15	6	9	8	6	0.080519	99/100	NonOverlappingTemplate
10	8	9	10	17	8	9	10	11	8	0.699313	99/100	NonOverlappingTemplate
5	6	9	14	12	11	12	8	11	12	0.574903	99/100	NonOverlappingTemplate
5	10	11	10	10	6	11	9	16	12	0.494392	98/100	NonOverlappingTemplate
7	17	7	10	5	8	11	9	11	15	0.191687	98/100	NonOverlappingTemplate
12	17	11	12	4	4	14	6	12	8	0.048716	99/100	NonOverlappingTemplate
10	9	12	10	9	11	12	9	11	7	0.987896	98/100	NonOverlappingTemplate
10	11	17	4	12	10	8	4	11	13	0.122325	100/100	NonOverlappingTemplate
11	8	9	8	12	13	9	11	13	6	0.834308	98/100	NonOverlappingTemplate
6	10	11	7	5	9	13	11	14	14	0.401199	100/100	NonOverlappingTemplate
6	7	17	9	11	6	7	12	14	11	0.202268	97/100	NonOverlappingTemplate
8	16	6	11	6	10	13	6	12	12	0.304126	99/100	NonOverlappingTemplate
14	11	9	9	9	11	2	14	7	14	0.181557	100/100	NonOverlappingTemplate
9	10	10	11	8	12	9	10	11	10	0.998821	99/100	NonOverlappingTemplate
8	7	14	11	6	16	11	10	7	10	0.419021	99/100	NonOverlappingTemplate
10	10	8	11	10	9	7	12	11	12	0.983453	100/100	NonOverlappingTemplate
13	6	3	12	11	11	9	8	15	12	0.249284	99/100	NonOverlappingTemplate
12	7	12	9	6	9	12	9	12	12	0.851383	99/100	NonOverlappingTemplate
9	11	10	6	6	11	12	13	14	8	0.657933	100/100	NonOverlappingTemplate
14	8	8	7	10	14	9	11	10	9	0.816537	100/100	NonOverlappingTemplate
13	9	10	10	9	9	9	10	10	11	0.997823	99/100	NonOverlappingTemplate
7	11	16	9	8	11	10	7	9	12	0.678686	100/100	NonOverlappingTemplate
8	8	14	10	11	9	4	8	11	17	0.236810	99/100	NonOverlappingTemplate
13	7	8	10	9	12	11	6	15	9	0.637119	97/100	NonOverlappingTemplate
7	6	6	10	24	10	10	9	11	7	0.003201	99/100	NonOverlappingTemplate
7	10	21	6	10	10	12	6	8	10	0.048716	99/100	NonOverlappingTemplate
9	15	10	9	5	6	7	14	12	13	0.304126	99/100	NonOverlappingTemplate
8	10	6	10	11	6	14	15	13	7	0.383827	100/100	NonOverlappingTemplate
14	10	8	11	10	13	9	9	6	10	0.851383	99/100	NonOverlappingTemplate
10	15	8	8	13	10	10	8	11	7	0.779188	100/100	NonOverlappingTemplate
14	9	7	8	12	8	18	6	10	8	0.202268	98/100	NonOverlappingTemplate
15	4	10	14	10	16	9	9	6	7	0.122325	99/100	NonOverlappingTemplate
12	6	17	12	5	7	10	10	10	11	0.289667	99/100	NonOverlappingTemplate
10	11	5	13	13	6	8	14	11	9	0.514124	99/100	NonOverlappingTemplate

7	8	8	9	14	10	13	9	7	15	0.554420	98/100	NonOverlappingTemplate
11	18	11	6	8	9	5	12	6	14	0.096578	100/100	NonOverlappingTemplate
5	8	12	14	13	11	7	12	10	8	0.574903	100/100	NonOverlappingTemplate
11	11	7	12	7	15	8	11	11	7	0.699313	99/100	NonOverlappingTemplate
11	12	14	12	12	7	4	10	9	9	0.574903	100/100	NonOverlappingTemplate
16	11	15	8	8	4	5	15	7	11	0.055361	99/100	NonOverlappingTemplate
13	9	9	9	8	8	8	9	17	10	0.595549	100/100	NonOverlappingTemplate
10	7	9	14	13	9	9	11	11	7	0.851383	100/100	NonOverlappingTemplate
9	6	13	10	4	9	9	12	13	15	0.334538	100/100	NonOverlappingTemplate
10	13	10	10	10	6	8	10	10	13	0.924076	99/100	NonOverlappingTemplate
8	7	9	12	15	12	7	14	9	7	0.514124	99/100	NonOverlappingTemplate
9	11	8	6	14	5	13	12	10	12	0.534146	99/100	NonOverlappingTemplate
7	9	12	7	10	7	10	10	14	14	0.699313	99/100	NonOverlappingTemplate
11	14	9	5	13	12	9	11	10	6	0.595549	98/100	NonOverlappingTemplate
3	9	14	17	10	9	9	13	4	12	0.055361	100/100	NonOverlappingTemplate
10	12	6	8	12	14	9	9	10	10	0.867692	100/100	NonOverlappingTemplate
16	8	10	11	11	10	10	7	12	5	0.534146	99/100	NonOverlappingTemplate
10	11	15	13	11	9	7	10	6	8	0.678686	100/100	NonOverlappingTemplate
12	14	9	7	12	11	5	13	7	10	0.554420	99/100	NonOverlappingTemplate
10	11	17	13	11	8	10	9	6	5	0.304126	100/100	NonOverlappingTemplate
8	9	13	10	7	12	11	6	15	9	0.637119	97/100	NonOverlappingTemplate
14	14	5	12	9	9	12	10	9	6	0.494392	100/100	NonOverlappingTemplate
14	13	12	5	7	14	8	11	8	8	0.419021	100/100	NonOverlappingTemplate
4	24	11	7	8	6	11	10	8	11	0.001509	100/100	NonOverlappingTemplate
8	15	13	10	15	8	6	7	9	9	0.401199	100/100	NonOverlappingTemplate
8	14	10	11	7	10	11	11	11	7	0.897763	99/100	NonOverlappingTemplate
6	11	9	9	16	8	14	7	12	8	0.419021	100/100	NonOverlappingTemplate
14	11	12	18	5	10	11	9	4	6	0.058984	100/100	NonOverlappingTemplate
12	11	13	8	11	9	14	8	8	6	0.739918	98/100	NonOverlappingTemplate
11	8	12	12	12	10	9	7	9	10	0.971699	99/100	NonOverlappingTemplate
9	14	11	8	6	7	12	9	11	13	0.719747	97/100	OverlappingTemplate
10	8	7	15	10	8	9	6	10	17	0.289667	100/100	Universal
15	9	13	7	11	9	8	12	11	5	0.534146	99/100	ApproximateEntropy
4	5	11	5	7	3	6	5	7	6	0.334538	59/59	RandomExcursions
6	7	5	5	4	11	5	5	6	5	0.474986	58/59	RandomExcursions
7	6	4	9	5	4	5	7	7	5	0.719747	57/59	RandomExcursions
4	4	4	6	8	4	7	9	7	6	0.554420	58/59	RandomExcursions
3	6	3	7	8	10	6	3	4	9	0.129620	58/59	RandomExcursions
8	6	7	7	4	3	3	7	8	6	0.514124	58/59	RandomExcursions
7	4	6	3	7	5	6	11	4	6	0.304126	57/59	RandomExcursions
5	2	4	7	3	5	10	7	8	8	0.162606	58/59	RandomExcursions
7	9	5	3	6	8	4	6	5	6	0.595549	59/59	RandomExcursionsVariant
8	8	2	5	5	6	6	5	6	8	0.554420	59/59	RandomExcursionsVariant
8	9	0	2	11	4	7	6	7	5	0.012650	59/59	RandomExcursionsVariant
11	2	5	6	5	5	7	6	4	8	0.202268	58/59	RandomExcursionsVariant
9	6	4	5	4	6	4	7	7	7	0.678686	58/59	RandomExcursionsVariant
9	6	7	4	3	7	5	5	11	2	0.090936	59/59	RandomExcursionsVariant
7	10	6	0	6	5	7	2	8	8	0.042808	59/59	RandomExcursionsVariant
6	7	9	3	4	1	8	8	7	6	0.162606	59/59	RandomExcursionsVariant
7	4	7	6	9	6	4	8	4	4	0.554420	58/59	RandomExcursionsVariant
4	11	5	4	3	2	4	9	11	6	0.012650	59/59	RandomExcursionsVariant
7	8	6	4	4	9	2	6	6	7	0.401199	58/59	RandomExcursionsVariant
7	9	7	4	5	6	3	2	5	11	0.090936	58/59	RandomExcursionsVariant
6	9	7	5	5	5	4	6	3	9	0.474986	58/59	RandomExcursionsVariant
10	4	6	6	7	3	10	3	3	7	0.102526	58/59	RandomExcursionsVariant
9	6	6	8	4	0	10	1	7	8	0.010988	58/59	RandomExcursionsVariant
6	7	7	8	4	4	5	8	6	4	0.719747	58/59	RandomExcursionsVariant
6	7	7	6	8	4	4	3	6	8	0.637119	58/59	RandomExcursionsVariant
9	4	8	10	3	5	3	3	5	9	0.071177	58/59	RandomExcursionsVariant

10	11	13	15	9	9	10	6	11	6	0.637119	98/100	Serial
13	14	15	5	14	6	6	12	7	8	0.122325	98/100	Serial
7	11	5	14	12	12	6	9	13	11	0.474986	100/100	LinearComplexity

The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 56 for a sample size = 59 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
