

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

TRẦN SỸ NAM

NGHIÊN CỨU GIẢI PHÁP NÂNG CAO HIỆU NĂNG CỦA
MỘT SỐ THUẬT TOÁN MÃ HÓA TRONG BẢO MẬT
LUỒNG IP

LUẬN ÁN TIẾN SĨ KỸ THUẬT

HÀ NỘI – 2026

BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

TRẦN SỸ NAM

NGHIÊN CỨU GIẢI PHÁP NÂNG CAO HIỆU NĂNG CỦA
MỘT SỐ THUẬT TOÁN MÃ HÓA TRONG BẢO MẬT
LUỒNG IP

NGÀNH: KỸ THUẬT MẬT MÃ
MÃ SỐ: 9520209

LUẬN ÁN TIẾN SĨ KỸ THUẬT

NGƯỜI HƯỚNG DẪN KHOA HỌC:

- PGS.TS. LƯƠNG THẾ DŨNG
- TS. HOÀNG VĂN THỨC

HÀ NỘI – 2026

LỜI CAM ĐOAN

Tôi xin cam đoan, đây là công trình nghiên cứu của riêng tôi. Các số liệu và kết quả trình bày trong luận án là hoàn toàn trung thực và chưa có tác giả nào công bố trong bất cứ một công trình nào khác.

Giáo viên hướng dẫn 1

Tác giả

PGS.TS. Lương Thế Dũng

Trần Sỹ Nam

Giáo viên hướng dẫn 2

TS. Hoàng Văn Thúc

LỜI CẢM ƠN

Luận án này được thực hiện tại Học viện Kỹ thuật mật mã – Ban Cơ yếu Chính phủ. Lời đầu tiên, NCS xin bày tỏ lòng biết ơn sâu sắc tới PGS.TS. Lương Thế Dũng, TS. Hoàng Văn Thức, các thầy đã tận tình giúp đỡ, định hướng và trang bị cho NCS các phương pháp nghiên cứu, kiến thức khoa học và kinh nghiệm quý báu để NCS có thể hoàn thành các kết quả nghiên cứu của luận án.

NCS xin cảm ơn Học viện Kỹ thuật mật mã, Khoa mật mã, Phòng Đào tạo, là cơ sở đào tạo và đơn vị quản lý đã tạo mọi điều kiện, hỗ trợ, giúp đỡ NCS trong quá trình học tập, nghiên cứu.

NCS xin cảm ơn Viện Khoa học - Công nghệ mật mã, Phân viện An toàn nghiệp vụ, Phân viện Khoa học mật mã đã tạo điều kiện thuận lợi, giúp đỡ, hỗ trợ NCS hoàn thành nhiệm vụ nghiên cứu.

Cuối cùng, NCS xin bày tỏ lời cảm ơn tới gia đình, đồng nghiệp và bạn bè, những người luôn là nguồn động viên, chia sẻ và hỗ trợ về tinh thần lẫn vật chất trong suốt hành trình học tập và nghiên cứu. Sự đồng hành của mọi người là động lực to lớn giúp tôi hoàn thành mục tiêu.

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT	v
DANH MỤC CÁC BẢNG	vi
DANH MỤC CÁC HÌNH VẼ.....	vii
MỞ ĐẦU	1
CHƯƠNG 1 TỔNG QUAN CÁC VẤN ĐỀ NGHIÊN CỨU	6
1.1. Thuật toán mã hóa AES	6
1.1.1. S-box và các đại lượng liên quan.....	8
1.1.2. Ma trận MDS và các đại lượng liên quan	10
1.1.3. Độ an toàn của AES	13
1.2. Thuật toán mã hóa Ascon.....	13
1.2.1. Thuật toán mã hóa có xác thực (AEAD)	13
1.2.2. Thuật toán Ascon	14
1.3. Giao thức bảo mật IPSec.....	15
1.3.1. Giao thức ESP	16
1.3.2. Các thành phần mật mã được sử dụng trong IPSec	20
1.3.3. Một số tấn công và kết quả đánh giá về bảo mật và an ninh, an toàn của giao thức IPSec.....	21
1.4. Phân tích các công trình nghiên cứu công bố gần đây và định hướng nghiên cứu của đề tài luận án.....	23
1.4.1. Phân tích các công trình về thành phần mật mã	23
1.4.2. Phân tích các công trình về kiến trúc phần cứng tốc độ cao cho thuật toán mã hóa	26
1.4.2.1. Các công trình về kiến trúc AES tốc độ cao trên FPGA	28
1.4.2.2. Các công trình về kiến trúc Ascon tốc độ cao trên FPGA	29
1.4.3. Phân tích các công trình về kiến trúc tốc độ cao cho giao thức IPSec	30
1.5. Kết luận chương 1	32
CHƯƠNG 2 NGHIÊN CỨU LỰA CHỌN CÁC THÀNH PHẦN MẬT MÃ AN TOÀN HIỆU QUẢ CHO CÀI ĐẶT TỐI ƯU AES	34
2.1. Nghiên cứu lựa chọn S-box 8-bit an toàn hiệu quả	34
2.2. Nghiên cứu lựa chọn ma trận MDS an toàn hiệu quả.....	50
2.3. Đánh giá độ an toàn của AES-P.....	54
2.3.1. Tiêu chí khuyến cáo	54
2.3.2. Kiểm thử thống kê ngẫu nhiên (NIST SP 800-22)	56
2.3.3. Độ phức tạp tính toán.....	57
2.4. Đánh giá hiệu năng của mã khối AES-P.....	60
2.5. Kết luận chương 2	61

CHƯƠNG 3 NGHIÊN CỨU XÂY DỰNG CÁC KIẾN TRÚC PHẦN CỨNG TỐC ĐỘ CAO CHO THUẬT TOÁN MÃ HÓA AES-P VÀ ASCON.....	62
3.1. Nghiên cứu xây dựng kiến trúc phần cứng tốc độ cao cho AES-P.....	62
3.1.1. Cải tiến các thành phần mật mã	63
3.1.2. Hoàn thiện kiến trúc đường ống.....	66
3.1.3. So sánh với các kết quả khác	72
3.2. Nghiên cứu xây dựng kiến trúc phần cứng tốc độ cao cho Ascon	75
3.2.1. Động lực nghiên cứu.....	75
3.2.2. Kiến trúc phần cứng đề xuất	79
3.2.3. So sánh với các kết quả khác	83
3.3. Đánh giá độ an toàn các kiến trúc đề xuất	84
3.4. Kết luận chương 3	85
CHƯƠNG 4 NGHIÊN CỨU ĐỀ XUẤT KIẾN TRÚC HMS-IPSEC	86
4.1. Động lực thiết kế kiến trúc HMS-IPSec	86
4.2. Nghiên cứu đề xuất kiến trúc phần cứng tổng thể	88
4.3. Nghiên cứu đề xuất kiến trúc IPSec tốc độ cao	96
4.3.1. Đề xuất kiến trúc IPSec tốc độ cao	96
4.3.2. So sánh với các kết quả khác	103
4.4. Đánh giá thực nghiệm	108
4.5. Kết luận chương 4	111
KẾT LUẬN	112
DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC ĐÃ CÔNG BỐ	115
MỤC LỤC THAM KHẢO	116
PHỤ LỤC	122
Phụ lục 1. Phần mềm kiểm tra tính chất mật mã của S-box	122
Phụ lục 2. Tập sboxes.txt sử dụng trong luận án	132
Phụ lục 3. Cấu tạo của các bảng IPSec	137
Phụ lục 4. Module S-box và MDS lựa chọn cài đặt theo phương pháp biểu diễn logic	139
Phụ lục 5. Đánh giá thực nghiệm trên Kit ZC706	143

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

Từ viết tắt	Nghĩa tiếng Anh	Nghĩa tiếng Việt
ASIC	Application-specific integrated circuit	Vì mạch tích hợp chuyên dụng
AES	Advanced Encryption Standard	Chuẩn mã hóa nâng cao
AEAD	Authenticated Encryption with Associated Data	Mã hóa có xác thực với dữ liệu liên kết
CAM/TCAM	Content Addressable Memory/Ternary Content Addressable Memory	Bộ nhớ nội dung theo địa chỉ/ Bộ nhớ nội dung theo địa chỉ ba trạng thái
DMA	Direct Memory Access	Truy cập bộ nhớ trực tiếp
ESP	Encapsulating Security Payload	Giao thức đóng gói bảo mật
FPGA	Field Programmable Gate Arrays	Mảng tích hợp có thể lập trình
HMS-IPSec	High-speed Multi-Crypto Secure IPSec Architecture	Kiến trúc IPSec đa mật mã an toàn tốc độ cao
IP	Internet Protocol	Giao thức Internet
IPSec	IP Security	Giao thức bảo mật IP
LWC	Lightweight Cryptography	Mật mã hạng nhẹ
LUT	Lookup Table	Bảng tra
MAC	Media Access Control	Điều khiển truy nhập môi trường
NAT	Network Address Translation	Biên dịch địa chỉ mạng
NPU	Network Processing Unit	Vi xử lý mạng
SoC	System on Chip	Hệ thống trên chip
SPD	Security Policy Database	Cơ sở dữ liệu chính sách an toàn
SAD	Security Association database	Cơ sở dữ liệu tham số, khóa an toàn
VPN	Virtual Private Network	Mạng riêng ảo bảo mật

DANH MỤC CÁC BẢNG

Bảng 1.1. Số nhánh và điểm bất động trong tầng khuếch tán của mã khối	12
.....	12
Bảng 1.2. Tấn công bicliques lên AES	13
Bảng 2.1. So sánh tính chất mật mã của Sp với các S-box tiêu chuẩn	43
Bảng 2.2. So sánh tính chất mật mã của Sp với các công trình khác	44
Bảng 2.3. So sánh tính chất mật mã của Sp với các S-box sử dụng PEIGEN	44
.....	44
Bảng 2.4. Biểu diễn ANF của f_0	48
Bảng 2.5. Độ phức tạp biểu diễn logic của S-box Sp	49
Bảng 2.6. So sánh số lượng LUT của S-box Sp trên FPGA	50
Bảng 2.7. So sánh số lượng các ma trận MDS	53
Bảng 2.8. So sánh với các kết quả khác	54
Bảng 2.9. So sánh hiệu ứng SAC của AES-P và AES nguyên gốc	56
Bảng 2.10. Kết quả kiểm thử NIST SP 800-22	57
Bảng 2.11. Độ an toàn thám vi sai	58
Bảng 2.12. Độ an toàn thám tuyến tính	58
Bảng 2.13. Kết quả số biến QUBO theo hai phương pháp	60
Bảng 2.14. Kết quả đánh giá hiệu năng	60
Bảng 3.1. Kích thước và độ trễ của kiến trúc đề xuất	72
Bảng 3.2. So sánh các kiến trúc AES tốc độ cao	73
Bảng 3.3. So sánh các kiến trúc Ascon tốc độ cao	83
Bảng 4.1. Vị trí thiết lập các bit	90
Bảng 4.2. Định dạng gói ESP header và ESP trailer	99
Bảng 4.3. Tần số tối đa của kiến trúc IPsec trên Zynq-7000 SoC	103
Bảng 4.4. Phân bố chu kỳ xung nhịp đồng hồ của các thành phần	103
Bảng 4.5. So sánh các kết quả cài đặt IPsec tốc độ cao	105
Bảng 4.6. Số khối ESP tối ưu nhất	108
Bảng 4.7. So sánh độ trễ của các công trình	108
Bảng 4.8. Các công nghệ sử dụng trong thực nghiệm	109
Bảng 4.9. Các IPCore sử dụng trong thiết kế	109

DANH MỤC CÁC HÌNH VẼ

Hình 1.1. Biến đổi ShiftRows trong AES.....	7
Hình 1.2. Quá trình mã hóa AEAD	14
Hình 1.3. Quá trình mã hóa Ascon [6].....	15
Hình 1.4. Các thành phần chính của IPSec.....	16
Hình 1.5. ESP chế độ tunnel.....	17
Hình 1.6. ESP chế độ transport.....	17
Hình 1.7. Cấu trúc gói ESP.....	18
Hình 2.1. So sánh hiệu ứng AC của AES-P và AES nguyên gốc	55
Hình 3.1. Cài đặt 1 bit đầu ra của S-box S_p	63
Hình 3.2. Lược đồ khóa pre-compute và on-the-fly	65
Hình 3.3. Các vị trí thêm thanh ghi của hàm vòng.....	68
Hình 3.4. Các vị trí thêm thanh ghi của lược đồ khóa.....	69
Hình 3.5. Kiến trúc tối ưu của hàm vòng AES-P	71
Hình 3.6. Kiến trúc tối ưu của lược đồ khóa AES-P	71
Hình 3.7. Giảm đồ thời gian của AES-P-128	72
Hình 3.8. So sánh tốc độ và độ trễ với các thiết kế AES tốc độ cao khác.....	74
Hình 3.9. Yêu cầu bộ nhớ cần thiết để hoạt động không gián đoạn.....	77
Hình 3.10. Giao tiếp LWC.....	78
Hình 3.11. Định dạng của PDI.....	78
Hình 3.12. Giao tiếp đề xuất.....	79
Hình 3.13. Kiến trúc Ascon-AEAD128 đề xuất.....	81
Hình 3.14. Giảm đồ thời gian của Ascon-AEAD128 đề xuất	82
Hình 4.1. Kiến trúc HMS-IPSec	89
Hình 4.2. Cấu trúc trường thông tin.....	90
Hình 4.3. Gói tin đi qua các giao diện mạng	90
Hình 4.4. Kiến trúc IPSec tốc độ cao trên phần cứng	95
Hình 4.5. Các đường dữ liệu của kiến trúc IPSec.....	96
Hình 4.6. Kiến trúc IPSec tốc độ cao đề xuất.....	97
Hình 4.7. Tốc độ với các gói tin kích thước khác nhau.....	104
Hình 4.8. Độ trễ với các gói tin kích thước khác nhau.....	104
Hình 4.9. So sánh tốc độ và tài nguyên của kiến trúc IPSec sử dụng 1 khối ESP với các các thiết kế IPSec tốc độ cao khác.....	106
Hình 4.10. Sự phụ thuộc của tốc độ IPSec lên số khối ESP Process	107
Hình 4.11. Sự phụ thuộc của hiệu quả cài đặt lên số khối ESP Process ..	107

MỞ ĐẦU

1. Lời nói đầu

Sự bùng nổ của kỷ nguyên số đang định hình một mạng lưới truyền thông ngày càng phức tạp, nơi dữ liệu được tạo ra, truyền tải và xử lý theo thời gian thực trên quy mô chưa từng có. Từ các trung tâm dữ liệu quy mô lớn, trí tuệ nhân tạo tại biên, đến mạng 5G/6G và Internet vạn vật (IoT), hạ tầng an ninh mạng phải đáp ứng không chỉ yêu cầu an toàn mà còn an toàn với hiệu suất tối đa: thông lượng cao, độ trễ thấp, khả năng mở rộng linh hoạt. Thách thức này bộc lộ hạn chế của xử lý dựa trên CPU truyền thống khi khối lượng dữ liệu ngày càng tăng, trong khi các nguyên thủy mật mã (mã hóa, xác thực, toàn vẹn) tác động trực tiếp lên luồng dữ liệu thời gian thực không còn đáp ứng được yêu cầu này. Điều này dẫn đến sự chuyển hướng sang các giải pháp phần cứng chuyên dụng như FPGA, ASIC nhằm nâng cao tốc độ, giảm độ trễ mà vẫn đảm bảo độ an toàn - những đặc tính khó đạt được nếu chỉ dựa vào phần mềm [3], [4], [5].

Cách tiếp cận dựa trên phần cứng càng trở nên cấp thiết hơn khi xét đến bản chất của các giao thức bảo mật luồng IP như IPSec hay TLS/DTLS. Các giao thức này tác động trực tiếp lên gói tin trong quá trình truyền tải, khiến chi phí tính toán của các nguyên thủy mật mã ảnh hưởng đáng kể đến hiệu suất tổng thể của hệ thống. Một độ trễ nhỏ cũng có thể gây suy giảm chất lượng dịch vụ (QoS) trong những ứng dụng yêu cầu độ nhạy thời gian cao như truyền phát video 4K/8K, hội nghị truyền hình đa điểm, thực tế ảo/tăng cường (AR/VR) hoặc điều khiển công nghiệp thời gian thực. Do đó, vấn đề cốt lõi không chỉ nằm ở việc xác định “thuật toán nào an toàn”, mà còn là “thuật toán nào vừa an toàn vừa có thể thực thi hiệu quả trên phần cứng”.

Hiện nay có nhiều thuật toán cung cấp dịch vụ mật mã khác nhau, song không phải tất cả đều phù hợp để triển khai hiệu quả trên nền tảng phần cứng. Ví dụ, DES và 3DES không còn đảm bảo an toàn và không thích hợp với bài toán hiệu năng cao. Ngược lại, AES (Advanced Encryption Standard) tiếp tục được ứng dụng rộng rãi cả trong thực tiễn và nghiên cứu học thuật nhờ những ưu điểm về độ an toàn và tính tối ưu khi triển khai trên phần cứng, đặc biệt là chế độ AES-GCM. Chế độ này thuộc nhóm AEAD (Authenticated Encryption with Associated Data), cho phép cung cấp đồng thời các thuộc tính bí mật, xác thực và toàn vẹn trong một lược đồ thống nhất, với mô hình an toàn rõ ràng và hạn chế sai sót cấu hình. Đáng chú ý, AES-GCM còn tỏ ra vượt trội về hiệu năng khi hiện thực trên phần cứng nhờ khả năng song song hóa cao của cả quá trình mã hóa và hàm nhân Galois (GHASH). Chính vì vậy, việc nâng cao hiệu năng của AES/AES-GCM trên phần cứng vẫn là chủ đề được quan tâm mạnh mẽ trong cộng đồng nghiên cứu quốc tế.

Bên cạnh đó, AEAD đã trở thành yêu cầu bắt buộc trong thiết kế thuật toán mã hóa xác thực hạng nhẹ cho các mạng IoT, do AES không thực sự được thiết kế cho bài toán này. Tháng 8 năm 2025, NIST đã chuẩn hoá Ascon là tiêu chuẩn thuật toán mã hoá xác thực hạng nhẹ. Trong bối cảnh mạng IoT, nơi việc tối ưu hóa cho các thiết bị tài nguyên hạn chế là yếu tố then chốt, một số trường hợp ưu tiên hiệu năng cao và độ trễ thấp để đáp ứng các tiêu chí bảo mật trong môi trường truyền thông thời gian thực, chẳng hạn như máy chủ bảo mật trung tâm, máy chủ biên, thiết bị IoT Gateway, Bởi vậy, vấn đề nghiên cứu cải thiện hiệu năng của Ascon trên nền tảng phần cứng cũng trở thành một xu hướng quan trọng và được quan tâm đáng kể trong những năm gần đây.

Các thuật toán mã hóa có thể được cải thiện hiệu suất theo hai cách: (i) tối ưu các thành phần bên trong thuật toán hoặc (ii) tối ưu kiến trúc khi triển khai trên

phần cứng. Tuy nhiên, việc tối ưu các thành phần bên trong của thuật toán mã hoá như S-box, ma trận MDS (Maximum Distance Separable) sẽ thực sự phù hợp và mang lại hiệu quả rõ rệt hơn cho AES so với Ascon, bởi sự khác biệt về cấu trúc và tính toán:

- AES (mã khối SPN): Hiệu năng bị chi phối mạnh bởi S-box (SubBytes) và ma trận MDS (MixColumns). Việc lựa chọn, thiết kế lại các thành phần này theo hướng phù hợp phần cứng sẽ giúp cải thiện đáng kể thông lượng và độ trễ.

- Ascon (AEAD hạng nhẹ dựa trên sponge/hoán vị): Được thiết kế tối giản cho các thiết bị có tài nguyên hạn chế, mỗi vòng được thực hiện đơn giản nên không gian cải tiến không còn nhiều khi chỉnh sửa các thành phần bên trong. Cải thiện hiệu năng chủ yếu đến từ các thay đổi kiến trúc như mở vòng lặp (unroll), lặp lại (iterative), tổ chức trạng thái/bus.

Trên cơ sở các phân tích và đánh giá nêu trên, để nâng cao hiệu năng của các thuật toán mã hóa AES và Ascon trong bảo mật luồng IP, luận án hướng đến ba mục tiêu cụ thể: tăng thông lượng xử lý dữ liệu, giảm độ trễ, giảm tài nguyên phần cứng sử dụng trong quá trình triển khai. Để đạt được các mục tiêu đó, luận án tập trung vào ba hướng nghiên cứu chính sau:

Thứ nhất, luận án nghiên cứu, lựa chọn và thiết kế các thành phần mật mã mới dành cho AES; cụ thể, đề xuất S-box và ma trận MDS có cấu trúc thân thiện với phần cứng để cải thiện thông lượng và độ trễ, đồng thời vẫn bảo đảm yêu cầu an toàn.

Thứ hai, luận án nghiên cứu xây dựng các kiến trúc phần cứng tốc độ cao cho AES và Ascon. Trong đó đề xuất các kiến trúc mới sử dụng các kỹ thuật song song hóa, kỹ thuật đường ống (pipeline), mở vòng lặp, tối ưu tổ chức thanh ghi, từ đó tìm ra kiến trúc phù hợp cho các hệ thống bảo mật hiệu năng cao.

Thứ ba, luận án tập trung vào việc đề xuất một kiến trúc phần cứng mới cho giao thức IPSec bởi khi xét đến bài toán tổng thể, việc cải thiện các thành phần mật mã riêng lẻ là chưa đủ để đáp ứng yêu cầu của các hệ thống tốc độ cao. Kiến trúc mới được thiết kế để tận dụng trực tiếp các kết quả ở hai hướng nghiên cứu trên, trong đó sử dụng thuật toán mã hoá AES và Ascon đã cải tiến, đồng thời đề xuất các thuật toán xử lý gói tin và phân luồng dữ liệu hiệu quả để nâng cao thông lượng và giảm độ trễ.

Để hiện thực hóa mục tiêu nghiên cứu, luận án sử dụng các phương pháp nghiên cứu sau:

- Khảo sát tổng quan các phương pháp xây dựng S-box và ma trận MDS; từ đó đề xuất phương pháp sinh các thành phần mật mã hiệu quả trên phần cứng.
- Vận dụng các kỹ thuật thiết kế kiến trúc phần cứng, đề xuất phương pháp tìm kiếm số thanh ghi nhằm xây dựng kiến trúc tốc độ cao, giảm độ trễ và sử dụng tài nguyên hiệu quả.
- Sử dụng ngôn ngữ mô tả phần cứng và lập trình nhúng để mô phỏng, triển khai và đánh giá thực nghiệm các kiến trúc đề xuất trên nền tảng SoC.

Bố cục của luận án:

Bên cạnh phần mở đầu, phần kết luận và phần phụ lục, luận án được chia thành bốn chương với cấu trúc như sau.

Chương 1. Tổng quan các vấn đề nghiên cứu

Nội dung của chương 1 trình bày tổng quan về các thành phần mật mã, thuật toán mã hóa ứng dụng trong giao thức bảo mật mạng IPSec. Chương 1 cũng phân tích các nghiên cứu liên quan, nhận diện các hạn chế còn tồn tại và đề xuất các hướng nghiên cứu nhằm khắc phục những hạn chế này.

Chương 2. Nghiên cứu lựa chọn các thành phần mật mã an toàn hiệu quả cho cài đặt tối ưu AES

Nội dung của chương 2 trình bày các phương pháp xây dựng thành phần mật mã, trong đó xây dựng S-box và ma trận MDS mới cho mã khối AES nhằm nâng cao hiệu năng khi thực thi trên phần cứng.

Chương 3. Nghiên cứu xây dựng các kiến trúc phần cứng tốc độ cao cho thuật toán mã hóa AES và Ascon

Chương 3 tập trung đề xuất các kiến trúc phần cứng tối ưu cho thuật toán mã hóa AES và Ascon. Các kiến trúc này có khả năng mã hóa dữ liệu tốc độ cao nhằm giải quyết những yêu cầu về hiệu năng cao, độ trễ thấp và hiệu quả tài nguyên trong các hệ thống hiện đại.

Chương 4. Nghiên cứu đề xuất kiến trúc phần cứng để nâng cao hiệu năng hoạt động của IPSec

Nội dung của chương 4 trình bày về kiến trúc phần cứng đề xuất để nâng cao hiệu năng hoạt động của IPSec. Kiến trúc sử dụng các kết quả đã nghiên cứu ở chương 2 và chương 3. Tiến hành thực nghiệm trên Kit ZC706 và đánh giá thực tế.

CHƯƠNG 1

TỔNG QUAN CÁC VẤN ĐỀ NGHIÊN CỨU

Chương 1 của luận án được xây dựng nhằm cung cấp một cái nhìn tổng quan về các thành phần mật mã cốt lõi, bao gồm S-box, ma trận MDS, và các đại lượng mật mã liên quan, cùng với việc phân tích các đặc tính kỹ thuật của AES, Ascon, và IPSec trên nền tảng phần cứng. Thông qua việc đánh giá các công trình nghiên cứu đã công bố, chương này xác định những thách thức còn tồn tại trong việc thiết kế các thành phần mật mã và kiến trúc phần cứng tốc độ cao, từ đó làm cơ sở cho việc đề xuất các hướng nghiên cứu mới.

1.1. Thuật toán mã hóa AES

Advanced Encryption Standard (AES) [11] là mã khối cấu trúc lặp với kích thước khối 128 bit, hỗ trợ các kích thước khóa 128 bits, 192 bits, và 256 bits tương ứng với 10, 12 và 14 vòng. AES được chuẩn hóa năm 2001 trong FIPS-197 bởi Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST). AES xử lý theo khối 128-bit được chia thành 16 bytes dưới dạng ma trận 4×4 , gọi là ma trận trạng thái. Tất cả các phép tính của AES đều hoạt động trên ma trận này. Có bốn bước cơ bản trong một vòng mã của AES là: AddRoundKey, SubBytes, ShiftRows, và MixColumns.

Biến đổi AddRoundKey

Trong biến đổi AddRoundKey, ma trận trạng thái được kết hợp với khóa vòng sử dụng phép XOR.

Biến đổi SubBytes

SubBytes sử dụng **S-box** và là thành phần phi tuyến tính duy nhất trong AES. Tuy nhiên tính chất này cũng làm cho nó bị hạn chế về tốc độ và tài nguyên.

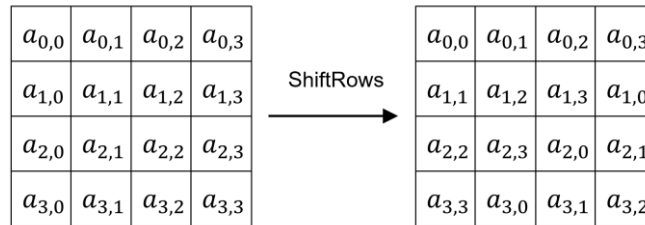
S-box được tính toán thông qua phép nghịch đảo trên trường $GF(2^8)$ và phép biến đổi affine trên trường $GF(2)$:

$$S(x) = \text{Affine}(x^{-1}) \quad (1.1)$$

$$\text{Affine} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Biến đổi ShiftRows

ShiftRows được cài đặt bằng cách dịch mỗi hàng của ma trận theo số bytes nhất định như hình dưới đây:



Hình 1.1. Biến đổi ShiftRows trong AES

Biến đổi MixColumns

Trong biến đổi MixColumns, một ma trận dịch vòng **MDS** (Maximum Distance Separable) được sử dụng để thực hiện phép biến đổi tuyến tính trên các cột của ma trận trạng thái. Ma trận này đảm bảo tính khuếch tán (diffusion) cao, giúp tăng cường độ an toàn của thuật toán bằng cách trộn dữ liệu trong mỗi cột.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Mỗi vòng đều cần một khóa vòng được tạo ra bởi lược đồ mở rộng khóa. Lược đồ mở rộng khóa bao gồm ba bước chính: RotWords, SubWords, và XORs. SubWords tương tự như SubBytes được thực hiện trên S-box, trong khi RotWord tương tự như ShiftRow.

1.1.1. S-box và các đại lượng liên quan

S-box là thành phần phi tuyến duy nhất trong các thuật toán mã hóa khối, đóng vai trò thực hiện nguyên lý xáo trộn (confusion) của Shannon. Một S-box kích thước $n \times m$ được định nghĩa là một ánh xạ $F: \mathbb{F}_{2^n} \rightarrow \mathbb{F}_2^m$. Để đánh giá sức mạnh mật mã, F thường được biểu diễn dưới dạng hệ hàm logic $F(x) = (f_1(x), f_2(x), \dots, f_m(x))$ với $f_i: \mathbb{F}_{2^n} \rightarrow \mathbb{F}_2$ với mọi $1 \leq i \leq m$.

Độ phi tuyến. Độ phi tuyến của S-box đặc trưng cho khả năng kháng lại thám mã tuyến tính. Độ phi tuyến của một hàm logic f , ký hiệu là $N_1(f)$ là khoảng cách Hamming ngắn nhất từ f đến tập các hàm affine A_n :

$$N_1(f) = \min_{\phi \in A_n} d(f, \phi) \quad (1.2)$$

Thông qua biến đổi Walsh-Hadamard $W_f(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus u \cdot x}$, độ phi tuyến được tính bằng công thức:

$$N_1(f) = 2^{n-1} - \frac{1}{2} \max_{u \in \mathbb{F}_2^n} |W_f(u)| \quad (1.3)$$

Độ phi tuyến của S-box F được xác định bởi giá trị nhỏ nhất trong số các độ phi tuyến của tất cả các tổ hợp tuyến tính khác không của các hàm logic thành phần.

$$N_1(F) = \min_{c \in \mathbb{F}_2^m} \{N_1(c \cdot F)\} = \min_{c \in \mathbb{F}_2^m} \{N_1(c_0 f_0 \oplus c_1 f_1 \oplus \dots \oplus c_{m-1} f_{m-1})\} \quad (1.4)$$

S-box có độ phi tuyến càng cao thì càng khó bị xấp xỉ bởi các hàm tuyến tính, giúp tăng khả năng kháng thám mã tuyến tính.

Tính chất đều vi sai. Tính chất này đo lường khả năng kháng lại thám mã vi sai. Một S-box S được gọi là δ – đều vi sai nếu với mọi sai biệt đầu vào $\alpha \neq 0$ và sai biệt đầu ra β , phương trình sau có tối đa δ nghiệm:

$$|\{x \in \mathbb{F}_2^n | S(x \oplus \alpha) \oplus S(x) = \beta\}| \leq \delta \quad (1.5)$$

Giá trị δ càng nhỏ, S-box càng có tính đều vi sai tốt, giúp triệt tiêu các đặc trưng vi sai có xác suất cao.

Bậc đại số. Bậc đại số của hàm logic g , ký hiệu $\deg(g)$, là bậc của đa thức cao nhất trong biểu diễn dạng chuẩn tắc đại số (ANF) của nó. Bậc đại số của S-box S là bậc lớn nhất trong số các hàm logic thành phần:

$$\deg(S) = \max_{i \in \{1, \dots, m\}} \deg(S_i) \quad (1.6)$$

khi đó ta có thể coi bậc của S-box S là bậc lớn nhất của tất cả các tổ hợp tuyến tính khác không của các hàm thành phần (vì phép cộng $\oplus$ không làm tăng bậc của hàm logic), tức là:

$$\deg(S) = \max_{b \in \mathbb{F}_2^m, b \neq 0} \deg(S_b); S_b = \sum_{i=1}^m b_i S_i \quad (1.7)$$

Bậc đại số cao giúp hệ mật kháng lại các cuộc tấn công vi sai bậc sao và các phương pháp phân tích cấu trúc đại số.

Bậc vào ra và số các quan hệ đại số. Gần đây, các nhà thiết kế hệ mật thường dùng một đại lượng khác đơn giản hơn đối với S-box S , đặc trưng cho khả năng kháng lại thám mã đại số, là bậc vào ra $\deg_{IO} S$. Cụ thể, khi xét đặc trưng này, chúng ta quan tâm đến các quan hệ đại số chính là các phương trình có các biến gồm các bit đầu vào và đầu ra. Bậc vào/ra của S là bậc nhỏ nhất của các quan hệ đại số $g(x_0, \dots, x_{n-1}; y_0, \dots, y_{m-1}) = 0$ thỏa mãn với mọi cặp (x, y) sao cho $y = S(x)$. Như vậy, bậc vào/ra của một S-box là bậc nhỏ nhất mà chúng ta có thể tìm được đối với các quan hệ đại số đúng cho S-box, nó giúp chúng ta đánh giá

được độ phức tạp trong quá trình giải hệ phương trình mô phỏng hệ mật. Điều này có nghĩa là S-box nào có bậc vào/ra càng nhỏ thì sẽ có những biểu diễn đại số có bậc nhỏ và do đó sẽ dễ giải hơn so với các S-box có bậc vào/ra cao. Ngoài ra, chúng ta cũng cần xét xem số lượng các phương trình có bậc như thế có nhiều không, hệ càng nhiều phương trình độc lập tuyến tính sẽ càng dễ giải hơn. Do đó, một S-box kháng lại thám mã đại số tốt càng có bậc vào/ra càng lớn càng tốt và số lượng các phương trình độc lập có bậc này phải càng nhỏ càng tốt.

Một số đại lượng khác. Ngoài một số đại lượng quan trọng ở trên, một đại lượng mật mã khác cho S-box được đặc trưng cho khả năng chống các thám mã đại số là độ dư thừa tuyến tính. Đại lượng này liên quan đến sự phụ thuộc giữa các hàm logic thành phần của S-box như sau:

Định nghĩa 1.1. [77] Cho S-box $S: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$. Ta gọi “S sở hữu độ dư thừa tuyến tính” khi tồn tại hai hàm logic thành phần của S tương đương affine, ngược lại ta gọi “S không sở hữu độ dư thừa tuyến tính”.

Định nghĩa 1.2. [1] Cho S-box $S: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$. Ta gọi “Độ dư thừa tuyến tính của S ” là đại lượng, kí hiệu là $LR(S)$, được xác định bởi số lượng các cặp hàm logic thành phần tương đương affine đôi một khác nhau. Đặc biệt, nếu S có $LR(S) = (2^n - 1)(2^n - 2)/2$ thì S được gọi là “sở hữu độ dư thừa tuyến tính toàn phần”.

1.1.2. Ma trận MDS và các đại lượng liên quan

Một tầng tuyến tính cung cấp tính khuếch tán bằng cách xáo trộn các bit đầu vào của một khối cố định và tạo ra khối đầu ra tương ứng. Có nhiều phương pháp được sử dụng để hiện thực tầng tuyến tính, bao gồm: ma trận MDS, mã tuyến tính, phép XOR và hoán vị bit ... Trong số đó ma trận MDS nhận được nhiều sự quan tâm trong nghiên cứu, bởi khả năng đảm bảo tính khuếch tán tối ưu. Có nhiều cách

để đánh giá tính khuếch tán của tầng tuyến tính: hiệu ứng thác [80]; tính thác chặt [81]; số nhánh [79]; số điểm bất động [79].

Hai tiêu chuẩn đầu tiên xem xét sự ảnh hưởng của 1 bit đầu vào lên các bit đầu ra, tính thác chặt xem xét sự ảnh hưởng của các bit đầu ra vào các bit đầu vào. Số nhánh là khái niệm được sử dụng rộng rãi nhất hiện nay, nó cho phép đánh giá một cách tổng thể tầng tuyến tính khi xem xét sự kháng lại thám mã vi sai và thám mã tuyến tính. Tiêu chuẩn cuối cùng nói đến tính chất điểm bất động của tầng tuyến tính, tiêu chuẩn này cụ thể như sau:

Số nhánh của tầng tuyến tính xây dựng trên cơ sở biến đổi tuyến tính L ký hiệu là $B(L)$, chính là số lượng nhỏ nhất các S-box tích cực trong 2 vòng mã liên tiếp của một mã pháp nào đó. Số nhánh được tính như sau. Gọi $X = (X_0, X_1, \dots, X_{d-1})$, trong đó $X_i \in \mathbb{F}_{2^n}$, là $d \times n$ bit dữ liệu đầu vào. Còn $\Gamma_X = (\Gamma_{X_0}, \Gamma_{X_1}, \dots, \Gamma_{X_{d-1}})$ là ký hiệu một véc tơ nhị phân có chiều dài d , với $\Gamma_{X_i} = 1$ khi $X_i \neq 0$, còn bằng 0 trong trường hợp còn lại. Ký hiệu $wt(\Gamma_X)$ là trọng số Hamming của Γ_X . Khi đó biểu thức của số nhánh là:

$$B(L) = \min \{wt(\Gamma_X) + wt(\Gamma_Y) : X \neq 0, Y = L(X)\} \quad (1.8)$$

Đối với một biến đổi tuyến tính bất kỳ, ta luôn có $B(L) \leq d + 1$.

Trong các thiết kế mật mã người thiết kế luôn hướng đến mục đích làm sao để có thể xây dựng được tầng tuyến tính có số nhánh cực đại.

Tiếp theo, chúng ta xem xét khái niệm điểm bất động của tầng tuyến tính (về sau gọi là điểm bất động của biến đổi tuyến tính). Trong luận án tiến sĩ tại trường đại học công nghệ Queensland năm 2010 [79], Muhammad Reza Z'aba đã xem xét khái niệm điểm bất động cho một biến đổi tuyến tính. Theo đó X là điểm bất động của biến đổi tuyến tính $L: \mathbb{F}_{2^n}^m \rightarrow \mathbb{F}_{2^n}^m$ nếu $X = L(X), X \in \mathbb{F}_{2^n}^m$. Có nghĩa là giá trị của nó không thay đổi dưới tác động của biến đổi tuyến tính này. Một

biến đổi tuyến tính $L: \mathbb{F}_{2^n}^m \rightarrow \mathbb{F}_{2^n}^m$ luôn có thể được biểu diễn bởi một ma trận không suy biến A có kích thước $m \times m$, các phần tử của ma trận này thuộc trường $\mathbb{F}_{2^n}$. Như vậy, số lượng điểm bất động của biến đổi tuyến tính trên chính là số nghiệm của hệ phương trình

$$(A - I) \cdot X = 0 \quad (1.9)$$

trong đó I là ma trận đơn vị kích thước $m \times m$. Từ đó số lượng điểm bất động cho biến đổi tuyến tính nói trên được tính theo công thức sau:

$$N_L = 2^{n(rank(A) - rank(A-I))} = 2^{n(n - rank(A-I))} \quad (1.10)$$

Do đó, số lượng các điểm bất động phụ thuộc vào hạng của ma trận $A - I$.

Bảng 1.1. Số nhánh và điểm bất động trong tầng khuếch tán của mã khối

Mã pháp	m	$B(L)$	$rank(A)$	$rank(A - I)$	N_L
AES	16	5	16	14	2^{16}
ARIA	16	8	16	7	2^{72}
PRESENT	64	2	64	40	2^{24}
Serpent	128	3	128	127	2
Kuznyechik	128	17	16	16	1

Trong Bảng 1.1 liệt kê số nhánh và điểm bất động của một số mã pháp dạng SPN như AES, ARIA, PRESENT và Serpent. Theo đó tầng khuếch tán trong AES tồn tại 2^{16} điểm bất động. Số nhánh của ma trận MDS trong AES là bằng 5. Thực nghiệm cho thấy không có mối liên hệ trực tiếp giữa số nhánh và số lượng điểm bất động. Một tầng tuyến tính có số nhánh lớn vẫn có thể tồn tại nhiều điểm bất động (như ARIA), dẫn đến nguy cơ bị tấn công dựa trên điểm bất động [83], [84].

Ngoài quan điểm về độ an toàn, khả năng cài đặt của một tầng tuyến tính được xem xét như một trong những tiêu chí hàng đầu trong thiết kế do khả năng cài đặt của nó liên quan trực tiếp đến tốc độ mã/dịch của một thuật toán. Như vậy việc xây dựng một tầng tuyến tính ngoài việc phải đảm bảo các tính chất mật mã

thì việc nghiên cứu khả năng cho phép cài đặt trên môi trường phần cứng và phần mềm của nó là vô cùng cần thiết và luôn có tính thời sự.

1.1.3. Độ an toàn của AES

Đến nay đã có nhiều tấn công lên AES trong đó phải kể đến tấn công tốt nhất có tên là *bicliques* lên đầy đủ các vòng AES [82]. Tuy nhiên tấn công biclique vẫn chỉ có thể về mặt lý thuyết bởi độ phức tạp tính toán của nó vẫn rất lớn. Bảng dưới đây là độ phức tạp tính toán để tấn công lên đầy đủ các vòng AES của [82]:

Bảng 1.2. Tấn công bicliques lên AES

AES	Độ phức tạp		
	Dữ liệu	Thời gian	Bộ nhớ
128	2^{88} CP	$2^{126.18}$	2^8
192	2^{88} CP	$2^{189.74}$	2^8
256	2^{40} CP	$2^{254.42}$	2^8

CP – chosen plaintext

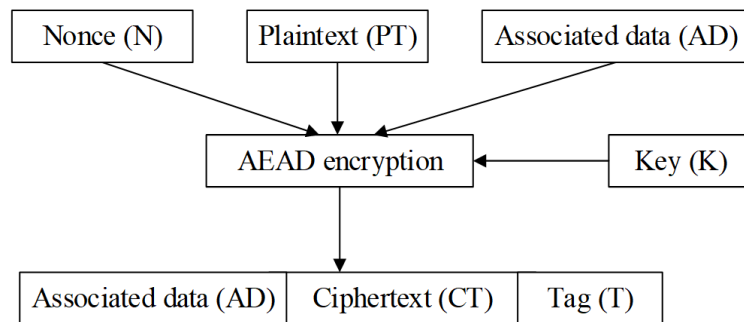
1.2. Thuật toán mã hóa Ascon

1.2.1. Thuật toán mã hóa có xác thực (AEAD)

Một trong những thách thức đối với bảo mật mạng IoT là sự đa dạng của các thiết bị, môi trường triển khai. Bài toán bảo mật cần phải đảm bảo từ các thiết bị tài nguyên hạn chế (edge devices) cho đến các thiết bị tốc độ cao (IoT gateway, cloud servers). Chính vì lý do đó đã có rất nhiều nghiên cứu để phát triển các thuật toán mật mã phù hợp cho mạng IoT [13]. Một trong những giải pháp gần đây là sử dụng Authenticated Encryption with Associated Data (AEAD), kết hợp các dịch vụ mật mã như bí mật, toàn vẹn và xác thực lại thành một thuật toán. Tháng 8 năm 2018, NIST đã khởi động một quá trình tiêu chuẩn hóa thuật toán mật mã hạng nhẹ (Lightweight Cryptography). Hầu hết các thuật toán hạng nhẹ tham gia đều là AEAD và hàm băm. Trong số 56 ứng cử viên ở vòng đầu tiên, 32 thuật toán được chọn vào vòng hai. Tháng 3 năm 2021, NIST thông báo 10 ứng cử viên lọt

vào vòng chung kết và Ascon là một trong số đó. Tháng 2 năm 2023, NIST công bố Ascon được chọn làm tiêu chuẩn thuật toán mã hoá xác thực hạng nhẹ.

Các thuật toán mã hóa truyền thống chỉ cung cấp tính bảo mật, để cung cấp tính toàn vẹn/xác thực các thuật toán này cần kết hợp với các dịch vụ khác. Điều này dẫn đến sự phát triển của khái niệm mã hóa xác thực (AE). Mã hoá có xác thực kết hợp tính bí mật và toàn vẹn trong một thuật toán duy nhất, hỗ trợ xác thực cả dữ liệu liên kết (AD). Quy trình AEAD nhận đầu vào là khoá K , nonce N , dữ liệu liên kết AD và bản rõ PT để sinh ra bản mã CT cùng thẻ xác thực T . Tại bước giải mã, thẻ T được kiểm tra trước tiên; bản mã chỉ được giải mã nếu thẻ T hợp lệ, ngược lại sẽ bị huỷ bỏ để đảm bảo an toàn.



Hình 1.2. Quá trình mã hóa AEAD

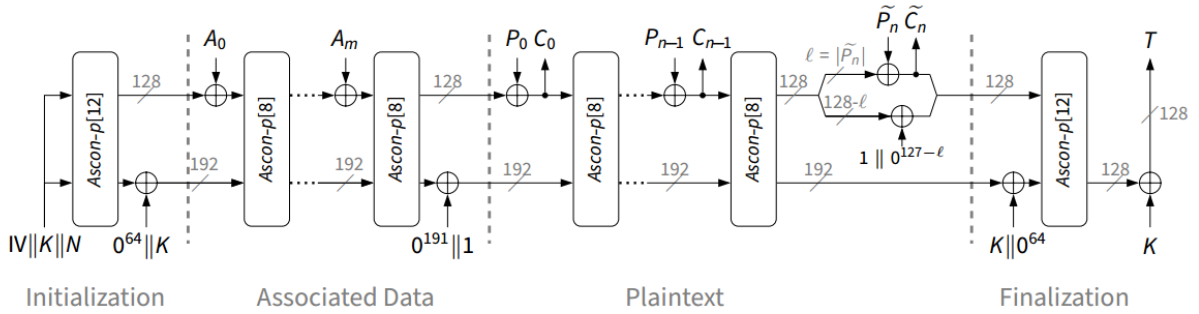
1.2.2. Thuật toán Ascon

Thuật toán Ascon-AEAD128 là một cơ chế mã hóa xác thực với dữ liệu liên kết (Authenticated Encryption with Associated Data – AEAD) được đặc tả trong chuẩn NIST SP 800-232 [6]. Ascon-AEAD128 được xây dựng theo mô hình sponge, sử dụng một hoán vị trạng thái cố định và được thiết kế nhằm đạt được sự cân bằng giữa bảo mật, hiệu năng và khả năng triển khai hiệu quả trên cả phần mềm và phần cứng.

Ascon-AEAD128 xử lý dữ liệu theo các khối 128 bit và áp dụng 8 vòng hoán vị trong các giai đoạn xử lý chính. Việc lựa chọn kích thước khối và số vòng

hoán vị nhằm cung cấp thông lượng cao, đồng thời duy trì mức độ an toàn phù hợp với các yêu cầu mã hóa có xác thực. Với đặc tính thông lượng tốt và cấu trúc gọn nhẹ, Ascon-AEAD128 phù hợp cả với các hệ thống yêu cầu hiệu năng cao cũng như trên các môi trường tài nguyên hạn chế.

Quá trình mã hóa hoặc giải mã có xác thực của Ascon bao gồm bốn giai đoạn: Khởi tạo, Dữ liệu liên kết (AD), Xử lý bản rõ/bản mã và Kết thúc.

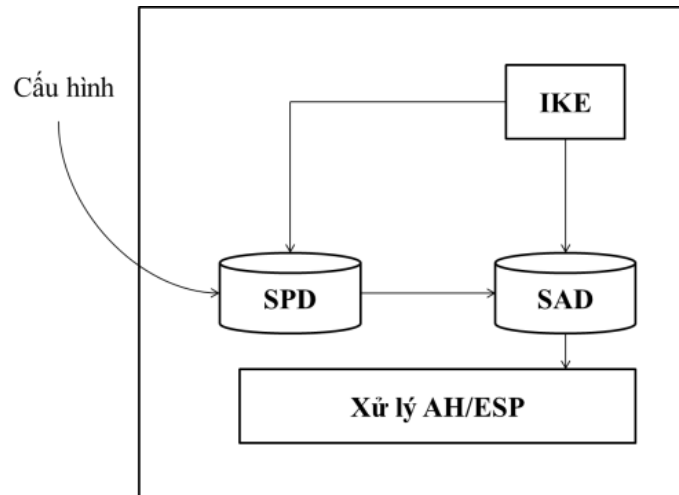


Hình 1.3. Quá trình mã hóa Ascon [6]

1.3. Giao thức bảo mật IPSec

Giao thức IPSec là một bộ giao thức VPN (Virtual Private Network) phổ biến. Phiên bản mới nhất của chuẩn IPSec được công bố trong RFC 4301. IPSec có thể được cài đặt theo các cách khác nhau trên để thực thi trên máy host, router, firewall, gateway an toàn hoặc như một thiết bị bảo mật độc lập [57]. Hầu hết các hệ điều hành Windows, Linux ngày nay đều hỗ trợ mặc định IPSec trong nhân. Các thành phần của công nghệ IPSec có thể thấy ở Hình 1.4. Trong đó SPD (Security Policy Database) là chính sách an toàn quyết định luồng dữ liệu nào cần được bảo mật. Các trường như địa chỉ IP nguồn/đích, protocol, cổng TCP/UDP sẽ được tìm kiếm để quyết định. IPSec cho phép ba quyết định: DROP, BYPASS, và PROTECT. Các thông tin về kết nối IPSec như giao thức, thuật toán, khóa, số sequence number được lưu trong cơ sở dữ liệu SAD (Security Association Database). Khi hai bên muốn thiết lập kết nối IPSec, các tham số cho phiên kết

nối cần phải được thỏa thuận. Các bên cũng cần phải được xác thực. Khi đó giao thức IKE (Internet Key Exchange) sẽ được sử dụng. IPSec có thể sử dụng ESP (Encapsulating Security Payload) hoặc AH (Authentication Header) để cung cấp các dịch vụ bảo mật. AH chỉ bảo vệ tính xác thực, toàn vẹn của gói IP, ESP hỗ trợ đầy đủ các dịch vụ cho gói tin IP bao gồm mã hóa, xác thực, toàn vẹn.



Hình 1.4. Các thành phần chính của IPSec

1.3.1. Giao thức ESP

Phiên bản hiện tại của ESP và IPSec là phiên bản 3 đã được ban hành trong RFC 4301 và 4303. ESP hỗ trợ hai chế độ: transport và tunnel. Trong chế độ tunnel (Hình 1.5), một gói IP mới được tạo ra từ gói tin IP nguyên gốc bằng cách: 1) chèn ESP header và trailer vào gói tin IP ban đầu; 2) mã hóa gói IP ban đầu và ESP trailer; 3) tính giá trị kiểm tra toàn vẹn ICV (Integrity Check Value) cho ESP header và các dữ liệu mã hóa; 4) chèn ICV vào cuối của gói tin; 5) thêm IP header mới cho gói tin. Chế độ ESP tunnel được sử dụng để triển khai theo mô hình VPNs gateway-to-gateway hoặc remote access.

Transport mode không tương thích với NAT. Chẳng hạn như trong gói TCP, TCP checksum được tính cho cả TCP và IP header. Nếu NAT được sử dụng khi đó địa chỉ nguồn hoặc đích của IP header sẽ thay đổi, NAT cần tính lại TCP checksum. Tuy nhiên do TCP header đã được mã bên trong ESP nên NAT sẽ thất bại. Do đó trên thực tế sử dụng ESP ở chế độ tunnel mode sẽ đáp ứng đầy đủ các tính năng và dịch vụ bảo mật.

Cấu trúc gói ESP

ESP chèn header và trailer vào đầu và cuối payload của các gói tin như Hình 1.7 dưới đây.

ESP header	Security Parameters Index (SPI)	
	Sequence Number	
Payload	Initialization Vector (tùy chọn)	
	Dữ liệu (thay đổi)	
ESP trailer	Padding (0-255 bytes)	
	Padding length	Next header
ESP trailer	Integrity Check Value (ICV)	

Hình 1.7. Cấu trúc gói ESP

ESP header gồm có hai trường chính:

- **SPI.** Mỗi IPsec SA sẽ chứa một giá trị SPI riêng. Giá trị SPI được sử dụng cùng với địa chỉ IP đích để xác định SA và khóa nào cần được sử dụng.
- **(Extended) Sequence Number.** Mỗi gói tin sẽ được gán một số có giá trị Sequence Number tăng lên, và chỉ có các gói tin có giá trị Sequence Number nằm

trong cửa sổ trượt mới được chấp nhận. Điều này sẽ chống lại các tấn công phát lại vì các gói sử dụng lại sẽ có cùng giá trị Sequence Number. Việc sử dụng Sequence Number cũng chống các tấn công DOS vì các gói tin cũ sẽ có giá trị Sequence Number nằm ngoài cửa sổ trượt sẽ bị loại bỏ trước khi thực hiện bất kỳ hoạt động khác. Phiên bản 2 của IPSec định nghĩa Sequence Number là giá trị 32 bit. Tuy nhiên ngày nay phần cứng có thể truyền được 100 Gbps, hoặc khoảng 150 triệu gói tin mỗi giây. Điều này có nghĩa 32 bit Sequence Number sẽ bị tràn trong 30 giây. Việc trao đổi khóa lại sau mỗi 30 giây là không thực tế, do đó phiên bản IPSec-v3 sử dụng ESN. Giá trị ESN có kích thước 64 bit và để tương thích với định dạng ESP cũ, chỉ có 32 bit thấp của ESN được truyền đi.

Phần tiếp theo của ESP là payload. Nó bao gồm phần dữ liệu mã và IV ở dạng rõ. IV được sử dụng trong quá trình mã hóa để chống lại thám mã. Giá trị IV thay đổi cho mỗi gói tin, do đó nếu hai gói tin có dữ liệu giống nhau thì IV sẽ giúp việc mã hóa hai gói tin là khác nhau.

Để ẩn các thông tin về kích thước, tần suất gửi dữ liệu qua IPSec, TFC (Traffic Flow Confidentiality) padding có thể được sử dụng. TFC padding được chèn vào dữ liệu rõ trước khi mã.

Phần cuối cùng là ESP trailer, nó bao gồm các trường sau:

- **ESP Padding.** Gói ESP có thể được padding trong trường hợp mã khối cần chẵn khối. Padding cũng cần thiết để ESP trailer là chẵn 4 byte.
- **ESP Padding Length.** Trường này chỉ định độ dài của padding theo bytes.
- **Next Header.** Trong trường hợp tunnel mode, giá trị của Next Header là 4, nghĩa là gói tin IP được bọc trong gói IP khác. Trong transport mode, Next Header có thể là 6 (TCP) hoặc 17 (UDP).

- **ICV**. Giá trị này được sử dụng để kiểm tra tính toàn vẹn của dữ liệu mã. Đối với AES-GCM, kích thước của ICV có thể là 8, 12 hoặc 16 byte. NIST khuyến cáo sử dụng kích thước 16 cho ICV.

UDP và ESP Encapsulation

ESP packet không thể vượt được các thiết bị NAT. Giao thức ESP cũng không có cổng, do đó NAT router thường sẽ loại bỏ các gói ESP do nó chỉ xử lý các gói UDP hoặc TCP. Để giải quyết vấn đề vượt NAT, các gói ESP có thể được đóng gói UDP (UDP-ESP). Các thiết bị NAT có thể sửa địa chỉ IP của gói UDP và theo dõi các thiết bị phía sau cổng UDP. Khi IKE phát hiện thiết bị nằm sau NAT, nó sẽ sử dụng cổng UDP 4500. UDP-ESP cũng sử dụng cổng 4500 để đảm bảo rằng thiết bị NAT có ánh xạ NAT duy nhất cho toàn bộ luồng traffic (ESP và IKE). Để phân biệt gói UDP-IKE với UDP-ESP, 4 byte chứa giá trị 0 sẽ được thêm vào trước IKE header do giá trị SPI là 4 byte và không thể bằng 0.

1.3.2. Các thành phần mật mã được sử dụng trong IPSec

ESP sử dụng mật mã khóa đối xứng để mã hóa gói IPSec. Do đó, mỗi bên kết nối đều phải có cùng một khóa để mã hóa và giải mã gói tin. Khi mã hóa, dữ liệu được chia thành các khối nhỏ và thực hiện các vòng mã sử dụng khóa lên dữ liệu. Thuật toán mã hóa kiểu này được gọi là thuật toán mã khối. Sau khi mã hóa được thực hiện, để bảo vệ tính toàn vẹn cho gói tin thuật toán MAC sẽ được sử dụng với một khóa chung lên dữ liệu. Giá trị MAC được chèn vào cuối gói tin và gửi cho bên nhận. Bên nhận tính lại giá trị MAC và so sánh xem hai giá trị để xác định gói tin có bị thay đổi. IPSec thường sử dụng thuật toán HMAC để bảo vệ tính toàn vẹn. Thuật toán mã khối Triple DES đã bị lỗi thời từ năm 2019 và không còn được sử dụng từ sau năm 2023. Các thuật toán HMAC-MD5 và HMAC-SHA-1 cũng không còn được chấp nhận bởi NIST. Các khuyến nghị mới nhất của NIST đối với mã khối là sử dụng AES kích thước khóa 128, 192, 256 bit ở chế độ CBC,

CTR, GCM, CCM. Đối với thuật toán toàn vẹn là HMAC-SHA256, HMAC-SHA384, HMAC-SHA-512, AES-GMAC.

Mã hóa và toàn vẹn cần hai thuật toán và hai khóa khác nhau. Thuật toán AEAD xuất hiện trong những năm gần đây như AES-GCM có thể kết hợp hai thuật toán lại với chỉ một khóa và giúp tăng tốc rất nhiều. Nó cũng cho phép khả năng xử lý liên tục khi xảy ra lỗi, dẫn đến quy trình xử lý lỗi tốt hơn và ít bị tấn công theo thời gian. Trong thuật toán AEAD, giá trị nonce cần phải là duy nhất cho mỗi quá trình mã hóa với cùng khóa. Giá trị này cũng không nhất thiết phải ngẫu nhiên. Giá trị nonce được tạo ra bởi IKE bao gồm các thành phần không gửi đi (salt) và thành phần gửi đi (IV). Thành phần không gửi đi được dẫn xuất từ PRF trong quá trình trao đổi khóa DH tương tự như tạo ra khóa mã hóa. Giá trị này không gửi trên đường truyền. Thành phần gửi đi IV thường là một bộ đếm. Việc sử dụng lại IV với cùng một khóa sẽ phá vỡ tính an toàn. Do đó các thuật toán này cần phải được sử dụng với trao đổi khóa tự động IKE và không được cấu hình tĩnh, thủ công.

1.3.3. Một số tấn công và kết quả đánh giá về bảo mật và an ninh, an toàn của giao thức IPSec

Mặc dù IPsec được thiết kế một cách an toàn từ đầu, nhưng đến nay IPsec vẫn có thể bị các tấn công cài đặt với các cấu hình cụ thể. Bellovin [72] là người đầu tiên chỉ ra việc thiếu cơ chế toàn vẹn trong phiên bản thứ nhất của giao thức mã hóa ESP sẽ dẫn đến các vấn đề về an toàn. Các tấn công của Bellovin chỉ dừng lại ở lý thuyết và không khả thi trên thực tế. Tuy vậy các tấn công này cũng cho thấy được sự nguy hiểm của việc mã hóa mà không có tính toàn vẹn. Các tấn công thực tế hơn lên các cài đặt chỉ mã hóa của IPsec ESP trong nhân Linux được thực hiện bởi [73], [74]. Nhờ sử dụng kỹ thuật tấn công tiên đoán đệm (padding oracle), các tác giả đã có thể phá vỡ các cài đặt chỉ mã hóa của IPsec ESP. Những tấn công

này yêu cầu IPSec sử dụng cấu hình ESP chỉ mã hóa ở chế độ CBC. Kẻ tấn công sẽ chỉnh sửa các trường của IP Header (Option và Protocol) để có thể kích hoạt các thông điệp báo lỗi ICMP sau đó dùng cho tấn công tiên đoán đệm.

Một tấn công khác được thực hiện bởi [76] đối với IPSec khi nó sử dụng kiểu xác thực là: MAC-then-Encrypt (thực hiện MAC trước rồi mới mã hóa). Trong trường hợp này IPSec được sử dụng giữa hai gateways và cấu hình để sử dụng mã khối ở chế độ CBC. Kẻ tấn công sẽ thực hiện một tấn công tiên đoán ESP trailer. Bộ tiên đoán sẽ xác định xem ESP trailer có được định dạng đúng không. Tấn công MAC-then-Encrypt cho thấy việc cấu hình sử dụng AH kết hợp ESP là không an toàn.

Các vấn đề khác có thể gây ra các tấn công cài đặt lên IPSec có thể kể đến:

- Một số cài đặt IPSec lưu khóa chia sẻ trước (PSKs) ở dạng rõ trên hệ thống. Bất kỳ người dùng nào có quyền truy cập vào hệ thống cũng có thể đọc được các khóa này. Do đó nếu hệ thống không có một cơ chế kiểm soát truy cập an toàn thì cài đặt dạng này cần phải được tránh.
- IPSec cho phép một số luồng dữ liệu đi qua mà không cần bảo vệ, chẳng hạn như các gói broadcast, IKE và Kerberos. Kẻ tấn công có thể lợi dụng các thông tin này để gửi các gói tin độc hại vượt qua chính sách lọc gói của IPSec. Do đó cần phải giám sát các luồng dữ liệu không cần bảo vệ khi đi qua IPSec, ví dụ như sử dụng hệ thống IDS/IPS để cảnh báo khi có các gói tin không được bảo vệ đi qua.
- Các lỗ hổng về cài đặt IPSec mà được phát hiện thông qua các cơ sở dữ liệu National Vulnerability Database (NVD) hoặc Common Vulnerabilities and Exposures (CVE).
- Mặc dù chế độ CBC vẫn được khuyến nghị sử dụng bởi NIST, tuy nhiên các tấn công cài đặt lên IPSec ở trên (tấn công ESP chỉ mã hóa và MAC-then-

Encrypt) đều lợi dụng cơ chế đệm của chế độ CBC. Do đó việc sử dụng AES ở chế độ CBC có thể coi là không còn an toàn.

1.4. Phân tích các công trình nghiên cứu công bố gần đây và định hướng nghiên cứu của đề tài luận án

Để xây dựng một hệ thống bảo mật mạng tốc độ cao, hiệu quả và tin cậy trên nền tảng phần cứng, cần có cái nhìn toàn diện từ cấp độ thành phần cơ sở đến kiến trúc hệ thống. Trong phần này, luận án sẽ đi sâu phân tích tình hình nghiên cứu trong và ngoài nước theo ba cấp độ chính: (1) Các thành phần mật mã S-box và ma trận MDS; (2) Các kiến trúc phần cứng cho thuật toán AES và thuật toán mã hoá xác thực hạng nhẹ Ascon, tập trung vào các kỹ thuật tối ưu hóa thông lượng, độ trễ; và (3) Các kiến trúc giao thức bảo mật mạng IPSec, nơi tích hợp các thuật toán trên vào môi trường truyền thông thực tế. Thông qua việc đánh giá ưu, nhược điểm của các công trình liên quan, NCS sẽ xác định các vấn đề tồn tại và khoảng trống nghiên cứu, từ đó làm cơ sở đề xuất các giải pháp và định hướng cụ thể cho luận án.

1.4.1. Phân tích các công trình về thành phần mật mã

S-box là một thành phần cốt lõi trong các thuật toán mã khối. Chúng được sử dụng rộng rãi trong nhiều chuẩn mã hóa như AES, DES, Kuznyechik,... Việc thiết kế và lựa chọn S-box thường được dựa trên các tiêu chí như mức độ an toàn, hiệu năng xử lý và chi phí tài nguyên. Nhiều nghiên cứu đã tập trung vào việc cải thiện hiệu suất và tối ưu hóa tài nguyên cho các S-box đã được tiêu chuẩn hóa, đặc biệt là S-box của AES khi triển khai trên phần cứng. Một trong những giải pháp phổ biến nhất là sử dụng bảng tra chứa toàn bộ 256 giá trị của S-box [14]. Phương pháp này có ưu điểm dễ triển khai, nhưng đánh đổi bằng việc tiêu tốn nhiều tài nguyên phần cứng cũng như có độ trễ cao nếu sử dụng RAM, ROM. Bên cạnh đó, một số giải pháp không sử dụng bảng tra tập trung vào việc tối ưu phép nghịch

đảo trên trường hữu hạn, tiêu biểu như sử dụng trường ghép (Composite Field) [15], các kỹ thuật tối ưu hóa logic [16].

Về mặt an toàn, S-box của mã khối AES được xây dựng dựa trên ánh xạ nghịch đảo [12]. Nó có độ phi tuyến cao nhưng lại không đạt được các tính chất mật mã quan trọng như bậc vào ra, độ dư thừa tuyến tính [17]. Do đó, trong những năm gần đây, nhiều công trình đã đề xuất các phương pháp sinh S-box mới nhằm nâng cao độ an toàn trong khi vẫn đảm bảo khả năng cài đặt hiệu quả trên phần cứng. Tuy nhiên, việc thiết kế hoặc tìm kiếm các S-box có kích thước lớn (8-bit trở lên) vừa có độ an toàn cao, vừa có độ hiệu quả cao vẫn là một thách thức mở. Các hướng tiếp cận để sinh S-box kích thước lớn hiện nay có thể được chia thành bốn nhóm chính: dựa trên đại số, sinh giả ngẫu nhiên, các kỹ thuật heuristic hoặc kết hợp từ các cấu trúc nhỏ hơn.

Các phương pháp dựa trên đại số thường sử dụng các phép toán trong trường hữu hạn như ánh xạ nghịch đảo, lũy thừa để xây dựng S-box lớn [12], [19], [22], [24]. Phương pháp này có ưu điểm là dễ phân tích, đảm bảo được các chỉ số mật mã như độ phi tuyến cao, bậc đại số lớn. Các S-box của AES được xây dựng với ánh xạ nghịch đảo được affine hóa đầu vào đầu ra. Tuy nhiên, nhược điểm của các S-box dạng này là có cấu trúc toán học rõ ràng khiến mã khối dễ tổn thương đối với các tấn công dựa trên cấu trúc điển hình như tấn công đại số.

Ngược lại, phương pháp sinh giả ngẫu nhiên tạo ra S-box bằng cách lựa chọn ngẫu nhiên hoặc giả ngẫu nhiên các ánh xạ và kiểm tra tính chất mật mã của chúng [18], [21], [23]. Ưu điểm của phương pháp này là tạo ra các S-box không có cấu trúc rõ ràng, từ đó tăng khả năng chống lại các tấn công dựa trên cấu trúc như đại số hay cửa hậu (trapdoor). Tuy nhiên, nhược điểm lớn là hiệu suất sinh rất thấp: số lượng S-box thỏa các tiêu chí cao về độ phi tuyến, bậc đại số hoặc độ dư thừa tuyến tính là cực kỳ ít trong không gian tìm kiếm, khiến việc tìm kiếm tốn

nhiều thời gian và tài nguyên tính toán. Ngoài ra, các S-box được tạo ngẫu nhiên thường không hiệu quả với phần cứng.

Trong khi đó, các kỹ thuật heuristic như hill climbing [26], thuật toán tối ưu hóa dựa trên hành vi của đàn ong [20], hay phương pháp tối ưu hóa nhiều lớp [25]. Phương pháp này cho phép sinh ra các S-box gần tối ưu với các đặc tính mật mã tốt và giảm thiểu cấu trúc dễ khai thác, đồng thời có thể kết hợp với các ràng buộc thiết kế để đảm bảo hiệu quả cài đặt phần cứng. Tuy nhiên, tính tối ưu chỉ mang tính tương đối, phụ thuộc vào hàm mục tiêu, và có thể không đảm bảo toàn vẹn nếu không được kiểm tra toàn diện.

Tóm lại, như đã phân tích ở trên, xây dựng các S-box song ánh có tính chất mật mã tốt nhanh chóng trở thành một nhiệm vụ khó khi kích thước của chúng tăng. Khi số các biến đầu vào n tăng lên 1 thì số các hàm logic trong không gian sẽ tăng thêm 2^{2^n} . Đó là lý do tại sao phương pháp sinh giả ngẫu nhiên gặp khó khăn khi n lớn. Vì thế, đây là một thách thức để cải tiến các kết quả cho các S-box kích thước 8 bit và thậm chí lớn hơn. Gần đây, các nghiên cứu tập trung vào phương pháp sinh S-box kết hợp từ các cấu trúc nhỏ hơn, cho phép kiểm soát tốt các đặc tính mật mã trong khi vẫn đảm bảo khả năng triển khai hiệu quả trên phần cứng. Điển hình là cấu trúc Butterfly, cho phép tạo các hoán vị $\mathbb{F}_{2^n}^2$ từ các hoán vị $\mathbb{F}_{2^n}$ [28],[29] được đưa ra bởi Leo Perrin và cộng sự.

Định hướng của luận án: Từ các phân tích trên, NCS lựa chọn tiếp cận dựa trên cấu trúc, cụ thể là cấu trúc Butterfly để xây dựng S-box từ các thành phần nhỏ hơn nhằm cân bằng giữa độ an toàn và khả năng hiện thực trên phần cứng.

Trong thiết kế thuật toán mã khối, bên cạnh vai trò then chốt của S-box, tầng tuyến tính cũng đóng vai trò đặc biệt quan trọng khi ảnh hưởng trực tiếp đến hiệu năng tổng thể cũng như mức độ khuếch tán của thuật toán. Vì vậy, việc thiết

kế các tầng tuyến tính – nhất là các ma trận MDS - luôn nhận được sự quan tâm rất lớn từ cộng đồng nghiên cứu mật mã. Nhiều công trình đã được công bố nhằm đề xuất các dạng ma trận MDS an toàn, hiệu quả, tiêu biểu như ma trận MDS dịch vòng [31], ma trận Hadamard [32], hay các ma trận MDS xây dựng từ lũy thừa của ma trận đồng hành (companion matrix) [33]. Một số nghiên cứu gần đây chủ yếu tập trung vào việc cải thiện những hạn chế cố hữu của ma trận dịch vòng trong AES, chẳng hạn như sự tồn tại của nhiều điểm bất động (fixed points), hoặc hiệu suất giải mã thấp hơn đáng kể so với quá trình mã hóa [79]. Trong đó tìm ra các dạng ma trận MDS đối xứng (involutory MDS matrices) hướng tới các thiết bị hạng nhẹ có tài nguyên phần cứng hạn chế. Tuy nhiên, trong các hệ thống hiệu năng cao, các ma trận đối xứng hạng nhẹ này không đáp ứng được yêu cầu về tốc độ xử lý. Để khắc phục sự bất cân xứng giữa mã hóa và giải mã, nhiều thuật toán mã khối hiện đại lựa chọn các chế độ hoạt động song song như CTR hoặc GCM - vốn chỉ cần sử dụng ma trận mã hóa cho cả hai quá trình - từ đó khai thác tối đa hiệu quả tính toán trên phần cứng. Bằng cách áp dụng các chế độ hoạt động này, ma trận mã hóa dạng dịch vòng vẫn cho thấy hiệu suất vượt trội khi triển khai trên các nền tảng phần cứng hiện đại, nhờ cấu trúc đơn giản và khả năng hỗ trợ thực thi song song hiệu quả.

Định hướng của luận án: NCS tập trung nghiên cứu và lựa chọn ma trận MDS dạng dịch vòng hoặc tựa dịch vòng cho tầng tuyến tính, nhằm đạt được sự cân bằng giữa hiệu năng xử lý và mức độ an toàn trên phần cứng.

1.4.2. Phân tích các công trình về kiến trúc phần cứng tốc độ cao cho thuật toán mã hóa

Hiệu năng của các kiến trúc trên phần cứng không chỉ phụ thuộc vào bản chất thuật toán mã hóa mà còn chịu ảnh hưởng mạnh mẽ từ nhiều yếu tố kỹ thuật, bao gồm công nghệ phần cứng sử dụng, độ rộng datapath, kỹ thuật cài đặt, cũng

như các tiêu chí đánh giá hiệu quả hệ thống. Việc lựa chọn và kết hợp các yếu tố này một cách hợp lý đóng vai trò then chốt trong việc tối ưu hóa tốc độ xử lý, hiệu suất tài nguyên và mức tiêu thụ điện năng của kiến trúc.

Nhiều nghiên cứu gần đây đã đề xuất các kiến trúc trên FPGA bởi nó cho thấy các ưu điểm so với công nghệ ASIC [10], như chi phí, mức độ phức tạp để chuyển thiết kế từ ý tưởng lên phần cứng là thấp hơn nhiều so với ASIC. Trong khi đó, độ rộng datapath lại liên quan trực tiếp đến số lượng dữ liệu có thể xử lý cùng một lúc: 8 bit, 32 bit hoặc 128 bit dữ liệu mỗi xung nhịp đồng hồ (clock). 8-bit datapath cần ít tài nguyên, mức độ tiêu thụ điện năng thấp, nhưng lại có số lượng xung nhịp đồng hồ lớn nhất và tốc độ thấp nhất. Trong khi đó 128-bit datapath có thể xử lý nhiều dữ liệu hơn trong một xung nhịp đồng hồ dẫn đến tốc độ cao hơn nhưng sẽ tiêu tốn nhiều tài nguyên hơn. Các kỹ thuật cài đặt (lập lại, mở vòng lặp, kỹ thuật đường ống) cũng là yếu tố ảnh hưởng trực tiếp đến hiệu suất, và tài nguyên. Trên thực tế, việc lựa chọn kỹ thuật phù hợp phụ thuộc vào các tiêu chí thiết kế cụ thể và thường đi kèm với các quyết định đánh đổi (trade-off). Ví dụ, để đạt được thông lượng cao, thiết kế có thể phải chấp nhận tiêu tốn nhiều tài nguyên hơn. Bên cạnh đó, do các nghiên cứu thường được triển khai trên các nền tảng phần cứng khác nhau, việc đánh giá hiệu quả kiến trúc cần được thực hiện một cách khách quan và công bằng. Trong lĩnh vực triển khai trên FPGA, một trong những tiêu chí đánh giá phổ biến và mang tính công bằng cao là tỷ lệ giữa thông lượng và chi phí cài đặt (Mbps trên số lượng LUT hoặc Slice), giúp phản ánh hiệu quả sử dụng tài nguyên của thiết kế một cách rõ ràng.

Do đó trong luận án này, tiêu chí Mbps/LUT hoặc Mbps/Slice được sử dụng thống nhất để đánh giá hiệu quả cài đặt của các kiến trúc trên FPGA.

1.4.2.1. Các công trình về kiến trúc AES tốc độ cao trên FPGA

Để đạt được thông lượng tối đa, kiến trúc của các thuật toán mã hóa cần xử lý trọn vẹn một khối dữ liệu trong mỗi chu kỳ xung nhịp. Trong các kỹ thuật hiện hành (lập lại, mở vòng lặp, đường ống), chỉ có kỹ thuật đường ống (pipeline) đáp ứng được yêu cầu khắt khe này. Do đó, bài toán tối ưu hóa kiến trúc đường ống cho AES trở thành trọng tâm nghiên cứu, xoay quanh hai thách thức chính: (1) rút ngắn đường tới hạn (critical path) và (2) cân bằng giữa độ trễ và hiệu suất thông qua việc bố trí hợp lý các thanh ghi trung gian.

Nhiều nghiên cứu đã đề xuất các kiến trúc đường ống đa giai đoạn nhằm giải quyết bài toán trên. Liu [35] giới thiệu kiến trúc 5 giai đoạn với S-box dạng bảng tra (LUT) và lược đồ khóa mới, tuy nhiên thiết kế này tiêu tốn tài nguyên lớn và hiệu quả chưa cao. Oukili [37] đề xuất kiến trúc 8 tầng để tăng tốc độ, nhưng việc sử dụng tới 5 thanh ghi trung gian cho S-box khiến độ trễ tăng đáng kể. Tương tự, Baby Chellam [38] áp dụng mở vòng lặp cho S-box và tinh chỉnh MixColumns, song việc chèn quá nhiều tầng đường ống vào khối MixColumns lại làm giảm khả năng đáp ứng thời gian thực.

Gần đây, các nghiên cứu tiếp tục mở rộng theo hướng chuyên biệt hóa cho từng ứng dụng, từ hệ thống 5G tốc độ cao đến thiết bị nhúng tài nguyên hạn chế. Visconti [39] triển khai AES-128 trên FPGA SoC ZCU102 với kiến trúc đường ống toàn phần, đạt thông lượng ấn tượng nhưng phải đánh đổi bằng chi phí tài nguyên lớn cho khối giải mã. Ngược lại, Kumar [40] và Priya [41] tập trung giảm diện tích bằng cách thay thế LUT bằng mạch logic tổ hợp cho S-box. Dù tiết kiệm tài nguyên, thiết kế của Kumar lại bị giới hạn về tần số hoạt động, trong khi thiết kế của Priya vẫn tiêu tốn nhiều tài nguyên do áp dụng đường ống cho cả S-box và MixColumns. Rahim [42] đề xuất kỹ thuật đường ống một phần cho 5G, đạt tốc

độ cao nhưng lại vấp phải vấn đề độ trễ lớn, hạn chế khả năng ứng dụng trong các hệ thống yêu cầu phản hồi tức thời.

Tại Việt Nam, Đồng Phạm Khôi [2] đã công bố một thiết kế ASIC CMOS 45nm ấn tượng dành cho IoT, đạt băng thông 111,3 Gbps với các chỉ số hiệu quả năng lượng và diện tích vượt trội. Tuy nhiên, cần phân biệt rõ rằng công trình này dựa trên công nghệ ASIC chuyên dụng. Trong khi đó, luận án tập trung vào nền tảng FPGA nhằm khai thác tính linh hoạt, khả năng tái cấu hình và triển khai nhanh, đồng thời vẫn đảm bảo thông lượng hàng trăm Gbps với chi phí hợp lý.

Định hướng của luận án: Tổng hợp lại, mặc dù các kiến trúc AES trên FPGA hiện nay đã đạt được những bước tiến về tốc độ, đa số vẫn chưa giải quyết triệt để bài toán đánh đổi giữa tài nguyên, độ trễ và hiệu suất. Một thiết kế thực sự cân bằng được cả ba yếu tố này vẫn là một thách thức mở. Xuất phát từ thực tế đó, luận án đề xuất một phương pháp thiết kế kiến trúc AES mới, hướng tới mục tiêu hiệu suất cao, độ trễ thấp với chi phí tài nguyên hợp lý.

1.4.2.2. Các công trình về kiến trúc Ascon tốc độ cao trên FPGA

Các nghiên cứu tiên phong như [44] và [45] đã triển khai Ascon trên FPGA với kỹ thuật mở vòng lặp, sử dụng chuẩn CAESAR API. Các đánh giá so sánh [46], [47] cho thấy Ascon có tỷ lệ thông lượng/diện tích (TPA) cao. Bên cạnh đó, nhiều kỹ thuật chuyên sâu đã được đề xuất như: tái cấu hình động (DPR) [48], tối ưu cho IoT/AI [49], [50], [51], cơ chế phát hiện lỗi [52], và vi xử lý tiết kiệm năng lượng [53].

Tuy nhiên, do đặc tính phản hồi (feedback) trong cấu trúc thuật toán, kỹ thuật đường ống truyền thông không thể áp dụng trực tiếp cho Ascon. Các thiết kế hiện tại [44]–[53], chưa đạt được kiến trúc xử lý trọn vẹn Ascon trong một chu kỳ xung nhịp. Hơn nữa, việc phụ thuộc vào API của các cuộc thi mật mã hạng nhẹ

khiến chúng khó tích hợp vào các hệ thống thực tế như IPSec, điển hình là hiện chưa có công trình nào đánh giá Ascon trong ngữ cảnh này.

Định hướng của luận án: Từ thực tế đó, NCS đề xuất kiến trúc phần cứng Ascon tốc độ cao (xử lý trong một chu kỳ xung nhịp) cùng giao tiếp hiệu quả, hướng tới khả năng ứng dụng thực tế trong các giao thức bảo mật như IPSec.

1.4.3. Phân tích các công trình về kiến trúc tốc độ cao cho giao thức IPSec

Trong bối cảnh mạng hiện đại, nhu cầu truyền tải dữ liệu tốc độ cao đi đôi với yêu cầu bảo mật nghiêm ngặt đã thúc đẩy xu hướng triển khai giao thức IPSec trực tiếp trên phần cứng. Tuy nhiên, các giải pháp hiện hành chủ yếu dựa vào vi xử lý mạng chuyên dụng (NPU) trên nền tảng ASIC. Mặc dù có hiệu năng tốt, ASIC lại bộc lộ nhược điểm về chi phí phát triển cao, thiếu tính linh hoạt và khó cập nhật khi thuật toán hoặc chính sách bảo mật thay đổi. Đây là rào cản lớn đối với các lĩnh vực đòi hỏi tính tùy biến cao và khả năng phản ứng nhanh như an ninh - quốc phòng và hạ tầng trọng yếu. Do đó, FPGA được coi như một nền tảng thay thế tiềm năng nhờ khả năng tái cấu hình và hiệu năng xử lý song song.

Các nghiên cứu gần đây trên nền tảng SoC (System on Chip) và FPGA chủ yếu tập trung vào hai hướng: nâng cao thông lượng và giảm thiểu độ trễ.

Về nâng cao thông lượng: Một số kiến trúc đa lõi (multi-core) đã được đề xuất để tăng tốc độ xử lý. Điển hình, công trình [58] sử dụng nhiều lõi AH/ESP và AES-HMAC-SHA-1 song song, tuy nhiên kết quả mới dừng lại ở mức tổng hợp (synthesis) trên Virtex-5 với tần số thấp (100MHz), chưa phản ánh chính xác hiệu năng thực tế. Đối với các ứng dụng IoT, giải pháp BITW (Bump in the Wire) trong [59] hỗ trợ ESP chế độ transport/tunnel nhưng lại thiếu tính năng xác thực toàn vẹn gói tin. Tương tự, kiến trúc trong [64] đạt 1,11 Gbps trên Virtex-6 nhưng kết quả đo đạc ở chế độ tunnel cũng bỏ qua bước kiểm tra toàn vẹn. Một hướng

tiếp cận khác trong [63] sử dụng cơ chế round-robin để chuyển đổi giữa các thuật toán, đạt 5,5 Gbps nhưng tiêu tốn tới 34% tài nguyên trên board Terasic DE5-Net.

Về giảm độ trễ: Đây là yếu tố then chốt cho các hệ thống thời gian thực và mạng 5G. Ullah và cộng sự [65] chỉ ra rằng độ trễ chủ yếu do quá trình xử lý trong nhân Linux và đề xuất sử dụng DPDK (Data Plane Development Kit) để xử lý tại user space, tuy nhiên đây vẫn là giải pháp phần mềm. Trong lĩnh vực ô tô, Lastinec [66] đề xuất đóng gói CAN frame vào UDP/IPSec để giảm độ trễ điều khiển. Đối với mạng 5G, Wang [67] giới thiệu kỹ thuật cắt gói (packet slicing) để xử lý song song, giúp giảm tắc nghẽn. Ngoài ra, các kỹ thuật quản lý bộ nhớ thông minh (APPAMM) [68] hay lọc gói tin bằng phần cứng [69] cũng được đề xuất để giảm tải cho CPU và tăng độ tin cậy.

Mặc dù các công trình trên đã mang lại nhiều cải tiến đáng kể về hiệu năng và độ trễ của IPSec, nhưng vẫn tồn tại một số hạn chế quan trọng cần được giải quyết:

- *Thứ nhất*, đa phần các thiết kế chưa xây dựng được một kiến trúc phân tách rõ ràng giữa đường dữ liệu tốc độ cao và đường điều khiển, dẫn đến hiện tượng nghẽn hoặc trì hoãn trong quá trình xử lý gói tin, từ đó làm gia tăng độ trễ không mong muốn.

- *Thứ hai*, để đạt được thông lượng cao, một số giải pháp phải phối hợp nhiều lõi mã hóa khác nhau, điều này làm gia tăng đáng kể mức tiêu thụ tài nguyên phần cứng, đồng thời làm giảm độ hiệu quả cài đặt.

- *Thứ ba*, nhiều nghiên cứu vẫn chưa hỗ trợ đầy đủ các chế độ an toàn nhất của IPSec, hoặc còn thiếu các thành phần quan trọng như kiểm tra toàn vẹn gói tin và ESN (Extended Sequence Number). Đặc biệt, hiện chưa có công trình nào đề xuất một kiến trúc phần cứng IPSec có khả năng vượt NAT (Network Address

Translation) ở tốc độ cao – một tính năng thiết yếu trong các hệ thống mạng thực tế. Dù NAT không còn là yêu cầu trong mạng IPv6, nhưng trên thực tế, NAT vẫn được ứng dụng rộng rãi trong các mạng IPv4 nhờ những ưu điểm của nó như khả năng cô lập mạng nội bộ, giảm thiểu bề mặt tấn công và tăng cường tính riêng tư cao hơn so với IPv6 – những yếu tố vốn rất quan trọng đối với các hệ thống đòi hỏi bảo mật nghiệp vụ cao.

Định hướng của luận án: Từ các phân tích ở trên, NCS đề xuất kiến trúc phần cứng mới mang tên **HMS-IPSec** (High-speed Multi-Crypto Secure IPSec architecture), trong đó kết hợp các kết quả nghiên cứu của luận án về các thành phần mật mã, kiến trúc phần cứng tốc độ cao cho AES, Ascon, để nâng cao hiệu năng hoạt động của IPSec, đồng thời khắc phục các hạn chế của các giải pháp đã có.

1.5. Kết luận chương 1

Chương 1 đã cung cấp một tổng quan toàn diện về các vấn đề nghiên cứu liên quan đến thuật toán mã hóa AES, thuật toán mã hóa xác thực Ascon, và giao thức bảo mật IPSec. Qua việc phân tích các thành phần mật mã cốt lõi như S-box và ma trận MDS, chương đã làm rõ vai trò của các đại lượng mật mã trong việc đảm bảo độ an toàn cũng như hiệu quả triển khai trên phần cứng. Đồng thời, chương đã đánh giá các công trình nghiên cứu gần đây về thiết kế S-box, tăng tuyến tính, và phát triển các kiến trúc phần cứng tốc độ cao cho AES, Ascon, và IPSec.

Những phân tích này chỉ ra rằng, mặc dù đã có nhiều tiến bộ trong việc tối ưu hóa hiệu năng và bảo mật của các thuật toán mã hóa và giao thức IPSec, vẫn còn tồn tại những thách thức đáng kể, bao gồm: việc cân bằng giữa độ an toàn và hiệu quả cài đặt của S-box kích thước lớn, lựa chọn ma trận MDS để giảm điểm bất động và tăng số nhánh, cũng như khắc phục các hạn chế về độ trễ và tiêu thụ

tài nguyên trong các kiến trúc phần cứng tốc độ cao. Đặc biệt, việc triển khai IPSec trên phần cứng vẫn chưa đáp ứng đầy đủ các yêu cầu về tốc độ, độ trễ thấp, và khả năng vượt NAT trong các hệ thống mạng thực tế.

Từ những cơ sở lý thuyết và thực tiễn được trình bày, luận án định hướng phát triển một kiến trúc phần cứng mới, HMS-IPSec, nhằm tích hợp hiệu quả các thành phần mật mã và tối ưu hóa hiệu năng của giao thức IPSec trên nền tảng FPGA. Các chương tiếp theo sẽ tập trung vào việc đề xuất và đánh giá các giải pháp cụ thể, bao gồm thiết kế S-box 8-bit dựa trên cấu trúc Butterfly, ma trận MDS dạng tựa dịch vòng, và phát triển kiến trúc phần cứng tốc độ cao cho AES và Ascon, hướng tới việc đáp ứng các yêu cầu khắt khe của các hệ thống mạng hiện đại.

CHƯƠNG 2

NGHIÊN CỨU LỰA CHỌN CÁC THÀNH PHẦN MẬT MÃ AN TOÀN HIỆU QUẢ CHO CÀI ĐẶT TỐI ƯU AES

Trong bối cảnh yêu cầu ngày càng cao về hiệu năng của các hệ thống mã hóa, việc cải thiện các thành phần mật mã như S-box và ma trận MDS đóng vai trò then chốt trong việc nâng cao hiệu quả của thuật toán mã khối, đặc biệt là AES. Chương 2 của luận án tập trung nghiên cứu việc thiết kế và lựa chọn các thành phần mật mã an toàn, hiệu quả cho AES, nhằm đạt được sự cân bằng giữa độ an toàn mật mã và hiệu năng triển khai trên phần cứng. Cụ thể, chương đề xuất một phương pháp sinh S-box 8-bit dựa trên cấu trúc Butterfly, tối ưu hóa biểu diễn logic của S-box, và lựa chọn ma trận MDS dạng tựa dịch vòng với các tiêu chí về số nhánh, điểm bất động. Kết quả nghiên cứu được đánh giá thông qua các tiêu chí mật mã quan trọng và hiệu quả triển khai, làm cơ sở cho việc phát triển kiến trúc AES phần cứng tốc độ cao trong chương tiếp theo. Các kết quả này được công bố trong công trình [C1], [C2].

2.1. Nghiên cứu lựa chọn S-box 8-bit an toàn hiệu quả

Trong [27], Fomin và cộng sự đã đề xuất một phương pháp xây dựng các hoán vị $2n$ bit dựa trên các hoán vị n bit, khai thác cấu trúc Butterfly do Perrin và cộng sự giới thiệu trong [28]. Phương pháp này cho phép tạo ra các hoán vị $2n$ bit có tính chất mật mã tốt, hiệu quả trên phần cứng. Tuy nhiên, nghiên cứu [27] chưa đưa ra một chiến lược cụ thể để lựa chọn các S-box có đặc tính mật mã tốt nhất cũng như phương pháp tối ưu các S-box này trên phần cứng. Trong phần này trình bày một phương pháp tìm kiếm các hoán vị $2n$ bit từ những hoán vị n bit dựa trên cấu trúc Butterfly.

Cấu trúc Butterfly xây dựng các hoán vị $2n$ bit dựa trên các hoán vị n bit được định nghĩa như sau trong [27]:

$$y_o = \begin{cases} \pi_1(x_i) \cdot y_i, & y_i \neq 0, \\ \hat{\pi}_1(x_i), & y_i = 0 \end{cases} \quad (2.1)$$

$$x_o = \begin{cases} \pi_2(y_i \cdot y_o), & y_o \neq 0, \\ \hat{\pi}_2(y_i), & y_o = 0 \end{cases} \quad (2.2)$$

Trong đó, $\pi_1, \pi_2, \hat{\pi}_1, \hat{\pi}_2$ là các hoán vị trên $\mathbb{F}_{2^m}$. Giá trị đầu vào $x = (x_i \parallel y_i)$ và đầu ra $y = (x_o \parallel y_o)$ là các giá trị trên $\mathbb{F}_{2^n}$, với $n = 2m$.

Từ cấu trúc này, NCS xây dựng Thuật toán 2.1. Thuật toán thực hiện việc tìm kiếm các S-box n bit dựa trên cấu trúc Butterfly, trong đó các hoán vị thành phần được sinh từ các hàm lũy thừa trên trường hữu hạn $\mathbb{F}_{2^m}$. Mỗi S-box được xây dựng từ tổ hợp các bộ hoán vị $(\pi_1, \pi_2, \hat{\pi})$, với mục tiêu tìm ra S-box có tính chất mật mã tốt nhất thông qua việc đánh giá các chỉ số quan trọng nhất dưới đây:

- Độ phi tuyến (Nonlinearity, NL)
- Xác suất thám mã vi sai (Maximum Differential Probability, MDP)
- Xác suất thám mã tuyến tính (Maximum Linear Probability, MLP)
- Bậc đại số (Algebraic Degree, AD)
- Bậc vào-ra (Input-Output Degree, degIO)
- Số điểm bất động (Fixed Points)

Trong khuôn khổ luận án này, NCS không xem xét và đánh giá các tiêu chí liên quan đến khả năng kháng tấn công kênh kẻ bởi các tiêu chí này phụ thuộc nhiều vào mô hình triển khai cụ thể, đòi hỏi phải có hệ thống đo đạc thực nghiệm chuyên dụng để thu thập và phân tích dấu vết rò rỉ. Hơn nữa, hầu hết các công trình liên quan đều đánh giá S-box và tầng tuyến tính dựa trên các tiêu chí cổ điển (NL, MDP, MLP, AD, degIO, số điểm bất động), nên việc sử dụng cùng hệ tiêu chí giúp các kết quả của luận án có thể so sánh trực tiếp và công bằng với các công trình liên quan.

Thuật toán 2.1. Thuật toán sinh S-box n -bit S_p dựa trên cấu trúc Butterfly

Đầu vào: n, m với $n = 2m$

Đầu ra: S-box n -bit S_p

Sinh S-box:

$best_score = 0$

$best_S = none$

For $i \leftarrow 0$ **to** $2^m - 1$:

For $j \leftarrow 0$ **to** $2^m - 1$:

if $(gcd(i, 2^m - 1) = 1 \ \&\& \ gcd(j, 2^m - 1) = 1)$

$S = GenSbox(\pi[i], \pi[j], \hat{\pi})$

$(NL, MDP, MLP, AD, degIO, FP) \leftarrow SboxProperties(S)$

$score \leftarrow weighted_eval(NL, MDP, MLP, AD, degIO, FP)$

if $score > best_score$

$best_score \leftarrow score$

$best_S \leftarrow S$

else continue

Return $best_S$

Các hàm sử dụng:

1 Function: $\pi[e]$

For $x \leftarrow 1$ **to** $2^m - 1$:

$\pi[e] \leftarrow x^e$

Return $\pi[e]$

2 Function: $\hat{\pi}$

For $x \leftarrow 1$ **to** $2^m - 1$:

$\hat{\pi} \leftarrow x^{-1}$

Return $\hat{\pi}$

3 Function: $GenSbox(\pi_1, \pi_2, \hat{\pi})$

For $x_i \leftarrow 0$ **to** $2^m - 1$:

For $y_i \leftarrow 0$ **to** $2^m - 1$:

Compute y_o :

if $y_i \neq 0$, $y_o \leftarrow \pi_1[x_i] \cdot y_i$

else $y_o \leftarrow \hat{\pi}[x_i]$

Compute x_o :

if $y_o \neq 0$, $x_o \leftarrow \pi_2[y_i] \cdot y_o$
 else $x_o \leftarrow \hat{\pi}[y_i]$

Compute S :

$k \leftarrow (x_i \ll m) \oplus y_i$
 $S[k] \leftarrow (x_o \ll m) \oplus y_o$

Return S

4 Function: *SboxProperties*(S)

Return ($NL, MDP, MLP, AD, degIO, FP$)

5 Function: *weighted_eval*($NL, MDP, MLP, AD, degIO, FP, weights$)

if $weights$ is default:

$weights = \{$
 " w_{NL} ": 5,
 " w_{MDP} ": 5,
 " w_{MLP} ": 5,
 " w_{AD} ": 2,
 " w_{degIO} ": 2,
 " w_{FP} ": 1
 $\}$

$score = ($
 $weights[w_{NL}] * NL$
 $- weights[w_{MDP}] * MDP$
 $- weights[w_{MLP}] * MLP$
 $+ weights[w_{AD}] * AD$
 $+ weights[w_{degIO}] * degIO$
 $- weights[w_{FP}] * 1$
 $)$

Return $score$

Trong quá trình sinh, thuật toán đánh giá từng S-box dựa trên các tiêu chí mật mã và lựa chọn S-box tốt nhất thông qua hàm đánh giá tổng hợp có trọng số (weighted evaluation). Cách tiếp cận này giúp đảm bảo rằng S-box đầu ra đạt được sự cân bằng giữa các tính chất mật mã quan trọng. Các trọng số trong hàm đánh

giá có thể được điều chỉnh linh hoạt, tùy theo mục tiêu ưu tiên của người thiết kế nhằm tăng cường những tính chất mật mã cụ thể.

Thuật toán đề xuất giúp giảm đáng kể không gian tìm kiếm so với phương pháp duyệt toàn bộ các hoán vị m bit. Cụ thể với trường hợp $m = 4$, có 8 hoán vị cho mỗi π_1, π_2 , do đó độ phức tạp thuật toán là $O(64)$ so với $O(16!)^2 \approx O(4.37 \times 10^{26})$ nếu duyệt toàn bộ các hoán vị. NCS xem xét 64 hoán vị với trường hợp $m = 4$, kết quả được thể hiện dưới đây:

STT	Hoán vị	NL	MDP	MLP	AD	degIO	FP
1	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2
2	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
3	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 01325467FECDA98$	104	0,0547	0,0352	7	2	2
4	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 01325467FECDA98$	108	0,0234	0,0244	7	2	2
5	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2
6	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
7	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	2	2
8	$\pi_1 = 0123456789ABCDEF$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2
9	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2
10	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
11	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 01325467FECDA98$	104	0,0547	0,0352	7	2	2
12	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 01BD9E67C583F24A$	108	0,0234	0,0244	7	2	2
13	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2
14	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
15	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	2	2

16	$\pi_1 = 01453276CD89FEBA$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2
17	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2
18	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
19	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 01325467FEC DAB98$	104	0,0547	0,0352	7	2	2
20	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 01BD9E67C583F24A$	108	0,0234	0,0244	7	2	2
21	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2
22	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
23	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	2	2
24	$\pi_1 = 01325467FEC DAB98$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2
25	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2
26	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
27	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 01325467FEC DAB98$	104	0,0547	0,0352	7	2	2
28	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 01BD9E67C583F24A$	108	0,0234	0,0244	7	2	2
29	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2
30	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
31	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	2	2
32	$\pi_1 = 01BD9E67C583F24A$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2
33	$\pi_1 = 01542376ABFE89DC$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2
34	$\pi_1 = 01542376ABFE89DC$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
35	$\pi_1 = 01542376ABFE89DC$ $\pi_2 = 01325467FEC DAB98$	104	0,0547	0,0352	7	2	2
36	$\pi_1 = 01542376ABFE89DC$ $\pi_2 = 01BD9E67C583F24A$	108	0,0234	0,0244	7	2	2
37	$\pi_1 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2

	$\pi_2 = 01542376ABFE89DC$						
38	$\pi_1 = 01542376ABFE89DC$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
39	$\pi_1 = 01542376ABFE89DC$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	2	2
40	$\pi_1 = 01542376ABFE89DC$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2
41	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2
42	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
43	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 01325467FEC DAB98$	104	0,0547	0,0352	7	2	2
44	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 01BD9E67C583F24A$	108	0,0234	0,0244	7	2	2
45	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2
46	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
47	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	3	2
48	$\pi_1 = 01E9BD7683A4C52F$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2
49	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2
50	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
51	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 01325467FEC DAB98$	104	0,0547	0,0352	7	2	2
52	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 01BD9E67C583F24A$	108	0,0234	0,0244	7	2	2
53	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2
54	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
55	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	2	2
56	$\pi_1 = 01DBE967A4F2835C$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2
57	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 0123456789ABCDEF$	104	0,0547	0,0352	7	2	2

58	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 01453276CD89FEBA$	104	0,0547	0,0352	7	2	2
59	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 01325467FECDAB98$	104	0,0547	0,0352	7	2	2
60	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 01BD9E67C583F24A$	108	0,0234	0,0244	7	2	2
61	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 01542376ABFE89DC$	104	0,0547	0,0352	7	2	2
62	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 01E9BD7683A4C52F$	108	0,0234	0,0244	7	2	2
63	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 01DBE967A4F2835C$	108	0,0234	0,0244	7	2	2
64	$\pi_1 = 019EDB76F2C5A438$ $\pi_2 = 019EDB76F2C5A438$	108	0,0234	0,0244	7	2	2

Từ 64 hoán vị trên, S-box 8-bit S_p được chọn với các tính chất mật mã tốt nhất. Cụ thể, các biến đổi thành phần của nó như sau:

$$\pi_1 = 0, 1, E, 9, B, D, 7, 6, 8, 3, A, 4, C, 5, 2, F$$

$$\pi_2 = 0, 1, D, B, E, 9, 6, 7, A, 4, F, 2, 8, 3, 5, C$$

$$\hat{\pi}_1 = 0, 1, 9, E, D, B, 7, 6, F, 2, C, 5, A, 4, 3, 8$$

$$\hat{\pi}_2 = 0, 1, 9, E, D, B, 7, 6, F, 2, C, 5, A, 4, 3, 8$$

$$S_p[256] = \{$$

00, 10, 90, E0, D0, B0, 70, 60, F0, 20, C0, 50, A0, 40, 30, 80,
01, 11, E2, 93, B4, D5, 76, 67, 88, 39, AA, 4B, CC, 5D, 2E, FF,
09, 5E, 3F, B1, 1D, C3, 82, DC, E9, F7, 46, 78, 94, 2A, AB, 65,
0E, 49, D1, 28, A2, 1B, F3, BA, 64, CD, E5, 3C, 56, 7F, 87, 9E,
0D, 2B, F5, 1E, 5A, 91, EF, C4, 37, 6C, 72, 89, BD, A6, 48, D3,
0B, 3D, 19, 84, E1, 4C, 98, A5, B2, 5F, DB, C6, 73, FE, 6A, 27,
07, 77, CE, A9, 4F, 58, 61, 16, DD, 9A, 33, F4, 22, 85, EC, BB,
06, 66, 2C, 3A, FB, 8D, 17, 71, 55, A3, 99, BF, EE, D8, C2, 44,
0F, A8, 63, 5B, 26, BE, 35, 9D, FC, D4, 8F, E7, 1A, 42, 79, C1,
02, B3, 86, C5, 9C, 6F, 4A, F9, 7B, E8, 2D, AE, D7, 14, 51, 32,
0C, FA, 57, ED, 3E, 74, B9, 43, 1F, 25, C8, 92, 81, 6B, D6, AC,
05, E4, B8, 7C, 83, A7, CB, 2F, 96, 12, 6E, DA, 45, 31, FD, 59,
0A, 8C, 9B, 47, 75, 29, DE, 52, CA, B6, F1, 6D, AF, E3, 34, 18,
04, 95, 7A, DF, C7, F2, AD, 38, 4E, 8B, 54, 21, 69, BC, 13, E6,
03, D2, A4, F6, 68, EA, 5C, 8E, 23, 41, B7, 15, 3B, C9, 9F, 7D,
08, CF, 4D, 62, D9, 36, 24, EB, A1, 7E, 1C, 53, F8, 97, B5, 8A
};

S-box S_p có hai điểm bất động. Do vậy NCS sử dụng phương affine hóa đầu vào đầu ra để đạt được các tính chất mật mã mong muốn. Theo đó biểu thức tính S-box có thể được tính theo công thức $S'(x) = B(S(A(x) \oplus a) \oplus b)$ trong đó A, B là các ma trận nhị phân 8×8 không suy biến và $a, b, x \in V_8$. Để tối ưu trong cài đặt phần cứng NCS lựa chọn các ma trận A, B sao cho chúng ít phần tử 1 nhất có thể. Kết quả thực nghiệm chỉ ra tồn tại bộ tham số A, B là ma trận đơn vị 8×8 , $a = 0$, $b = 1$. Với biến đổi như vậy, khi đó mỗi giá trị đầu ra của S-box lựa chọn sẽ được tính theo công thức: $S_p = y \oplus 1$, trong đó $y = S_p(x)$. Khi đó S-box đề xuất sẽ như sau:

$S_p[256] = \{$
 01, 11, 91, E1, D1, B1, 71, 61, F1, 21, C1, 51, A1, 41, 31, 81,
 00, 10, E3, 92, B5, D4, 77, 66, 89, 38, AB, 4A, CD, 5C, 2F, FE,
 08, 5F, 3E, B0, 1C, C2, 83, DD, E8, F6, 47, 79, 95, 2B, AA, 64,
 0F, 48, D0, 29, A3, 1A, F2, BB, 65, CC, E4, 3D, 57, 7E, 86, 9F,
 0C, 2A, F4, 1F, 5B, 90, EE, C5, 36, 6D, 73, 88, BC, A7, 49, D2,
 0A, 3C, 18, 85, E0, 4D, 99, A4, B3, 5E, DA, C7, 72, FF, 6B, 26,
 06, 76, CF, A8, 4E, 59, 60, 17, DC, 9B, 32, F5, 23, 84, ED, BA,
 07, 67, 2D, 3B, FA, 8C, 16, 70, 54, A2, 98, BE, EF, D9, C3, 45,
 0E, A9, 62, 5A, 27, BF, 34, 9C, FD, D5, 8E, E6, 1B, 43, 78, C0,
 03, B2, 87, C4, 9D, 6E, 4B, F8, 7A, E9, 2C, AF, D6, 15, 50, 33,
 0D, FB, 56, EC, 3F, 75, B8, 42, 1E, 24, C9, 93, 80, 6A, D7, AD,
 04, E5, B9, 7D, 82, A6, CA, 2E, 97, 13, 6F, DB, 44, 30, FC, 58,
 0B, 8D, 9A, 46, 74, 28, DF, 53, CB, B7, F0, 6C, AE, E2, 35, 19,
 05, 94, 7B, DE, C6, F3, AC, 39, 4F, 8A, 55, 20, 68, BD, 12, E7,
 02, D3, A5, F7, 69, EB, 5D, 8F, 22, 40, B6, 14, 3A, C8, 9E, 7C,
 09, CE, 4C, 63, D8, 37, 25, EA, A0, 7F, 1D, 52, F9, 96, B4, 8B
 $\};$

Tính chất mật mã của S-box S_p được so sánh với các S-box tiêu biểu khác trên thế giới. Hiện nay, việc đánh giá các tính chất mật mã của S-box vẫn được thực hiện qua nhiều nguồn khác nhau, dẫn đến sự thiếu đồng nhất và không nhất quán trong việc xác định độ an toàn. Để đảm bảo tính công bằng và toàn diện trong đánh giá, luận án đã phát triển một chương trình phân tích các tính chất mật mã

của S-box (Phụ lục 1). Trong đó ngoài các tính chất quan trọng nhất, một số tính chất khác như độ dư thừa tuyến tính toàn phần CLR (Complete Linear Redundancy), SAC (Strict Avalanche Criterion), BIC (Bit Independence Criterion) cũng được xem xét. Ngoài ra luận án cũng đánh giá tính chất mật mã của các S-box sử dụng công cụ PEIGEN [90].

Kết quả so sánh được thể hiện ở Bảng 2.1 và Bảng 2.2 (dữ liệu được lưu trữ trong tệp *sboxes.txt*, Phụ lục 2).

Bảng 2.1. So sánh tính chất mật mã của S_p với các S-box tiêu chuẩn

Tính chất	S_p	AES	Kuznyechik	Camellia	SEED_S0	Kalyna_Pi0
NL	108	112	100	112	112	104
MDP	0,0234	0,0156	0,0312	0,0156	0,0156	0,0312
MLP	0,0244	0,0156	0,0478	0,0156	0,0156	0,0351
AD	7	7	7	7	7	7
degIO	3	2	3	2	2	3
FP	0	0	0	0	2	0
CLR	False	True	False	True	True	False
SAC	0,5166	0,5049	0,5125	0,4983	0,5042	0,5039
BIC-SAC	0,4907	0,4954	0,5059	0,4967	0,4983	0,4985
BIC-NL	111,57	112	106,79	112	112	106,64

S-box lựa chọn S_p có độ phi tuyến lớn hơn các S-box của Kuznyechik, Kalyna, Blarus. Độ phi tuyến này nhỏ hơn một chút so với các S-box của AES, Camelia, SEED.

Tương tự S-box S_p có độ phi tuyến chỉ nhỏ hơn [22], [24]. Điều này bởi [22], [24] cũng được xây dựng dựa trên đại số và có các tính chất khá giống với AES, Camelia, SEED. Tuy nhiên các S-box dạng này lại có một số tính chất yếu như bậc vào ra thấp, sở hữu độ dư thừa tuyến tính toàn phần.

Bảng 2.2. So sánh tính chất mật mã của S_p với các công trình khác

Tính chất	S_p	[19]	[20]	[21]	[22]	[23]	[24]	[25]
NL	108	96	106	90	112	96	112	96
MDP	0,0234	0,0390	0,0234	0,0390	0,0156	0,0468	0,0156	0,0312
MLP	0,0244	0,0625	0,0295	0,0881	0,0156	0,0625	0,0156	0,0625
AD	7	7	7	7	7	7	7	7
degIO	3	3	2	3	2	3	2	3
FP	0	1	0	1	0	0	2	2
CLR	False	False	False	False	True	False	True	False
SAC	0,5166	0,4978	0,5042	0,5034	0,4980	0,5029	0,5049	0,5015
BIC-SAC	0,4907	0,4983	0,4971	0,4960	0,5018	0,4974	0,4954	0,5018
BIC-NL	111,57	103,86	109,86	103,21	112	104	112	104,21

Bảng 2.3. So sánh tính chất mật mã của S_p với các S-box sử dụng PEIGEN

S-box	NL	DU	MDP	Lin	MLP	AD
AES	112	4	0,0156	32	0,0156	7
S_p	108	6	0,0234	40	0,0244	7
Kuznyechik	100	8	0,0312	56	0,0478	7
Camellia	112	4	0,0156	32	0,0156	7
SEED_S0	112	4	0,0156	32	0,0156	7
Kalyna_Pi0	104	8	0,0312	48	0,0351	7
[19]	96	10	0,0390	64	0,0625	7
[20]	106	6	0,0234	44	0,0295	7
[21]	90	10	0,0390	76	0,0881	7
[22]	112	4	0,0156	32	0,0156	7
[23]	96	12	0,0468	64	0,0625	7
[24]	112	4	0,0156	32	0,0156	7
[25]	96	8	0,0312	64	0,0625	7

Khi một S-box chỉ cần các phương trình ràng buộc bậc hai để mô tả mối quan hệ giữa 8 bit vào và 8 bit ra ($\text{degIO} = 2$), toàn bộ thuật toán mã khối có thể được biểu diễn dưới dạng hệ đa thức đa biến bậc ≤ 2 (MQ-2). Các hệ đa thức MQ-2 này tiềm ẩn nguy cơ có thể được mô hình hóa trực tiếp thành bài toán QUBO và chạy trên máy tính lượng tử kiểu annealing, nơi trạng thái năng lượng thấp nhất tương ứng với nghiệm của hệ phương trình. Burek và cộng sự đã minh họa rõ điều này khi họ chuyển đổi thành công toàn bộ AES-128 thành dạng QUBO sử dụng 237,915 biến logic trên nền tảng D-Wave [8]. Điều này tương tự với trường hợp S-box có độ dư thừa tuyến tính toàn phần, tức là tồn tại nhiều quan hệ tuyến tính độc lập giữa các bit đầu vào, đầu ra và thậm chí giữa các trạng thái trung gian. Khi đó, số lượng phương trình ràng buộc độc lập giảm xuống đáng kể, khiến hệ phương trình MQ-2 trở nên dễ giải hơn. Hệ quả là cấu trúc đại số của S-box mất đi tính “kháng tuyến tính” vốn có, tạo điều kiện để các kỹ thuật giải hệ phương trình trên máy tính lượng tử - đặc biệt là những phương pháp dựa trên mô hình QUBO hoặc annealing - khai thác và tìm nghiệm nhanh hơn, làm suy giảm mức độ an toàn tổng thể của thuật toán. S-box S_p được thiết kế để cải thiện bậc vào ra ($\text{degIO} = 3$), loại bỏ độ dư thừa tuyến tính toàn phần để tăng khả năng chống chịu trước các tấn công lượng tử.

Do đó, S-box S_p đề xuất có ưu điểm trong việc kháng tấn công đại số lượng tử. Mặc dù độ phi tuyến giảm nhẹ ($112 \rightarrow 108$), nhưng độ an toàn vẫn đủ cao để kháng các tấn công thống kê cổ điển vi sai, tuyến tính. Bài toán QUBO cho phép mô hình hóa với bậc vào ra 2, nhưng khi nâng lên 3 sẽ phải hạ bậc về 2 dẫn đến tăng số lượng lớn các biến logic, tạo ra một rào cản kỹ thuật rõ rệt đối với việc thiết lập và giải bài toán đại số bằng các máy tính lượng tử dạng annealing ở hiện tại và trong tương lai gần.

Về mặt cài đặt, S-box S_p có thể được triển khai theo hai phương pháp chính: tra bảng hoặc biểu diễn logic. Phương pháp tra bảng có ưu điểm là dễ thực hiện và đơn giản trong thiết kế, tuy nhiên lại tiêu tốn nhiều tài nguyên phần cứng và có độ trễ cao nếu sử dụng các bộ nhớ như ROM hoặc RAM. Trong phần tiếp theo, NCS tiến hành tối ưu hóa biểu diễn logic của S-box S_p nhằm hướng đến việc triển khai hiệu quả hơn trên phần cứng, đặc biệt là trong các ứng dụng yêu cầu tốc độ cao và độ trễ thấp.

Tối ưu biểu diễn logic của S-box S_p

Có thể nói quá trình tối ưu hàm logic để giảm số phép toán logic là một thủ tục phức tạp kết hợp nhiều kỹ thuật như: “loại bỏ trùng lặp; loại bỏ phần tử chung; kỹ thuật giảm số phép NOT; đưa biểu thức vào/ra trong dấu ngoặc; tối ưu hóa phép XOR cơ bản; sử dụng dạng biểu diễn ANF (Algebraic Normal Form); hợp các phân chúng, ...”. Kế thừa từ cách biểu diễn logic của các tác giả trong công trình [78], trong công bố [C1], NCS đã tiếp tục cải tiến để giảm độ phức tạp hàm logic. Theo đó, biểu diễn logic tối ưu của S-box Kuznyechik mà NCS đã tìm ra là 206 phép toán logic, giảm 20 so với 226 phép logic mà [78] đưa ra. Áp dụng cho S-box S_p , NCS tìm được biểu diễn logic tối ưu có độ phức tạp 191 phép toán logic cơ bản, ít hơn 35 phép logic so với S-box Kuznyechik của [78]. Cụ thể như sau:

Đối với phép nhân trong trường $GF(2^4) = GF(2) / p(x)$, với $p(x) = x^4 \oplus x \oplus 1$, sử dụng biến đổi như trong [78]. Đặt đầu vào là $x = (x_3, x_2, x_1, x_0)$, $y = (y_3, y_2, y_1, y_0)$ và đầu ra tương ứng là $z = x \otimes y = (z_3, z_2, z_1, z_0)$:

$$z_3 = P_1 \cdot y_3 \oplus x_1 \cdot y_2 \oplus x_2 \cdot y_1 \oplus x_0 \cdot y_0$$

$$z_2 = P_2 \cdot y_3 \oplus P_1 \cdot y_2 \oplus x_1 \cdot y_1 \oplus x_2 \cdot y_0$$

$$z_1 = (x_2 \oplus x_1) \cdot y_3 \oplus P_2 \cdot y_2 \oplus P_1 \cdot y_1 \oplus x_1 \cdot y_0$$

$$z_0 = x_1 \cdot y_3 \oplus x_2 \cdot y_2 \oplus x_3 \cdot y_1 \oplus x_0 \cdot y_0$$

Với $P_1 = x_3 \oplus x_0$, $P_2 = x_3 \oplus x_2$. Như vậy biểu diễn logic của phép nhân trong trường $x^4 \oplus x \oplus 1$ là 31 phép logic.

Loại bỏ phân nhánh trong thuật toán. Theo [78], chúng ta có biểu diễn sau, với $\tau_{0000}(y) = \overline{y_3 + y_2 + y_1 + y_0}$:

$$y_{0j} = \tau_{0000}(y_i) \cdot \hat{\pi}_1(x_{ij}) \oplus \pi_1(x_{ij}) \cdot y_{ij}$$

$$x_{0j} = \tau_{0000}(y_0) \cdot \hat{\pi}_2(y_{ij}) \oplus \pi_2(y_{ij} \cdot y_{0j})$$

Như đã thấy cần 12 phép logic cho cả biểu diễn với $j \in \{1, 2, 3, 4\}$. Thuật toán cần sử dụng 2 lần loại bỏ phân nhánh, do đó cần 32 phép logic.

Đối với S-box 4-bit. Biểu diễn logic của π_1 với $\pi_1 = 0, 1, e, 9, b, d, 7, 6, 8, 3, a, 4, c, 5, 2, f$. Bước đầu tiên biểu diễn π_1 dưới dạng các hàm logic như sau: $\pi_1 = (f_3, f_2, f_1, f_0)$ với $f_3 = 0b0011110010101001$, $f_2 = 0b0010011100011101$, $f_1 = 0b0010101101100011$ và $f_0 = 0b0101111001000101$. Đặt đầu vào và đầu ra tương ứng là $x = (x_3, x_2, x_1, x_0)$ và $y = (y_3, y_2, y_1, y_0)$. Ta có $y = \pi_1(x)$ có thể hiểu là $y_i = f_i(x)$ với $i \in \overline{0,3}$. Bước thứ hai, biểu diễn f_i dưới dạng đại số cơ bản hay còn gọi là ANF. Trong bước này sử dụng biến đổi Fourier rời rạc để tìm biểu diễn ANF của f_i (Bảng 2.4).

Lúc này f_0 được biểu diễn dưới dạng ANF như sau:

$$f_0(x) = x_0 \oplus x_2 \oplus x_0x_2 \oplus x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_2x_3 \oplus x_0x_2x_3$$

Bước thứ ba, sử dụng các phép biến đổi được trình bày trong [78] để rút gọn hàm logic $f_0(x)$. Ta có biến đổi sau:

$$\begin{aligned} f_0(x) &= (x_0 \oplus x_2) \oplus (x_0x_1x_2 \oplus x_0x_1x_3) \\ &\quad \oplus (x_0x_2 \oplus x_2x_3 \oplus x_0x_2x_3) \\ &= (x_0 \oplus x_2) \oplus x_0x_1(x_2 \oplus x_3) \oplus x_2(x_0 \oplus x_3 \oplus x_0x_3) \\ &= (x_0 \oplus x_2) \oplus x_0x_1(x_2 \oplus x_3) \oplus x_2(x_0 \oplus x_3) \end{aligned}$$

Tương tự như vậy với các hàm logic f_1, f_2 và f_3 ta có:

$$f_1(x) = x_1 \oplus x_0(x_0 \oplus x_2) \oplus \overline{x_3}((x_0 \oplus x_1) \oplus (x_1 \oplus x_2))$$

$$f_2(x) = x_0 \oplus (x_0 + x_1) \oplus x_1(x_0 + x_3) \oplus x_3(x_2 \oplus (x_1 + x_2))$$

$$f_3(x) = x_1 \oplus (x_0 \oplus x_2) \oplus (x_0 + x_3) \oplus x_2(x_1 + x_2)$$

Bảng 2.4. Biểu diễn ANF của f_0

x_3	x_2	x_1	x_0	f_0				
0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1
0	0	1	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0
0	1	0	0	1	1	1	1	1
0	1	0	1	1	0	0	1	1
0	1	1	0	1	1	0	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	0
1	0	1	0	0	0	0	0	0
1	0	1	1	0	0	1	1	1
1	1	0	0	0	0	0	0	1
1	1	0	1	1	1	1	0	1
1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	1	0

Bước cuối cùng thực hiện nguyên lý quan trọng để rút gọn hàm logic cũng như đẩy nhanh tốc độ thực thi trên phần cứng là đặt biến chung. Ở đây dễ dàng nhận thấy với các hàm logic f_0, f_1, f_2 và f_3 xuất hiện nhiều các đa thức chung. Do vậy việc đặt biến chung tạo nên lợi thế khi triển khai cài đặt các hàm logic này trên phần cứng. Đặt:

$$P_1 = x_0 + x_1; P_2 = x_1 + x_2$$

$$P_3 = x_0 \oplus x_2; P_4 = x_0 + x_3$$

$$P_5 = x_2 \cdot P_4$$

Các hàm logic lúc này được biến đổi thành:

$$y_0 = f_0(x) = P_3 \oplus P_5 \oplus x_0 \cdot x_1 \cdot (x_2 \oplus x_3)$$

$$y_1 = f_1(x) = x_1 \oplus x_0 \cdot P_3 \oplus \overline{x_3} \cdot (P_1 \oplus P_2)$$

$$y_2 = f_2(x) = x_0 \oplus P_1 \oplus P_5 \oplus x_3 \cdot (x_2 \oplus P_2)$$

$$y_3 = f_3(x) = x_1 \oplus P_3 \oplus P_4 \oplus x_2 \cdot P_2$$

Nhận thấy lúc này để biểu diễn π_1 chúng ta chỉ cần 25 phép toán logic (giảm đi 6 phép toán so với 31 phép toán trước cần trước khi đặt biến chung).

Với π_2 , $\hat{\pi}_1$ và $\hat{\pi}_2$ thực hiện tương tự với π_1 . Kết quả như sau.

Biểu diễn logic của π_2 (cần 25 phép toán logic):

$$P_1 = x_0 + x_2; P_2 = x_2 \cdot x_3.$$

$$P_3 = x_0 \oplus x_1; P_4 = x_0 + x_3.$$

$$y_0 = x_0 \cdot x_3 \oplus x_1 \cdot P_1 \oplus \overline{P_2} \cdot P_3.$$

$$y_1 = x_0 \cdot x_1 \oplus x_0 \cdot P_2 \oplus P_1 \oplus P_4.$$

$$y_2 = (x_0 \oplus x_2) \cdot (x_1 + x_3) \oplus P_1 \oplus P_3.$$

$$y_3 = x_3 \cdot (x_1 + x_2) \oplus P_4 \oplus x_2 \oplus P_3.$$

Biểu diễn logic của $\hat{\pi}_1$ và $\hat{\pi}_2$ (cần 23 phép toán logic):

$$P_1 = x_0 \oplus x_1; P_2 = x_1 \cdot x_2.$$

$$y_0 = x_3 \oplus (x_2 + P_1) \oplus P_2 \cdot (x_0 \oplus x_3).$$

$$y_1 = x_3 \oplus x_1 \cdot (x_0 + x_3) \oplus x_2 \cdot P_1.$$

$$y_2 = x_0 \cdot (x_2 + x_3) \oplus x_0 \cdot x_1 \oplus x_2 \oplus x_3.$$

$$y_3 = (x_3 + (x_1 \oplus x_3) \oplus x_3 \cdot (x_0 \oplus P_2)).$$

Như vậy, chúng ta có bảng so sánh như sau về độ phức tạp cho S-box S_p :

Bảng 2.5. Độ phức tạp biểu diễn logic của S-box S_p

Phép logic	AND	OR	NOT	XOR	Tổng
Hai phép $\otimes$	32			30	62
Hai lần bỏ phân nhánh	16	6	2	8	32
π_1	7	3	1	14	25
π_2	8	4	1	12	25
$\hat{\pi}_1$	7	4		12	23
$\hat{\pi}_2$	7	4		12	23
Affine đầu ra				1	1
Tổng	67	31	4	89	191

Độ phức tạp của S-box S_p là 191 phép (trong đó gồm 67 phép AND, 31 phép OR, 4 phép NOT và 89 phép XOR).

Bảng 2.6 là so sánh cài đặt của S-box S_p với các công trình khác trên FPGA. (Chi tiết cài đặt module S-box S_p xem ở Phụ lục 4).

Bảng 2.6. So sánh số lượng LUT của S-box S_p trên FPGA

Work	Device	LUT	Max Freq (Mhz)	Throughput (Gbps)	Mbps/LUT
This work	Virtex-6 xc6vlx240t	35	800	6,4	183
[30]	Virtex-6 xc6vlx240t	31	724,638	5,79	187
[43]	Virtex-6 xc6vlx240t	32	696,37	5,57	170
[37]	Virtex-6 xc6vlx240t	40	732,279	5,86	146
This work	Virtex-7 xc7vx690t	32	826	6,6	206
[38]	Virtex-7 xc7vx690t	32	813	6,5	203
[35]	Virtex-7 xc7vx690t	40	593	4,74	118

S-box S_p đạt tần số hoạt động cao hơn so với các công trình liên quan trước đây. Đặc biệt trên nền tảng FPGA Virtex-7, hiệu quả sử dụng tài nguyên - tính theo Mbps/LUT - của S-box S_p là 206, cao hơn so với [35] và [38].

2.2. Nghiên cứu lựa chọn ma trận MDS an toàn hiệu quả

Trong biến đổi MixColumn của AES sử dụng ma trận dịch vòng MDS kích thước 4×4 với các phần tử của trường $\mathbb{F}_{2^8}$ và đa thức sinh bất khả quy là $f(x) = x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$. Như đã phân tích, trong phần này NCS đề xuất ma trận MDS dạng tựa dịch vòng cho tầng tuyến tính nhằm đảm bảo các tiêu chí sau:

- Là ma trận MDS với số nhánh cực đại
- Có ít nhất số điểm bất động
- Có nhiều hệ số 1 nhất có thể, đồng thời các hệ số còn lại phải cài đặt được hiệu quả

Từ ba tiêu chí trên, NCS lựa chọn ma trận có dạng như sau:

$$\begin{pmatrix} c_0 & 1 & 1 & 1 \\ 1 & 1 & c_1 & c_2 \\ 1 & c_3 & 1 & c_4 \\ 1 & c_5 & c_6 & 1 \end{pmatrix} \quad (2.3)$$

Để thấy ma trận (2.3) là ma trận dạng tựa dịch vòng. Nó có 9 hệ số 1, nhiều hơn 1 so với 8 của ma trận MDS AES, các hệ số c_i là các phần tử của trường $\mathbb{F}_{2^8}$ với đa thức sinh bất khả quy $f(x) = x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$. Tiếp theo cần đảm bảo các hệ số c_i có thể dễ dàng cài đặt. Các phép nhân với phần tử sinh của trường, hoặc nhân với nghịch đảo của nó, hoặc nhân với phép một đa thức bậc thấp của nó (ví dụ $\alpha \oplus 1, \alpha^{-1} \oplus 1$) có thể đáp ứng tiêu chí này. Do đó các hệ số c_i sẽ được lựa chọn trong tập $B = \{\alpha, \alpha^{-1}, \alpha + 1, \alpha^{-1} + 1\}$. Trên trường hữu hạn $\mathbb{F}_{2^8}$ với đa thức sinh bất khả quy $f(x) = x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$ và phần tử sinh $\alpha = 2$, tập B sẽ như sau:

$$B = \{2, 2^{-1} = 141, 2 \oplus 1 = 3, 2^{-1} \oplus 1 = 140\} = \{2, 141, 3, 140\}$$

Để tìm kiếm ma trận thỏa mãn tất cả các tiêu chí trên, NCS duyệt toàn bộ các phần tử c_i trong tập B này. Nhờ giới hạn tập các phần tử c_i nên không gian tìm kiếm giảm chỉ còn $4^7 = 16384$ ma trận. Kết quả NCS lựa chọn được ma trận M_p như sau :

$$M_p = \begin{pmatrix} \alpha & 1 & 1 & 1 \\ 1 & 1 & \alpha & \alpha \oplus 1 \\ 1 & \alpha \oplus 1 & 1 & \alpha \\ 1 & \alpha & \alpha^{-1} \oplus 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 3 \\ 1 & 3 & 1 & 2 \\ 1 & 2 & 140 & 1 \end{pmatrix}$$

Khi áp vào biến đổi MixColumns sẽ có dạng

$$\begin{pmatrix} y_{00} & y_{01} & y_{02} & y_{03} \\ y_{10} & y_{11} & y_{12} & y_{13} \\ y_{20} & y_{21} & y_{22} & y_{23} \\ y_{30} & y_{31} & y_{32} & y_{33} \end{pmatrix} = \begin{pmatrix} \alpha & 1 & 1 & 1 \\ 1 & 1 & \alpha & \alpha \oplus 1 \\ 1 & \alpha \oplus 1 & 1 & \alpha \\ 1 & \alpha & \alpha^{-1} \oplus 1 & 1 \end{pmatrix} \quad (2.4)$$

$$\times \begin{pmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{pmatrix},$$

trong đó $y_{ij}, x_{ij} \in \mathbb{F}_{2^n}, 0 \leq i, j \leq 3$. Để tính được các phần tử $y_{00}, y_{01}, y_{02}, y_{03}$, ta thực hiện:

$$\begin{cases} y_{00} = \alpha x_{00} \oplus x_{10} \oplus x_{20} \oplus x_{30} \\ y_{10} = x_{00} \oplus x_{10} \oplus \alpha x_{20} \oplus (\alpha \oplus 1)x_{30} \\ y_{20} = x_{00} \oplus (\alpha \oplus 1)x_{10} \oplus x_{20} \oplus \alpha x_{30} \\ y_{30} = x_{00} \oplus \alpha x_{10} \oplus (\alpha^{-1} \oplus 1)x_{20} \oplus x_{30} \end{cases} \Leftrightarrow \begin{cases} y_{00} = \alpha x_{00} \oplus x_{10} \oplus x_{20} \oplus x_{30} \\ y_{10} = x_{00} \oplus x_{10} \oplus \alpha(x_{20} \oplus x_{30}) \oplus x_{30} \\ y_{20} = x_{00} \oplus \alpha(x_{10} \oplus x_{30}) \oplus x_{10} \oplus x_{20} \\ y_{30} = x_{00} \oplus \alpha x_{10} \oplus (\alpha^{-1} \oplus 1)x_{20} \oplus x_{30} \end{cases}$$

Như vậy, để tính được một cột của ma trận đầu ra cần 15 phép XOR và 5 phép Xtime. Để thực hiện toàn bộ (2.4) sẽ cần $15 \times 4 = 60$ phép XOR và $5 \times 4 = 20$ phép Xtime.

Đối với ma trận của AES, phép MixColumns được thực hiện như sau:

$$\begin{pmatrix} y_{00} & y_{01} & y_{02} & y_{03} \\ y_{10} & y_{11} & y_{12} & y_{13} \\ y_{20} & y_{21} & y_{22} & y_{23} \\ y_{30} & y_{31} & y_{32} & y_{33} \end{pmatrix} = \begin{pmatrix} \alpha & \alpha \oplus 1 & 1 & 1 \\ 1 & \alpha & \alpha \oplus 1 & 1 \\ 1 & 1 & \alpha & \alpha \oplus 1 \\ \alpha \oplus 1 & 1 & 1 & \alpha \end{pmatrix} \times \begin{pmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{pmatrix} \quad (2.5)$$

Các hàng và các cột của trận trong MixColumns của AES là giống nhau. Để tính được phần tử y_{00} ta thực hiện.

$$y_{00} = \alpha x_{00} \oplus (\alpha \oplus 1)x_{10} \oplus x_{20} \oplus x_{30} = \alpha(x_{00} \oplus x_{10}) \oplus x_{10} \oplus x_{20} \oplus x_{30}$$

Biểu thức này cần 4 phép XOR và 1 phép Xtime. Để thực hiện được (2.5) sẽ cần $16 \times 4 = 64$ phép XOR và $4 \times 4 = 16$ phép Xtime.

Từ đây, NCS có Bảng 2.7 so sánh, trong đó 1 phép Xtime được tính bởi 3 phép XOR. Thấy rằng ma trận đề xuất M_p có số cổng XOR nhiều hơn so với ma trận MixColumns của AES. Ma trận M_p được cài đặt theo phương pháp biểu diễn logic (Chi tiết cài đặt M_p xem ở Phụ lục 4). Kết quả tổng hợp trên FPGA Virtex-7 xc7vx690t cho thấy M_p cần 132 LUT so với 128 LUT của ma trận AES, tuy nhiên tần số hoạt động không suy giảm, do đó hiệu năng tổng thể của tầng tuyến tính vẫn được giữ nguyên.

Về mặt an toàn, việc ma trận M_p chỉ có 1 điểm bất động là một ưu điểm quan trọng so với ma trận AES. Bởi việc tồn tại nhiều điểm bất động ở tầng tuyến tính sẽ làm tăng khả năng tồn tại các trạng thái “ít bị khuếch tán”, từ đó trong một số kịch bản có thể dẫn tới các đường vi sai/tuyến tính với số lượng S-box tích cực thấp hơn mức mong muốn.

Bảng 2.7. So sánh số lượng các ma trận MDS

Ma trận	Số phép XOR	Số nhánh	Điểm bất động	Nguồn
$\begin{pmatrix} \alpha & 1 & 1 & 1 \\ 1 & 1 & \alpha & \alpha \oplus 1 \\ 1 & \alpha \oplus 1 & 1 & \alpha \\ 1 & \alpha & \alpha^{-1} \oplus 1 & 1 \end{pmatrix}$	120	5	1	NCS
<i>AES</i>	112	5	2^{16}	[12]
$\begin{pmatrix} \alpha & 1 & 1 & 1 \\ 1 & 1 & \alpha & \alpha^2 \oplus \alpha \\ 1 & \alpha^2 \oplus \alpha & 1 & \alpha \\ 1 & \alpha & \alpha^2 \oplus \alpha & 1 \end{pmatrix}$	144	5	1	[85]

2.3. Đánh giá độ an toàn của AES-P

Độ an toàn của mã khối AES với các thành phần mật mã mới (sau đây gọi là **AES-P**) được đánh giá thông qua ba nhóm tiêu chí chính: 1) Tiêu chí khuếch tán; 2) Kiểm thử thống kê ngẫu nhiên; và 3) Độ phức tạp tính toán. Các tiêu chí này đảm bảo việc đánh giá độ an toàn của mã khối AES-P được thực hiện một cách toàn diện và có hệ thống.

2.3.1. Tiêu chí khuếch tán

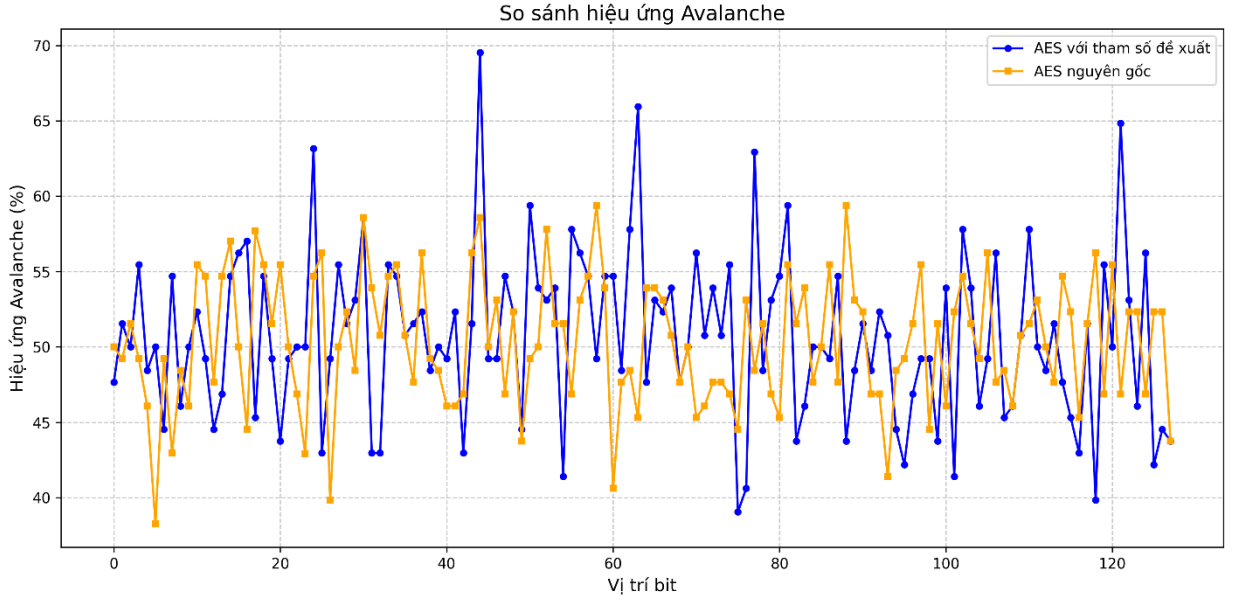
Hiệu ứng AC (Avalanche Effect) được sử dụng để đo định lượng ngẫu nhiên (phi tuyến tính) của các thuật toán mật mã, đặc biệt là mã khối. Một thuật toán mã khối tốt phải đảm bảo rằng một thay đổi nhỏ ở bản rõ hoặc khóa (lật 1 bit) sẽ khuếch tán và tạo ra những thay đổi lớn trong đầu ra. Theo tiêu chuẩn, ít nhất 50% số bit đầu ra phải thay đổi [86]. Hiệu ứng AC được tính theo công thức dưới đây:

$$Avalanche\ effect = \frac{Số\ bit\ khác\ biệt\ giữa\ hai\ bản\ mã}{Tổng\ số\ bit\ trong\ bản\ mã} \times 100\% \quad (2.6)$$

Hình 2.1 so sánh hiệu ứng AC của AES-P và AES nguyên gốc sử dụng dữ liệu bản rõ (*0xc012560c2d43f7e0b4fdc50c6062a7a9*) và khóa (*0x4cab64e83dbd1766d5f87d26d9824c09*). Kết quả thu được bằng cách lật 1 bit từ vị trí 0 đến 127. Hiệu ứng AC của luận án cũng được so sánh với một số kết quả AES sửa đổi khác:

Bảng 2.8. So sánh với các kết quả khác

Thuật toán	Hiệu ứng AC
AES-P	69,53%
[87]	60%
[88]	57,81%
AES nguyên gốc	59,38%



Hình 2.1. So sánh hiệu ứng AC của AES-P và AES nguyên gốc

Hiệu ứng AC cho thấy số lượng bit đầu ra thay đổi khi một bit ở đầu vào bị lật, nhờ đó cung cấp một đánh giá định lượng nhưng mang tính tương đối về mức độ khuếch tán trên 128 bit đầu ra. Tuy nhiên, AC chỉ cho biết tổng số bit thay đổi trung bình, chứ không phản ánh chi tiết cách từng bit đầu ra phản ứng với từng bit đầu vào. Vì vậy, để đánh giá chính xác và đầy đủ hơn tính khuếch tán và mức độ “nhảy” của từng bit đầu ra, cần sử dụng tiêu chuẩn SAC (Strict Avalanche Criterion), trong đó mỗi phần tử của ma trận SAC biểu diễn xác suất một bit đầu ra thay đổi khi một bit đầu vào cụ thể bị lật. NCS đánh giá thuật toán AES-P theo tiêu chí SAC với 10.000 cặp bản rõ ngẫu nhiên. Phương pháp đánh giá như sau:

1. Sinh ngẫu nhiên 10000 cặp bản rõ (P, P') , trong đó P' khác P đúng 1 bit tại vị trí i ($i = 0, 1, \dots, 127$).
2. Mã hoá cả P và P' bằng cùng một khoá K , thu được C và C' .
3. Tính xác suất bit đầu ra thứ j thay đổi:

$$SAC[i][j] = \frac{(\text{Số lần bit } j \text{ của } C \text{ khác bit } j \text{ của } C')}{10000} \quad (2.7)$$

4. Tính các chỉ số thống kê:

- Giá trị trung bình (Mean): Trung bình của tất cả các phần tử trong ma trận SAC (lý tưởng: 0.5).
- Độ lệch chuẩn (Std): Đo mức độ phân tán của các giá trị SAC (lý tưởng: càng nhỏ càng tốt, < 0.05).
- Min/Max: Giá trị nhỏ nhất và lớn nhất trong ma trận SAC (lý tưởng: gần 0.5).

Kết quả so sánh với AES nguyên gốc ở Bảng 2.9 cho thấy AES-P có giá trị trung bình là 0.5000, chứng tỏ mỗi bit đầu ra có xác suất thay đổi đúng 50% khi lật một bit đầu vào, phản ánh tính chất của một hàm ngẫu nhiên hoàn hảo. Độ lệch chuẩn của thuật toán rất nhỏ (0.0050), thấp hơn 10 lần so với ngưỡng yêu cầu, cho thấy các giá trị SAC phân bố rất đồng đều quanh giá trị trung bình, không có bit đầu ra nào bị lệch đáng kể so với giá trị 0.5.

Bảng 2.9. So sánh hiệu ứng SAC của AES-P và AES nguyên gốc

Chỉ số	AES-P	AES nguyên gốc
Giá trị trung bình	0,5000	0,5000
Độ lệch chuẩn	0,0050	0,0050
Giá trị nhỏ nhất	0,4766	0,4762
Giá trị lớn nhất	0,5178	0,5189

Kết quả của cả AC và SAC cho thấy phiên bản AES-P đã thay thế S-box S_p và ma trận MDS M_p đạt hiệu năng khuếch tán tương đương với AES nguyên gốc.

2.3.2. Kiểm thử thống kê ngẫu nhiên (NIST SP 800-22)

Độ an toàn của AES-P còn được đánh giá thông qua bộ kiểm thử thống kê NIST SP 800-22 [7]. Bộ kiểm thử này bao gồm 15 bài kiểm tra thống kê để đánh giá xem chuỗi bit đầu ra (bản mã) có đạt tiêu chí như một chuỗi ngẫu nhiên thực sự hay không. Phương pháp đánh giá như sau:

1. Sinh 100 triệu bit bản mã bằng cách mã hoá các bản rõ ngẫu nhiên với khoá cố định.
2. Chạy bộ kiểm thử NIST SP 800-22 trên chuỗi bit này
3. Mỗi bài kiểm tra trả về một p-value trong khoảng $[0,1]$. Nếu $p\text{-value} \geq 0,01$, bài kiểm tra được coi là PASS (chuỗi bit có hành vi ngẫu nhiên). Nếu $p\text{-value} < 0,01$, bài kiểm tra FAIL (có dấu hiệu không ngẫu nhiên).

Kết quả so sánh với AES nguyên gốc ở Bảng 2.10 cho thấy AES đề xuất có chất lượng ngẫu nhiên thống kê tương đương AES nguyên gốc.

Bảng 2.10. Kết quả kiểm thử NIST SP 800-22

Tiêu chí	AES-P	AES nguyên gốc
Tổng số test	188	188
Test PASS	186 (98,9%)	184 (97,9%)
Test FAIL	2 (1,1%)	4 (2,1%)
Test chính PASS	15/15 (100%)	15/15 (100%)

2.3.3. Độ phức tạp tính toán

Đối với thám mã vi sai và tuyến tính. Trong [12], các tác giả thiết kế đã chứng minh an toàn của AES trước hai thám mã vi sai và tuyến tính thông qua chiến lược vệt lan rộng. Trong đó, qua 4 vòng mã pháp sẽ đảm bảo tối thiểu 25 S-box tích cực vi sai/tuyến tính (Định lý 9.5.1, [12]) khi sử dụng ma trận MDS kích cỡ 4×4 trên trường $GF(2^8)$ (số nhánh ma trận là 5). S-box của AES có giá trị $MDP = MLP = 0,015625 = 2^{-6}$. Khi đó, với 4 vòng của mình, AES đạt được độ an toàn với độ phức tạp của thám mã tuyến tính và thám mã vi sai đều là $2^{6 \times 25} = 2^{150}$. Còn với S-box S_p được đề xuất, có giá trị $MDP = 0,0234 = 2^{-5,41}$

và $MLP = 0.0244 = 2^{-5,36}$, do đó độ phức tạp của thám mã vi sai và thám mã tuyến tính lần lượt là $2^{5,41 \times 25} = 2^{135,25}$ và $2^{5,36 \times 25} = 2^{134}$, không tốt bằng AES nguyên gốc, tuy nhiên các độ phức tạp này đều lớn hơn độ phức tạp vét cạn khóa 2^{128} . Luận án phân tích độ phức tạp của AES và AES-P qua các vòng:

Bảng 2.11. Độ an toàn thám vi sai

Số vòng	S-box tích cực	AES: $-\log_2(p)$	AES: Margin	AES-P: $-\log_2(p)$	AES-P: Margin
4	25	150,0	+22,0	135,2	+7,2
6	30	180,0	+52,0	162,5	+34,5
8	50	300,0	+172,0	270,7	+142,7
10	55	330,0	+202,0	297,8	+169,8
12	75	450,0	+322,0	406,1	+278,1
14	80	480,0	+352,0	433,2	+305,2

Bảng 2.12. Độ an toàn thám tuyến tính

Số vòng	S-box tích cực	AES: $-\log_2(p)$	AES: Margin	AES-P: $-\log_2(p)$	AES-P: Margin
4	25	150,0	+22,0	134	+6
6	30	180,0	+52,0	160,7	+32,7
8	50	300,0	+172,0	267,9	+139,9
10	55	330,0	+202,0	294,7	+166,7
12	75	450,0	+322,0	401,8	+273,8
14	80	480,0	+352,0	428,6	+300,6

AES-P có biên an toàn nhỏ hơn AES chuẩn do độ phi tuyến giảm từ 112 xuống 108. Tuy nhiên, chênh lệch này không ảnh hưởng đến an toàn thực tế vì:

- Ngay cả biên thấp nhất (+6 bit ở 4 vòng) vẫn đảm bảo độ phức tạp lớn hơn 2^{128} .
- Biên an toàn của AES-P ở 10 vòng (+166,7 bit) giúp nâng tổng độ phức tạp lên $2^{294,7}$, lớn hơn 2,3 lần ngưỡng an toàn 2^{128} .

Đặc biệt AES-P giải quyết được giới hạn cấu trúc không thể khắc phục của các S-box dạng nghịch đảo ($\deg IO=2$). Điều này giúp tăng cường khả năng kháng tấn công đại số/lượng tử. Chi tiết về phân tích độ phức tạp thám mã đại số được trình bày ở dưới đây.

Đối với thám mã đại số/lượng tử. Luận án tiến hành tính toán số lượng biến QUBO của AES-128 và AES-P bằng phương pháp $\deg IO$ -aware, sử dụng các phương trình bậc thấp nhất để mô tả mối quan hệ vào-ra của S-box. Kết quả như sau:

Với AES-128 chuẩn ($\deg IO = 2$), toàn bộ thuật toán được chuyển thành bài toán QUBO chỉ với 16.064 biến, bao gồm: 128 biến khóa, 1.280 biến khóa vòng, 1.280 biến đầu ra SubBytes, 1.152 biến đầu ra MixColumns, và 12.224 biến phụ cho các phép XOR (MixColumns, AddRoundKey, Key Schedule). Đặc biệt, số biến phụ dành cho S-box bằng 0, vì 39 phương trình ẩn bậc 2 của S-box AES khớp trực tiếp vào dạng QUBO bậc 2 mà không cần hạ bậc (quadraticization).

Với AES-P ($\deg IO = 3$), do S-box S_p không có bất kỳ phương trình bậc 2 nào, mọi mô tả đại số đều phải sử dụng 441 phương trình bậc 3. Mỗi đơn thức bậc 3 ($x_i \cdot x_j \cdot x_k$) phải được hạ bậc bằng cách đưa thêm biến phụ ($z_{ij} = x_i \cdot x_j$) dẫn đến số biến QUBO tăng đáng kể. Luận án phân tích hai phương pháp hạ bậc với kết quả tổng hợp như sau:

Phương pháp Q1 (theo phương trình) - gán tập biến phụ riêng cho mỗi phương trình, với 5.692 tích bậc 2 ($z_{ij} = x_i \cdot x_j$) duy nhất mỗi S-box, cho tổng 1.138.400 biến phụ S-box. Phương pháp Q2 (theo xuất hiện) gán biến phụ riêng cho mỗi lần xuất hiện đơn thức bậc 3, với 17.349 lần xuất hiện mỗi S-box, cho tổng 3.469.800 biến phụ - đây là cận trên.

Kết quả này có ý nghĩa thực tiễn quan trọng khi xét đến phần cứng quantum annealer: mỗi biến QUBO logic cần 5–15 qubit vật lý, nên AES-128 yêu cầu khoảng 160.640 qubit vật lý, trong khi AES-P (phương pháp Q1) yêu cầu khoảng 11,5 triệu qubit vật lý. Mặc dù cả hai đều vượt xa năng lực của D-Wave Advantage hiện tại (~5.000 qubit), tỷ lệ chênh lệch $71,8\times$ đảm bảo rằng khi công nghệ quantum annealer phát triển đủ mạnh để đe dọa AES, AES-P vẫn duy trì được biên an toàn đáng kể.

Bảng 2.13. Kết quả số biến QUBO theo hai phương pháp

Phương pháp	AES (degIO=2)	AES-P (degIO=3)	Tỷ lệ
Q1 (theo phương trình)	16.064	1.153.528	71,8x
Q2 (theo xuất hiện)	16.064	3.484.928	216,9x

2.4. Đánh giá hiệu năng của mã khối AES-P

Để đánh giá hiệu năng thực thi trên phần mềm của thuật toán AES-P, luận án sử dụng công cụ đo lường tiêu chuẩn của thư viện OpenSSL phiên bản 1.1.1w. Cấu hình thử nghiệm được thực hiện trên hệ thống sử dụng bộ vi xử lý Intel Core i7-6700 CPU 3.4 GHz. Hiệu năng của AES-P được so sánh trực tiếp với AES nguyên gốc cài đặt bằng ngôn ngữ C (không phải cài đặt AES-NI) sử dụng kỹ thuật tra bảng (T-box) - gộp các bước SubBytes, ShiftRows và MixColumns. Kịch bản đánh giá được thực hiện thông qua lệnh “openssl speed -evp aes-128-cbc”. Kết quả chi tiết được trình bày tại Bảng 2.14.

Bảng 2.14. Kết quả đánh giá hiệu năng

Kích thước dữ liệu	AES-128-CBC (kB/s)	AES-P-128-CBC (kB/s)
16 bytes	179.357	211.555
64 bytes	231.404	276.710
256 bytes	284.738	254.306
1024 bytes	278.991	299.803
8192 bytes	289.382	297.939
16384 bytes	291.917	291.563

Về mặt lý thuyết, khi sử dụng kỹ thuật tra bảng T-box, việc thay đổi cấu trúc S-box hay ma trận MDS hầu như không gây ra sự khác biệt đáng kể về khối lượng tính toán, dẫn đến hiệu năng thực thi của các phương án là tương đương nhau. Kết quả thực nghiệm tại Bảng 2.14 đã minh chứng điều này khi tốc độ mã hóa của AES-P duy trì được sự ổn định và đạt mức tương đương so với AES nguyên gốc, với tốc độ tối đa đạt xấp xỉ 292 MB/s (tại kích thước dữ liệu lớn).

2.5. Kết luận chương 2

Chương 2 đã trình bày một cách toàn diện quá trình nghiên cứu và lựa chọn các thành phần mật mã an toàn, hiệu quả cho thuật toán AES, bao gồm S-box 8-bit S_p và ma trận MDS M_p . S-box S_p , được sinh dựa trên cấu trúc Butterfly, đạt được các tính chất mật mã tốt với độ phi tuyến (108), bậc vào-ra (3), và không sở hữu độ dư thừa tuyến tính toàn phần, giúp tăng cường khả năng kháng lại các thám mã đại số. Biểu diễn logic tối ưu của S-box S_p giảm độ phức tạp xuống còn 191 phép toán logic, mang lại hiệu quả cao khi triển khai trên phần cứng. Ma trận MDS M_p giúp tăng độ an toàn nhờ giảm số điểm bất động mà không ảnh hưởng đến hiệu suất của AES.

Các đánh giá an toàn thông qua hiệu ứng AC, SAC cho thấy AES-P có khả năng khuếch tán mạnh, vẫn đảm bảo an toàn trước thám mã vi sai và tuyến tính. Đặc biệt AES-P có khả năng chống lại tấn công thám mã đại số trên máy tính lượng tử. Hiệu năng phần mềm của AES-P được duy trì ổn định trong thư viện OpenSSL, trong khi triển khai phần cứng sẽ được phân tích chi tiết ở chương tiếp theo. Kết quả của chương này đặt nền tảng vững chắc cho việc phát triển kiến trúc AES-P tốc độ cao, hướng tới các ứng dụng yêu cầu hiệu năng và bảo mật cao trong các hệ thống mạng hiện đại.

CHƯƠNG 3

NGHIÊN CỨU XÂY DỰNG CÁC KIẾN TRÚC PHẦN CỨNG TỐC ĐỘ CAO CHO THUẬT TOÁN MÃ HÓA AES-P VÀ ASCON

Trong bối cảnh nhu cầu về các hệ thống mã hóa tốc độ cao ngày càng tăng, đặc biệt trong các ứng dụng truyền thông thời gian thực và xử lý dữ liệu lớn, việc thiết kế các kiến trúc phần cứng hiệu quả cho các thuật toán mã hóa AES và Ascon trở thành một yêu cầu cấp thiết. Chương 3 của luận án tập trung nghiên cứu và đề xuất các kiến trúc phần cứng tốc độ cao cho hai thuật toán này trên nền tảng FPGA, hướng tới mục tiêu nâng cao tốc độ, giảm độ trễ.

Các kết quả nghiên cứu chính của chương bao gồm:

1. Đề xuất kiến trúc AES-P tốc độ cao đạt thông lượng 105,7 Gbps, độ trễ 48,4 ns và hiệu quả cài đặt 31,48 Mbps/slice trên FPGA Virtex-7. Kết quả này vượt qua phần lớn các công trình hiện có và được công bố trong [J1].

2. Đề xuất kiến trúc mới cho Ascon với độ trễ mã hóa tương đương kiến trúc đường ống nhưng không phụ thuộc vào bộ nhớ lưu trữ, phù hợp cho các hệ thống IoT thời gian thực. Kết quả này được công bố trong công trình [J2].

3.1. Nghiên cứu xây dựng kiến trúc phần cứng tốc độ cao cho AES-P

Để xây dựng một kiến trúc đường ống hiệu quả cho AES-P, NCS áp dụng phương pháp tiếp cận gồm hai bước chính:

1. **Cải tiến các thành phần:** Tinh chỉnh các khối chức năng có ảnh hưởng nhiều nhất đến đường tới hạn (critical path) và tài nguyên, bao gồm SubBytes, MixColumns và Lược đồ sinh khóa (Key Schedule).

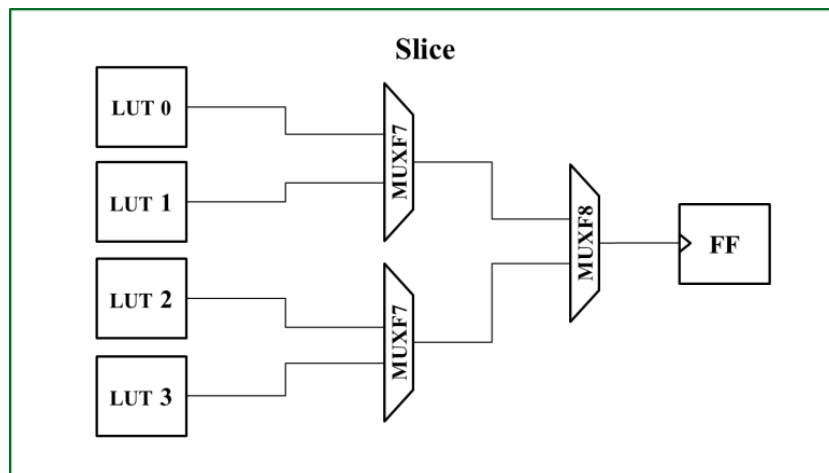
2. **Hoàn thiện kiến trúc đường ống:** Xác định vị trí và số lượng thanh ghi (registers) phù hợp để cân bằng giữa tốc độ, độ trễ và tài nguyên tiêu thụ.

3.1.1. Cải tiến các thành phần mật mã

a) Cải tiến khối SubBytes

Thành phần SubBytes sử dụng S-box S_p đã được đề xuất và phân tích ở Chương 2. S-box này được thiết kế hiệu quả cho các phần tử logic cơ bản trên FPGA, giúp giảm thiểu tài nguyên và rút ngắn đường tới hạn. S-box S_p có thể được cài đặt theo phương pháp tra bảng sử dụng các khối BRAM nội trên FPGA. Cách tiếp cận này giúp tiết kiệm tài nguyên LUT, tuy nhiên thường không đạt được hiệu suất cao nhất do đặc trưng truy cập đồng bộ của BRAM, làm phát sinh độ trễ ít nhất một chu kỳ xung nhịp [34]. Trong luận án này, S-box S_p được cài đặt theo phương pháp tối ưu biểu diễn logic (Phụ lục 4), cho phép truy cập tổ hợp với độ trễ thấp, từ đó cải thiện hiệu năng tổng thể của khối AES-P.

Cụ thể, để cài đặt 1 bit đầu ra của S-box S_p , kiến trúc chỉ sử dụng 4 LUT-6, 2 MUXF7, 1 MUXF8 và 1 Flip-Flop (FF). Toàn bộ logic này nằm gọn trong cấu trúc nội tại của một Slice, giúp giảm thiểu độ trễ kết nối (routing delay). Như vậy, một S-box 8-bit hoàn chỉnh chỉ tiêu tốn 32 LUT-6, 16 MUXF7, 8 MUXF8 và 8 FF.



Hình 3.1. Cài đặt 1 bit đầu ra của S-box S_p

Trong khi đó, giải pháp của Liu [35] sử dụng 5 6-LUT và 4 FF để cài đặt 1 bit đầu ra S-box, dẫn đến việc tiêu tốn nhiều tài nguyên hơn và phát sinh độ trễ lớn hơn giữa các phần tử. Giải pháp của Oukili [37] cài đặt S-box theo logic và cần số lượng thanh ghi lớn dẫn đến gia tăng tài nguyên và độ trễ (5 thanh ghi). S-box S_p chỉ cần 4 6-LUT cần để cài đặt 1-bit đầu ra của S-box, trong khi [35] cần 5 6-LUT. Tương tự độ trễ của S-box S_p chỉ cần 1 chu kỳ xung nhịp đồng hồ so với 5 chu kỳ của Oukili [37]. Bởi thành phần SubBytes, SubWord chiếm tỷ trọng lớn trong tổng tài nguyên của AES-P (hơn 80%), việc tối ưu S-box giúp giảm ít nhất 20% số lượng LUT và slice so với các kiến trúc của [35] và [37].

b) Thiết kế khối MixColumns

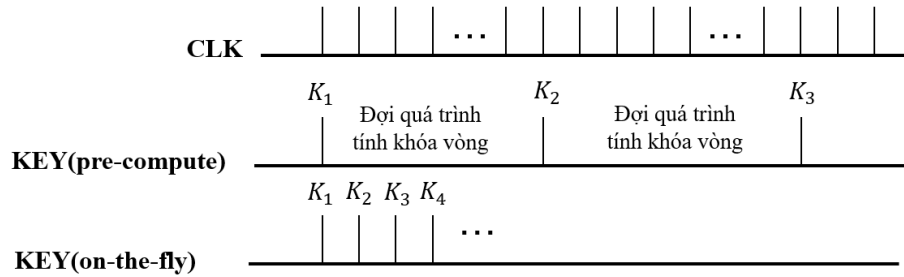
Thành phần MixColumns được xây dựng dựa trên ma trận M_p (đã trình bày ở Chương 2). Về mặt tài nguyên, ma trận này nhiều hơn 4 LUT so với ma trận gốc AES. Tuy nhiên, mức chênh lệch này là nhỏ và không đáng kể trong tổng thể tài nguyên của toàn bộ hệ thống. Quan trọng hơn, ma trận M_p vẫn đảm bảo hiệu năng và độ trễ tương đương, không gây ảnh hưởng tiêu cực đến hiệu quả hoạt động của hệ thống.

c) Cải tiến Lược đồ sinh khóa (Key Schedule)

Trong các thiết kế AES truyền thống, đặc biệt là các kiến trúc lặp (iterative), phương pháp tính khóa trước (pre-compute) thường được sử dụng bằng cách tính toán toàn bộ các khóa vòng và lưu trữ vào các khối bộ nhớ BRAM hoặc LUTRAM. Tuy nhiên, cách tiếp cận này bộc lộ hạn chế lớn về mặt hiệu năng khi tần số hoạt động bị giới hạn bởi độ trễ truy cập bộ nhớ và logic điều khiển địa chỉ khóa. Đồng thời, việc tính khóa trước tạo ra độ trễ khởi tạo ban đầu mỗi khi thay đổi khóa người dùng, gây gián đoạn luồng dữ liệu trong các ứng dụng đòi hỏi tốc độ cao.

Để khắc phục các hạn chế trên, NCS đề xuất sử dụng phương pháp sinh khóa động (on-the-fly) kết hợp với kỹ thuật đường ống (pipeline). Phương pháp này cho phép các khóa vòng được tạo đồng thời với quá trình mã hóa, nhờ đó loại bỏ nhu cầu lưu trữ toàn bộ khóa vòng trong bộ nhớ và hỗ trợ thay đổi khóa người dùng mà không làm gián đoạn luồng xử lý dữ liệu. Kết quả thực nghiệm trên FPGA Virtex-7 cho thấy kiến trúc đường ống kết hợp sinh khóa động đạt tần số hoạt động cực đại cao hơn khoảng 2,7 lần so với kiến trúc lập sử dụng phương pháp tính khóa trước. Mặc dù số lượng tài nguyên LUT sử dụng tăng khoảng 6 lần, kiến trúc đề xuất không sử dụng bất kỳ tài nguyên bộ nhớ LUTRAM hoặc BRAM nào. Kết quả này cho thấy việc kết hợp phương pháp sinh khóa động với kỹ thuật đường ống không chỉ loại bỏ hoàn toàn nhu cầu lưu trữ khóa vòng trong phần cứng mà còn cho phép hệ thống đạt hiệu năng xử lý rất cao trên nền tảng FPGA.

Hình 3.2 minh họa sự khác biệt: Trong khi phương pháp pre-compute buộc phải chờ tính xong toàn bộ khóa vòng cho K_1 mới có thể xử lý K_2 , phương pháp on-the-fly cho phép bắt đầu xử lý K_2 ngay sau 1 chu kỳ xung nhịp đồng hồ.



Hình 3.2. Lược đồ khóa pre-compute và on-the-fly

Lược đồ khóa nguyên thủy của AES có sự phụ thuộc dữ liệu lớn:

$$K'_0 = K_0 \oplus Rcon \oplus SubWord(RotWord(K_3))$$

$$K'_1 = K'_0 \oplus K_1$$

$$K'_2 = K'_1 \oplus K_2$$

$$K'_3 = K'_2 \oplus K_3$$

Sự phụ thuộc chuỗi này tạo ra đường tới hạn dài, buộc các thiết kế như [37], [38] phải dùng từ 5 đến 7 thanh ghi trung gian.

NCS đề xuất cải tiến lược đồ tính toán bằng cách định nghĩa các biến trung gian P_i :

$$P_0 = K_0 \oplus Rcon$$

$$P_1 = K_1 \oplus P_0$$

$$P_2 = K_2 \oplus P_1$$

$$P_3 = K_3 \oplus P_2$$

Khi đó, các khoá vòng được tính song song:

$$K'_0 = P_0 \oplus SubWord(RotWord(K_3))$$

$$K'_1 = P_1 \oplus SubWord(RotWord(K_3))$$

$$K'_2 = P_2 \oplus SubWord(RotWord(K_3))$$

$$K'_3 = P_3 \oplus SubWord(RotWord(K_3))$$

Cải tiến này giúp giảm số lượng thanh ghi trung gian xuống chỉ còn tối thiểu 2 (thay vì 4-7 như trước đây), qua đó có thể giảm tới 50% độ trễ so với các kiến trúc sử dụng lược đồ gốc có từ 4 tầng đường ống trở lên, trong khi vẫn đảm bảo độ an toàn tương đương.

3.1.2. Hoàn thiện kiến trúc đường ống

Kỹ thuật đường ống giúp tăng tần số hoạt động nhưng đồng thời làm tăng tài nguyên và độ trễ tổng thể (tính theo chu kỳ xung nhịp). Việc xác định điểm cân bằng hiệu quả nhất (trade-off) giữa ba yếu tố Tốc độ (T), Tài nguyên (A) và Độ trễ (L) là bài toán phức tạp.

Để giải quyết vấn đề này, NCS đề xuất tham số hiệu quả H và Mệnh đề 3.1 làm cơ sở lý thuyết cho việc thiết kế.

Mệnh đề 3.1. Gọi $H = \frac{T}{A \times L}$ là tham số đánh giá hiệu quả của kiến trúc đường ống cho một mã khối, trong đó T là tốc độ, A là tài nguyên, L là độ trễ. Gọi k là số lượng thanh ghi được chèn vào kiến trúc đường ống. Khi đó tồn tại một giá trị k^* sao cho H là cực đại. Tại k^* , kiến trúc đường ống đạt được sự đánh đổi cân bằng nhất giữa tốc độ, tài nguyên và độ trễ.

Chứng minh:

Trong kiến trúc đường ống của một mã khối, tốc độ, độ trễ phụ thuộc vào tần số theo các công thức (3.1), (3.2):

- Tốc độ:

$$T = B \times F \quad (3.1)$$

với B là kích thước khối dữ liệu được xử lý trong một chu kỳ, F là tần số hoạt động của hệ thống.

- Độ trễ:

$$L = \frac{N_c}{F} \quad (3.2)$$

với $N_c = r \times k$, và r là số vòng của mã khối, k là số thanh ghi.

- Tài nguyên: Mối quan hệ giữa tần số và tài nguyên phụ thuộc vào công nghệ và thực nghiệm cụ thể. Trong luận án này mô hình hoá mối quan hệ này theo công thức (3.3) dưới đây:

$$A = \beta \times F^\alpha \quad (3.3)$$

trong đó β là hằng số, α là tham số phản ánh mức độ đánh đổi tài nguyên để tăng tần số F . Các tham số α và β phụ thuộc vào họ FPGA, công cụ tổng hợp cụ thể. Trong luận án này, mô hình (3.3) được sử dụng như một công cụ phân tích để nhận diện vùng cân bằng trade-off, thay vì áp đặt một hằng số bất biến cho mọi nền tảng công nghệ.

Ta có tham số H tính theo công thức:

$$H = \frac{T}{A \times L} \quad (3.4)$$

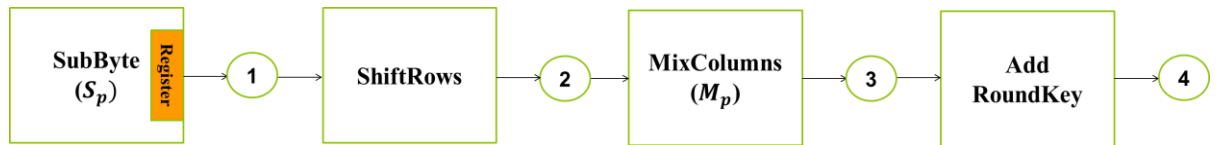
$$\begin{aligned} H &= \frac{B \times F^2}{\beta \times F^\alpha \times N_c} \\ &= \frac{B \times F^{(2-\alpha)}}{\beta \times r \times k} \end{aligned} \quad (3.5)$$

Nếu $\alpha > 2$, khi đó tài nguyên sẽ tăng rất nhanh so với tần số theo công thức (3.3), kiến trúc có thể đạt tốc độ cao nhưng phải trả giá lớn về mặt tài nguyên. Ngược lại nếu $\alpha < 2$, khi đó tài nguyên sẽ lên tăng ít đáng kể so với tần số. Đây là trường hợp phù hợp với các kiến trúc cần tài nguyên hạn chế. Đối với kiến trúc đường ống tốc độ cao, kiến trúc sẽ đạt sự đánh đổi tối ưu nhất khi tồn tại giá trị k^* sao cho $\alpha \approx 2$. Tham số H được tính như sau:

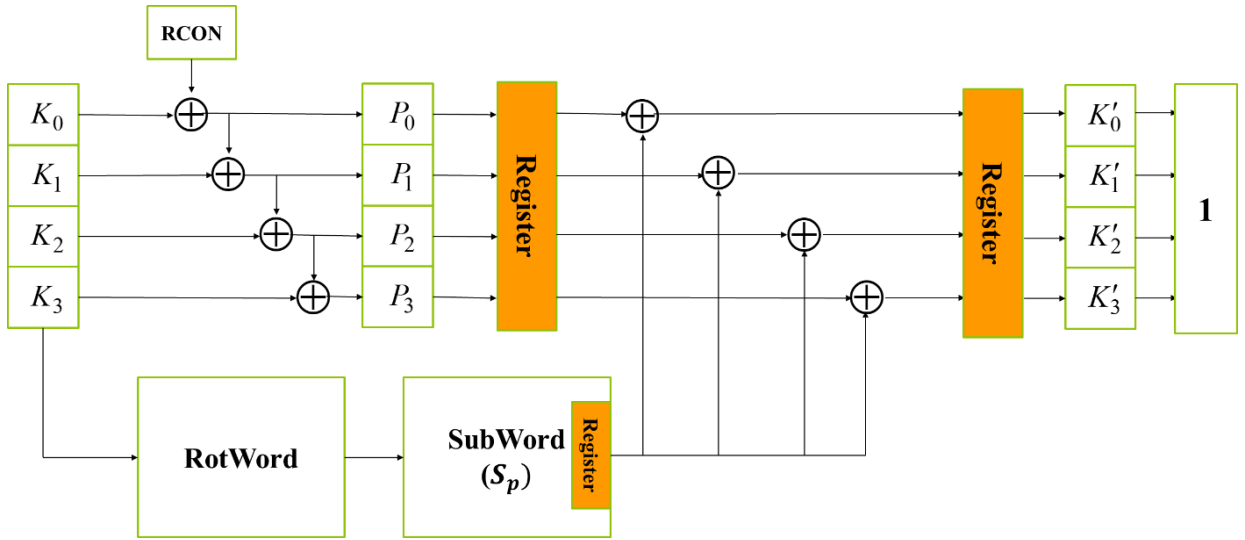
$$H = \frac{B}{\beta \times r \times k^*} \quad (3.6)$$

Giá trị của H không còn phụ thuộc vào tần số F . Bởi vậy giá trị lớn nhất của H sẽ tương ứng với kiến trúc có sự đánh đổi tốt nhất giữa tốc độ, tài nguyên và độ trễ. Mặc dù các tham số α và β phụ thuộc vào đặc thù của từng công nghệ FPGA cụ thể, nhưng khi giá trị thực nghiệm của α tiệm cận ngưỡng lý thuyết bằng 2, kiến trúc đường ống sẽ đạt đến vùng cân bằng tối ưu theo tiêu chí H . Do đó mệnh đề được chứng minh. ■

Trên cơ sở S-box và lược đồ khóa đã đề xuất, có các vị trí chèn thanh ghi dưới đây:



Hình 3.3. Các vị trí thêm thanh ghi của hàm vòng



Hình 3.4. Các vị trí thêm thanh ghi của lược đồ khóa

Dựa trên Mệnh đề 3.1 và các vị trí chèn thanh ghi (Hình 3.3, Hình 3.4), NCS xây dựng thuật toán tìm kiếm cấu hình tối ưu. Không gian tìm kiếm là $2^n - 1$ cấu hình. Giả sử mỗi cấu hình cần thời gian tổng hợp là τ . Khi đó thời gian xấu nhất để tìm ra kiến trúc tối ưu là:

$$T_{worst} = (2^n - 1) \cdot \tau = O(2^n \cdot \tau) \quad (3.7)$$

Để tăng tốc độ hội tụ, thuật toán ưu tiên xét các vị trí có độ phức tạp logic cao (SubBytes) trước.

Thuật toán 3.1. Tìm kiếm số thanh ghi tối ưu cho kiến trúc đường ống

k : Số thanh ghi thêm vào.

n : Số thanh ghi tối đa thêm vào.

$C_n^k = (\underbrace{1, \dots, 1}_k, \dots, 0)$: Tổng số vị trí có thể sắp xếp k thanh ghi

C_n^k : Vị trí j của các thanh ghi k trong tổ hợp C_n^k .

Đầu vào: S-box đề xuất, lược đồ khóa đề xuất

Đầu ra: Số thanh ghi và vị trí thanh ghi tối ưu

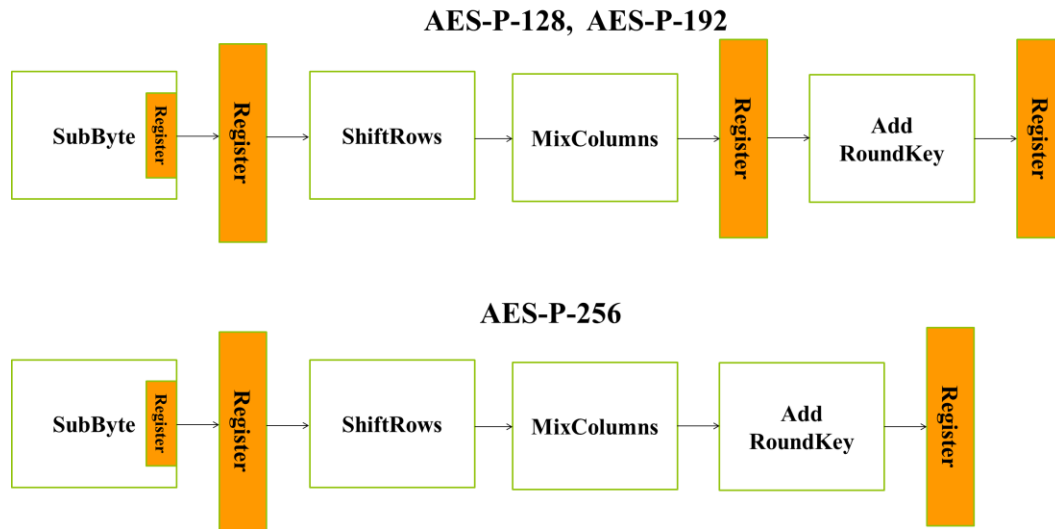
1 $k = 0, location = 0$

```

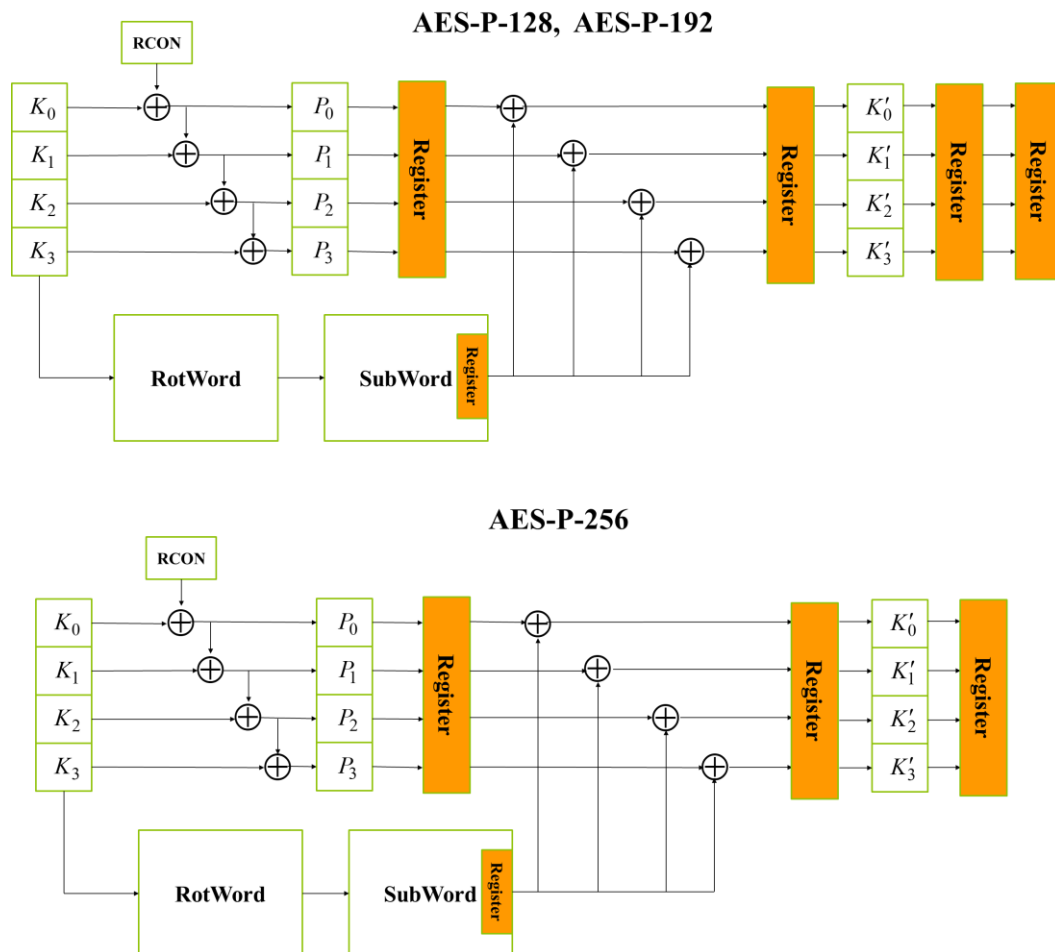
2  Tìm tần số lớn nhất  $F_0$  và tài nguyên (slices)  $A_0$ 
3  Tính tốc độ  $T_0 = F_0 \times B$ .
4  Tính độ trễ  $L_0 = \frac{N_c}{F_0}$ 
5  Tính  $H_0 = \frac{T_0}{A_0 \times L_0}$ 
6   $H_{max} = H_0$ 
5  for ( $i = 1; i < n; i++$ ) do
6      for ( $j = 0; j < C_n^i; j++$ ) do
7          Sắp xếp  $i$  thanh ghi vào vị trí  $j$  trong hàm vòng AES. Ưu tiên vị
            trí có mức logic cao trước (SubBytes). Chính sửa lược đồ khóa
            tương ứng với  $i + 1$  tầng pipeline.
8          Tìm tần số lớn nhất  $F_{i,j}$  và tài nguyên  $A_{i,j}$ .
9          Tính  $T_{i,j} = F_{i,j} \times B$ ,  $L_{i,j} = \frac{N_c}{F_{i,j}}$ ,  $H_{i,j} = \frac{T_{i,j}}{A_{i,j} \times L_{i,j}}$ .
10         if ( $H_{i,j} \geq H_{max}$ ) then
11              $H_{max} = H_{i,j}$ 
12              $L_{min} = L_{i,j}$ 
13              $k = i, location = C_n^i j$ 
12         end
13 end
14 Return  $k, location$ .

```

Áp dụng thuật toán trên, NCS xác định được cấu hình tối ưu là $k = 3$ cho AES-P-128, AES-P-192 và $k = 2$ cho độ dài khóa AES-P-256. Vị trí các thanh ghi được mô tả chi tiết trong Hình 3.5 và Hình 3.6.

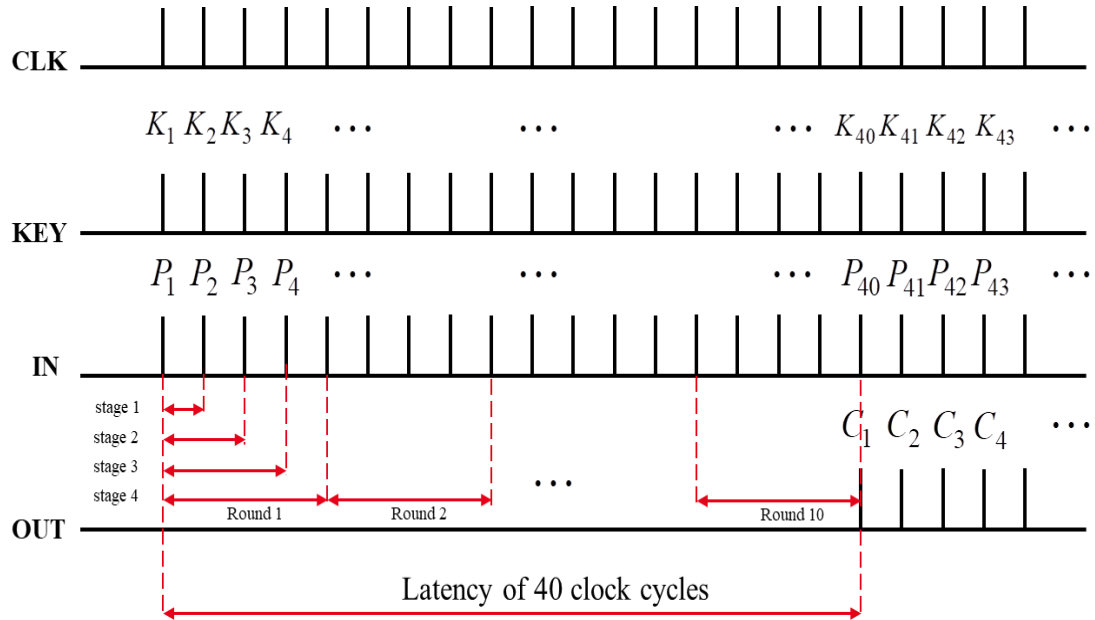


Hình 3.5. Kiến trúc tối ưu của hàm vòng AES-P



Hình 3.6. Kiến trúc tối ưu của lược đồ khóa AES-P

Với AES-P-128, kiến trúc có thể mã hóa 128-bit dữ liệu mỗi chu kỳ xung nhịp đồng hồ. Bản mã sẽ nhận được sau 40 chu kỳ xung nhịp.



Hình 3.7. Giảm đồ thời gian của AES-P-128

3.1.3. So sánh với các kết quả khác

Kiến trúc đề xuất được triển khai trên nền tảng phần cứng FPGA Virtex-7 (xc7vx690t). Bảng 3.1 trình bày các thông số hiệu năng đạt được.

Bảng 3.1. Kích thước và độ trễ của kiến trúc đề xuất

	Frequency (MHz)	Clock cycle	Latency (ns)
AES-P-128	826	40	48,4
AES-P-192	689	61	88,45
AES-P-256	740	42	56,7

Phân tích:

- **AES-P-128:** đạt tần số cao nhất ở mức 826 MHz với độ trễ tổng cộng chỉ 48,4 ns trên 40 chu kỳ xung nhịp, nhờ cấu trúc đường ống 4 tầng.

- **AES-P-256:** mặc dù AES-P-256 có cấu trúc khóa dài hơn (256 bit) tương ứng với 8 word 32 bit trong quá trình mở rộng khóa, nhưng cấu trúc này lại là bội

số của 4 word (128 bit) - kích thước khối dữ liệu trong AES-P. Nhờ vậy, đường ống giữa khối mở rộng khóa và khối mã hóa có thể hoạt động đồng pha, trong đó mỗi lần mở rộng khóa sinh ra hai khoá vòng (round key) hoàn chỉnh. Nhờ sự đồng bộ này, dù số vòng mã hóa nhiều hơn, tần số hoạt động vẫn đạt được 740 MHz và độ trễ tổng thể chỉ 56,7 ns.

- **AES-P-192:** sử dụng khóa 192 bit, tương ứng với 6 word 32 bit trong lược đồ mở rộng khóa. Số lượng 6 word này không phải bội của 4 word (128 bit) dẫn đến sự “lệch pha” giữa tiến trình mở rộng khóa và tiến trình mã hóa. Kết quả là khoá vòng của AES-P-192 phải được ghép lại từ hai nhóm khóa liên tiếp, trong đó nửa đầu lấy từ phần cuối của nhóm trước và nửa sau lấy từ phần đầu của nhóm kế tiếp. Chính sự không đồng bộ trong quá trình mở rộng khóa này kéo dài thời gian lan truyền tín hiệu và giới hạn tần số hoạt động tối đa xuống còn 689 MHz, thấp nhất trong ba cấu hình, đồng thời làm độ trễ tổng thể tăng lên 88,45 ns.

Bảng 3.2 tổng hợp các thông số kỹ thuật và hiệu suất của kiến trúc đề xuất so với các công trình liên quan.

Bảng 3.2. So sánh các kiến trúc AES tốc độ cao

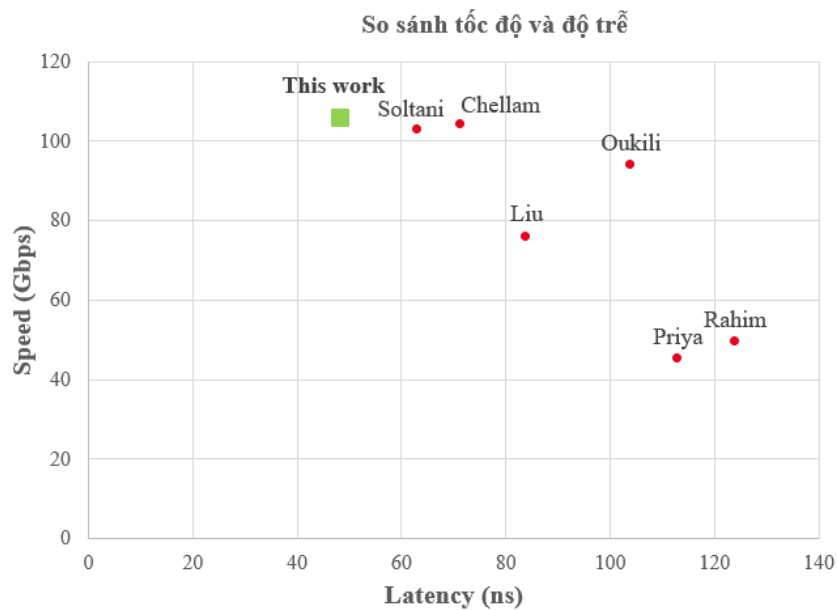
Work	Key size	Device	Pipeline Stage	Slice	BRAM	Freq (MHz)	Latency (ns)	Speed (Gbps)	Efficiency (Mbps/Slice)
This work	128	xc7vx690t	4	3.357	0	826	48,4	105,7	31,48
This work	192	xc7vx690t	4	3.845	0	689	88,45	88,192	22,93
This work	256	xc7vx690t	3	4.416	0	740	56,7	94,72	21,45
Chellam [38]	128	xc7vx690t	5	2.617	0	813	71,34	104,06	39,7
Oukili [37]	128	xv6vlx240t	8	5.759	0	732	104	93,73	16,27
Liu [35]	128	xc7vx690t	5	4.339	0	593	84	75,92	17,50
Soltani [36]	128	xc6vlx240t	5	28.520	0	803,988	63,24	102,9	3,6
Priya [41]	128	xc5vlx50t	4	7.200	0	352,628	113	45,133	6,26
Rahim [42]	128	xc6vlx240t	4	n/a	0	386,250	124	49,44	n/a

Từ bảng so sánh trên có thể thấy, thiết kế AES-P-128 trong luận án đạt được thông lượng cao nhất lên đến 105,7 Gbps, cùng với hiệu suất sử dụng tài nguyên đạt 31,48 Mbps/Slice. Đây là kết quả rất ấn tượng, đặc biệt khi xét đến việc tài

nguyên phần cứng sử dụng chỉ chiếm khoảng 3% số slice đối với AES-P-128 và 4% đối với AES-P-256 trên thiết bị xc7vx690t.

So với thiết kế của Liu [35], kiến trúc đề xuất giảm 22% số slice sử dụng trong khi vẫn đạt tốc độ và thông lượng cao hơn đáng kể. Ngoài ra, độ trễ của kiến trúc này cũng thấp hơn rõ rệt so với các thiết kế có cùng hoặc nhiều tầng pipeline hơn: giảm 32% so với Chellam [38], 53% so với Oukili [37], và 42% so với Liu [35].

Tuy nhiên, cần lưu ý rằng việc tối ưu hóa cho tốc độ cao dẫn đến mức tiêu thụ năng lượng ước tính khoảng 4,3 W, cao hơn so với một số thiết kế hướng đến tiết kiệm năng lượng (thông số 4,3W được trích xuất từ báo cáo năng lượng Power Report của công cụ Vivado sau giai đoạn Post-Implementation). Đây là sự đánh đổi chấp nhận được cho các hệ thống hiệu năng cao (High-Performance Computing). Do đó, các nghiên cứu trong tương lai nên tập trung vào việc áp dụng các kỹ thuật tối ưu hóa năng lượng nhằm giảm thiểu tiêu thụ điện mà vẫn đảm bảo hiệu năng hệ thống.



Hình 3.8. So sánh tốc độ và độ trễ với các thiết kế AES tốc độ cao khác

Các kết quả trên được đo ở chế độ ECB để đảm bảo tính công bằng khi so sánh. Tuy nhiên, trong các giao thức bảo mật mạng IP, chế độ ECB ít được sử dụng do những hạn chế về tính bảo mật mà thay vào đó là các chế độ như CTR hoặc GCM. Khi chuyển đổi sang các chế độ này, thông lượng lõi AES-P sẽ có sự suy giảm nhất định do phát sinh chi phí tài nguyên cho các khối logic điều khiển và khối logic phụ trợ (bộ đếm, bộ nhân Galois GHASH). Mức độ suy giảm hiệu năng phụ thuộc trực tiếp vào cách cài đặt các khối này, cũng như phương thức phối ghép các khối chức năng này với lõi AES-P trên nền tảng phần cứng.

3.2. Nghiên cứu xây dựng kiến trúc phần cứng tốc độ cao cho Ascon

3.2.1. Động lực nghiên cứu

Trong bối cảnh mạng IoT, nơi việc tối ưu hóa cho các thiết bị tài nguyên hạn chế là yếu tố then chốt, một số trường hợp yêu cầu ưu tiên hiệu năng cao và độ trễ thấp để đáp ứng các tiêu chí bảo mật trong môi trường truyền thông thời gian thực. Các thuật toán mã hóa nhẹ trong những trường hợp này không chỉ cần đảm bảo an toàn dữ liệu, mà còn phải duy trì tốc độ xử lý và truyền tải đủ lớn để không ảnh hưởng đến chức năng hệ thống. Một số ứng dụng tiêu biểu có thể kể đến:

- IoT Gateways: Là cầu nối giữa các thiết bị IoT và hệ thống backend, IoT Gateways thực hiện các chức năng như tổng hợp dữ liệu, chuyển đổi giao thức và bảo mật. Khả năng xử lý thông lượng cao là yếu tố quan trọng để quản lý hiệu quả các luồng dữ liệu vào và ra, đặc biệt trong các kịch bản có nhiều thiết bị kết nối. Chẳng hạn, trong một nhà máy thông minh với hàng trăm cảm biến, IoT Gateways cần mã hóa và xác thực dữ liệu nhanh chóng để duy trì hoạt động không gián đoạn.

- Xử lý và truyền video thời gian thực: Các hệ thống giám sát an ninh thông minh, xe tự hành, UAV, và thực tế ảo/tăng cường (AR/VR) yêu cầu tốc độ xử lý rất cao với độ trễ cực thấp. Ví dụ, trong các hệ thống nhận dạng khuôn mặt hoặc

phát hiện hành vi bất thường, dữ liệu video phải được mã hóa ngay lập tức để vừa bảo vệ quyền riêng tư, vừa đảm bảo rằng thông tin đầu vào được xử lý kịp thời. Đặc biệt, các thiết bị bay không người lái (UAV) được sử dụng trong các ứng dụng như nông nghiệp thông minh, giao hàng tự động, hoặc tìm kiếm cứu nạn cần khả năng mã hóa nhanh chóng các luồng dữ liệu video, cảm biến và điều khiển. Độ trễ trong các hệ thống này thường được yêu cầu dưới mức 1 mili giây, đặt ra thách thức lớn đối với các thuật toán mã hoá xác thực hạng nhẹ.

Động lực chính cho việc đề xuất kiến trúc Ascon xuất phát từ những bất cập trong các công trình Ascon trước đây (phần 1.4.2.2), đặc biệt là trong các thiết kế hướng đến thông lượng cao như [50] và [54]. Các công trình này đều áp dụng kỹ thuật mở vòng lặp 4 vòng của phép hoán vị Ascon trong một chu kỳ xung nhịp, qua đó tăng tốc độ xử lý trên mỗi vòng. Tuy nhiên, để mã hóa và xác thực một khối dữ liệu 128 bit, cả hai thiết kế vẫn yêu cầu hai chu kỳ xung nhịp bởi kiến trúc mở vòng lặp chưa được tối ưu hóa hoàn toàn. Hệ quả của việc kéo dài số chu kỳ xử lý một khối là yêu cầu bộ nhớ đệm trung gian (buffer memory) tăng lên tương ứng để đảm bảo dữ liệu không bị mất mát trong quá trình mã hóa. Mỗi quan hệ này được định lượng bằng phương trình:

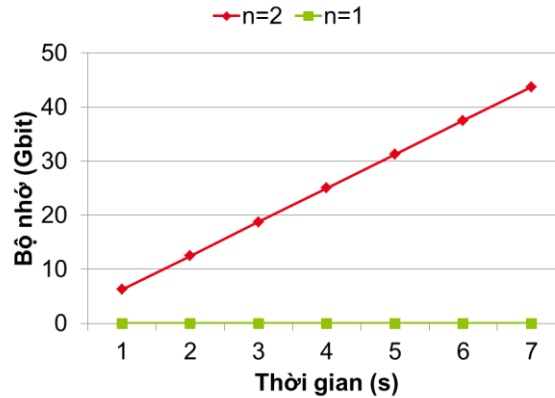
$$Memory = t \cdot \frac{(n - 1)}{n} \cdot B \cdot f \quad (3.8)$$

Trong đó:

- t : Thời gian hoạt động liên tục của hệ thống (đơn vị: giây)
- n : Số chu kỳ xung nhịp để xử lý một khối dữ liệu
- B : Kích thước khối (bit)
- f : Tần số hoạt động (MHz)

Áp dụng phương trình (3.8) với một ví dụ cụ thể: nếu hệ thống yêu cầu hoạt động liên tục trong 1 giây ở tần số 100 MHz với mỗi khối dữ liệu có kích thước

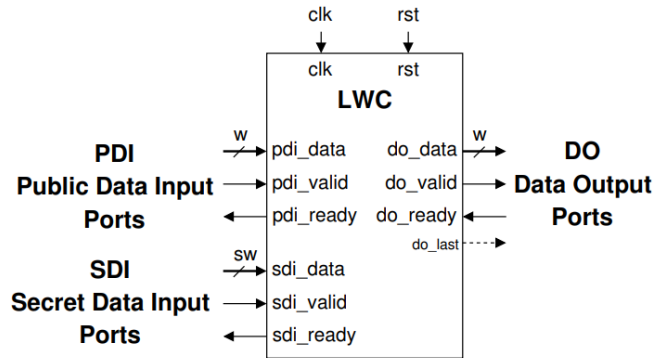
128 bit và cần 2 chu kỳ để xử lý, thì bộ nhớ cần thiết sẽ là 6,4 Gbit. Đây là một con số rất lớn đối với tài nguyên bộ nhớ on-chip của hầu hết các FPGA hiện đại, vốn chỉ cung cấp từ vài megabit đến vài chục megabit bộ nhớ BRAM. Điều này buộc các hệ thống phải sử dụng bộ nhớ ngoài (external memory), làm tăng độ trễ, mức tiêu thụ điện năng và độ phức tạp của thiết kế.



Hình 3.9. Yêu cầu bộ nhớ cần thiết để hoạt động không gián đoạn

Một bất cập khác là các công trình hiện đều sử dụng giao tiếp từ API phần cứng được cung cấp bởi các cuộc thi mật mã hạng nhẹ hoặc chưa rõ ràng về mặt giao tiếp. Điều này nhằm đảm bảo việc đánh giá một cách công bằng giữa các cài đặt của các nhà thiết kế khác nhau. Tuy nhiên API phần cứng này là một giao thức kết nối bao gồm các lệnh, header, và các loại dữ liệu khác nhau cho quá trình mã hóa, giải mã có xác thực [55]. Điều này gây khó khăn cho các nhà thiết kế khi phải tuân theo định dạng header và trình tự thao tác phức tạp của giao thức. Điều này đặc biệt khó khăn khi ứng dụng thuật toán Ascon trong các hệ thống lớn, phức tạp.

Hình 3.10 là giao tiếp LWC API bao gồm 3 đường dữ liệu chính là PDI (Public Data Inputs), SDI (Secret Data Inputs), và DO (Data Outputs) tương ứng với dữ liệu vào, dữ liệu khóa và dữ liệu ra.



Hình 3.10. Giao tiếp LWC

Với mỗi gói dữ liệu mã hóa, xác thực đều phải truyền vào các instruction, theo sau đó là các segment tương ứng với từng kiểu dữ liệu (Hình 3.11).

instruction = ACTKEY
instruction = ENC
seg_0_header
seg_0 = Npub
seg_1_header
seg_1 = AD
seg_2_header
seg_2 = Plaintext

Hình 3.11. Định dạng của PDI

Dưới đây là một ví dụ của mã hóa xác thực sử dụng LWC API:

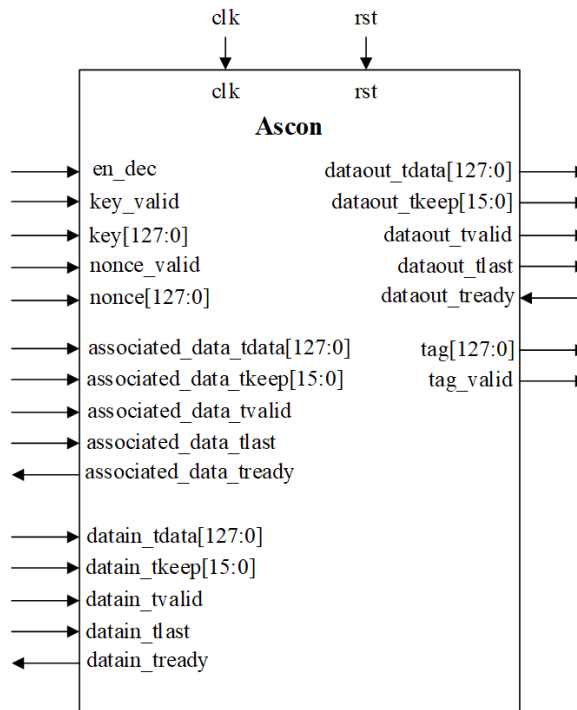
```
##### Authenticated Encryption
##### MsgID= 17, KeyID= 9 Ad Size = 31, Pt Size = 16
# Instruction: Opcode=Activate Key
INS = 70000000
# Instruction: Opcode=Authenticated Encryption
INS = 20000000
# Info :          Npub, EOI=0 EOT=1, Last=0, Length=16 bytes
HDR = D2000010
DAT = 97458EE9475E18E75B64EAA1BB7253E2
# Info :          Associated Data, EOI=0 EOT=1, Last=0, Length=31 bytes
HDR = 1200001F
DAT =
87FC680FF801FAE628DB1827F5073E84E358053423057A99C5EAF552826F1400
```

```
# Info :      Plaintext, EOI=1 EOT=1, Last=1, Length=16 bytes
HDR = 47000010
DAT = EF2848B5C71EDE4AB83E665BF7233DF9
```

Để mã hóa 16 bytes dữ liệu rõ cùng với 31 bytes dữ liệu xác thực phải cần thêm 2 instruction và 3 header tương ứng với 20 bytes dữ liệu điều khiển. Điều này đòi hỏi thêm tài nguyên logic để phân tích cú pháp và đếm, tăng độ trễ và giảm hiệu năng của hệ thống.

3.2.2. Kiến trúc phần cứng đề xuất

Kiến trúc Ascon được đề xuất trong phần này nhằm khắc phục được hai hạn chế đã chỉ ra ở trên. Hình 3.12 là giao tiếp NCS đề xuất cho kiến trúc Ascon tốc độ cao. Giao tiếp được xây dựng với độ rộng dữ liệu 128 bit, tương thích hoàn toàn với độ dài khối của thuật toán mã hóa Ascon-AEAD128.



Hình 3.12. Giao tiếp đề xuất

Giao tiếp bao gồm nhiều nhóm tín hiệu khác nhau, mỗi nhóm đảm nhận một vai trò riêng biệt trong việc điều khiển, truyền dữ liệu và xác nhận trạng thái. Cụ thể:

1. Tín hiệu đồng bộ hóa và điều khiển

- en_dec: Tín hiệu điều khiển chế độ hoạt động, nếu giá trị là '1' hệ thống thực hiện mã hóa, nếu là '0' thì thực hiện giải mã.

2. Giao tiếp khóa và nonce

- key_valid: Tín hiệu 1 chu kỳ xung nhịp báo có dữ liệu khóa.
- key[127:0]: Giá trị khóa sử dụng trong phiên mã hóa/giải mã (128 bit).
- nonce_valid: Tín hiệu 1 chu kỳ xung nhịp báo có dữ liệu Nonce.
- nonce[127:0]: Nonce dùng để đảm bảo tính không trùng lặp của mỗi phiên mã hóa, giúp đảm bảo tính bảo mật theo chuẩn AEAD.

Các tín hiệu khóa và nonce chỉ được nạp một lần duy nhất trước khi bắt đầu quá trình mã hóa hoặc giải mã, phù hợp với đặc tả của thuật toán Ascon.

3. Giao tiếp các dữ liệu

Giao tiếp dữ liệu bản rõ, dữ liệu liên kết, dữ liệu mã hóa tuân theo định dạng AXI Stream với các cờ tvalid, tready, tkeep, tlast giúp truyền tải linh hoạt các gói dữ liệu độ dài bất kỳ mà vẫn đảm bảo tính đồng bộ.

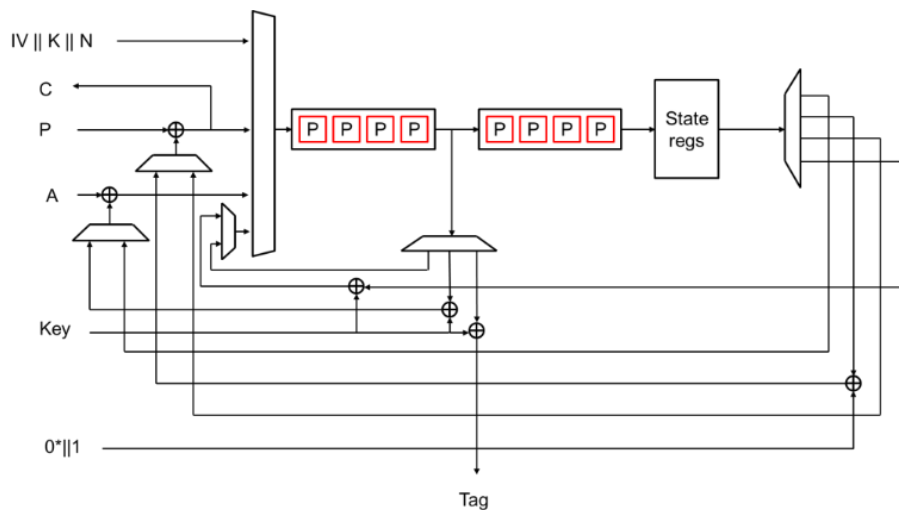
- datain/associated_data/dataout_tdata: 128 bit dữ liệu mỗi chu kỳ xung nhịp.
- datain/associated_data/dataout_tkeep: Báo số bytes hợp lệ trong 128 bit dữ liệu cuối cùng.
- datain/associated_data/dataout_tvalid: Báo dữ liệu hợp lệ.
- datain/associated_data/dataout_tlast: Tín hiệu 1 chu kỳ xung nhịp báo dữ liệu cuối cùng.

- `datain/associated_data/dataout_tready`: Tín hiệu xác nhận từ khối Ascon rằng nó sẵn sàng nhận dữ liệu tiếp theo.

4. Giao tiếp Tag xác thực

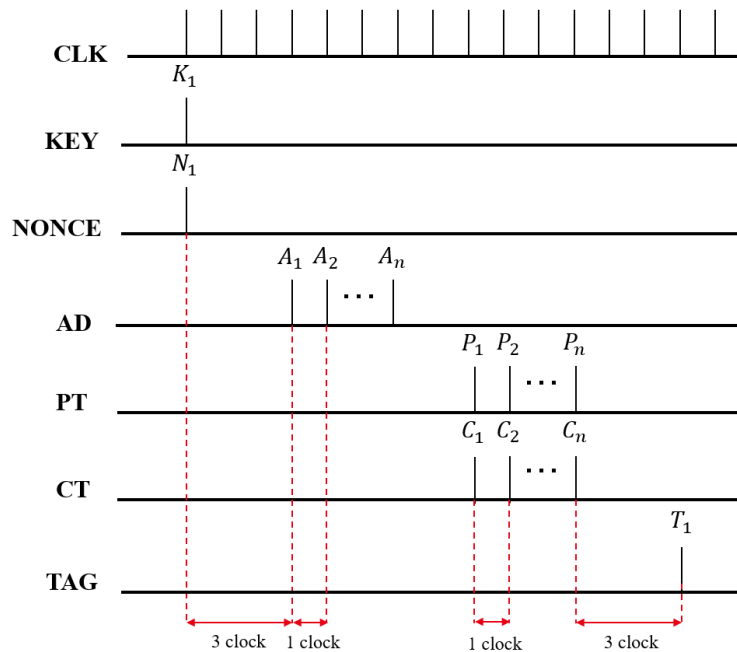
- `tag`: dữ liệu xác thực đầu ra của thuật toán Ascon, dùng để kiểm tra tính toàn vẹn và xác thực dữ liệu.
- `tag_valid`: Tín hiệu 1 chu kỳ xung nhịp cho biết tag đã sẵn sàng và hợp lệ.

Như đã đề cập, kỹ thuật đường ống, vốn được sử dụng phổ biến để tăng thông lượng trong các thiết kế phần cứng, không thể áp dụng trực tiếp cho thuật toán Ascon do đặc tính phản hồi của thuật toán. Tính chất này xuất phát từ việc các vòng lặp (round) trong Ascon yêu cầu trạng thái đầu ra của vòng trước làm đầu vào cho vòng tiếp theo, gây khó khăn trong việc phân chia các giai đoạn xử lý thành các bước độc lập như trong kỹ thuật đường ống truyền thống. Để khắc phục hạn chế này và đáp ứng yêu cầu về hiệu năng cao trong các ứng dụng nhạy cảm với độ trễ, luận án đề xuất một kiến trúc phần cứng tối ưu cho Ascon-AEAD128, tận dụng kết hợp hai kỹ thuật lặp lại và mở vòng lặp, như được minh họa chi tiết trên Hình 3.13.



Hình 3.13. Kiến trúc Ascon-AEAD128 đề xuất

Cụ thể, kiến trúc được thiết kế với sự phân chia rõ ràng giữa các giai đoạn xử lý của thuật toán Ascon-AEAD128, bao gồm giai đoạn Khởi tạo (Initialization), Dữ liệu liên kết (Associated Data), Xử lý bản rõ/bản mã (Plaintext/Ciphertext Processing), và Kết thúc (Finalization). Đối với các giai đoạn Khởi tạo và Kết thúc, biến đổi Ascon được áp dụng kỹ thuật mở vòng lặp $\times 4$, tức là mở rộng để thực hiện bốn vòng lặp của hàm biến đổi trong một chu kỳ xung nhịp. Để hoàn thành đầy đủ 12 vòng lặp yêu cầu trong các giai đoạn này, kỹ thuật iterative loop được sử dụng, cho phép lặp lại ba lần quá trình mở vòng lặp $\times 4$. Phương pháp này giúp giảm đáng kể yêu cầu về tài nguyên phần cứng, chẳng hạn như số lượng cổng logic và diện tích trên FPGA, vì không cần triển khai toàn bộ 12 vòng lặp dưới dạng phần cứng cố định. Quan trọng hơn, do các giai đoạn Khởi tạo và Kết thúc chỉ được thực hiện một lần duy nhất trong toàn bộ quá trình mã hóa hoặc giải mã, việc sử dụng iterative loop không ảnh hưởng đến tốc độ tổng thể của kiến trúc, đảm bảo hiệu năng vẫn được duy trì ở mức cao.



Hình 3.14. Giảm độ thời gian của Ascon-AEAD128 để xuất

Trong khi đó, để tối ưu hóa thông lượng cho các giai đoạn Dữ liệu liên kết và Xử lý bản rõ/bản mã, vốn là các giai đoạn chiếm phần lớn thời gian xử lý trong các ứng dụng thực tế, biến đổi Ascon được mở vòng lặp $\times 8$, tức là mở rộng để thực hiện tám vòng lặp trong một chu kỳ xung nhịp. Việc áp dụng mở vòng lặp $\times 8$ cho phép kiến trúc xử lý toàn bộ một khối dữ liệu 128 bit (bao gồm mã hóa hoặc giải mã) chỉ trong một chu kỳ xung nhịp, đạt được tốc độ tối đa. So với các công trình trước đây, chẳng hạn như [50] và [54], vốn chỉ sử dụng mở vòng lặp $\times 4$ và yêu cầu hai chu kỳ xung nhịp để xử lý một khối dữ liệu 128 bit, kiến trúc đề xuất cải thiện đáng kể hiệu năng bằng cách giảm số chu kỳ xung nhịp xuống còn một, đồng thời vẫn duy trì mức tiêu thụ tài nguyên hợp lý. Điều này không chỉ nâng cao thông lượng mà còn giảm yêu cầu bộ nhớ đệm theo phương trình (3.8), giúp đáp ứng các hệ thống đòi hỏi sự cân bằng giữa tốc độ và hiệu quả sử dụng tài nguyên.

3.2.3. So sánh với các kết quả khác

Kiến trúc Ascon đề xuất được cài đặt và tổng hợp trên thiết bị Virtex-7. Mã nguồn của kiến trúc cũng được công bố trên [56]. Kết quả được so sánh với các kiến trúc khác trong Bảng 3.3.

Bảng 3.3. So sánh các kiến trúc Ascon tốc độ cao

Implementation	Device	Clock cycle	LUT	Maximum Frequency (MHz)	TP (Mbps)	TP/Area (Mbps/LUT)
This work	Virtex-7	1	6.536	104	13.312	2,036
S. Khan [50]	Virtex-7	2	3.350	226,4	14.512	4,33
[54]	Virtex-7	2	3.678	132,1	8.454	2,30
J. Kaur [51]	Kintex 7	2	n/a	104	6.709	n/a
Rezvani [46]	Artix-7	8	1.898	263	1.683	0,887

So sánh với công trình của Khan [50], mặc dù thông lượng tối đa của kiến trúc đề xuất thấp hơn khoảng 8% (13,3 Gbps so với 14,5 Gbps), nhưng hiệu quả thực tế trong hệ thống lại cao hơn nhờ hai yếu tố:

1. Loại bỏ nhiễu cho giao thức: Kiến trúc đề xuất loại bỏ hoàn toàn các header điều khiển của LWC API, giúp thông lượng thực tế tiệm cận với thông lượng lý thuyết.

2. Nâng cao hiệu quả tích hợp hệ thống: Với đặc tính xử lý 1 chu kỳ/khối, kiến trúc không yêu cầu bộ nhớ đệm đầu vào/đầu ra, giúp giảm thiểu độ trễ hệ thống và đơn giản hóa việc tích hợp vào các đường ống xử lý dữ liệu phức tạp như IPSec hay IoT Gateway.

3.3. Đánh giá độ an toàn các kiến trúc đề xuất

Việc đánh giá an toàn cho các kiến trúc phần cứng đề xuất được xem xét trên hai khía cạnh chính: an toàn thuật toán và an toàn cài đặt.

Về mặt thuật toán, độ an toàn của AES đã được phân tích chi tiết tại Mục 2.3 (Chương 2). Đối với Ascon, kiến trúc đề xuất tuân thủ chặt chẽ các đặc tả chuẩn của thuật toán, đảm bảo các tính chất an toàn mật mã của thuật toán gốc được bảo toàn nguyên vẹn trong quá trình thực thi trên phần cứng.

Về mặt an toàn cài đặt, trọng tâm đánh giá là khả năng kháng lại các tấn công kênh kề (Side-Channel Attacks - SCA), bao gồm phân tích năng lượng DPA (Differential Power Analysis), CPA (Correlation Power Analysis), EMA (Electromagnetic Analysis) và định thời (timing). Các kiến trúc AES và Ascon đề xuất an toàn trước các tấn công định thời do không sử dụng các nhánh điều kiện và mỗi bản rõ đều được xử lý trong số chu kỳ xung nhịp cố định. Bên cạnh đó, các phương pháp đường ống, mở vòng lặp sử dụng trong các kiến trúc đề xuất cũng được khuyến nghị trong các nghiên cứu trước đây nhằm tăng cường khả năng chống lại tấn công DPA [89].

Tuy nhiên, các kiến trúc hiện tại chưa tích hợp các cơ chế bảo vệ đặc thù nhằm chống lại các tấn công kênh kề nâng cao như CPA hoặc EMA. Theo Groß và cộng sự, Ascon có thể được bảo vệ chống DPA bằng Threshold Implementation

(TI) với gia tăng diện tích khoảng $3,1\times$ so với bản không bảo vệ, cho thấy đây là hướng tăng cường thực tế và phù hợp để mở rộng trong tương lai [44].

Các giải pháp tăng cường khả năng chống tấn công kênh kề sẽ không được đề cập chi tiết trong khuôn khổ của luận án này mà sẽ được xem xét và phát triển trong các nghiên cứu tiếp theo.

3.4. Kết luận chương 3

Chương 3 đã xây dựng các kiến trúc phần cứng tối ưu cho thuật toán mã hóa AES và Ascon, nhằm giải quyết những yêu cầu về tốc độ cao, độ trễ thấp và hiệu quả tài nguyên trong các hệ thống hiện đại. Kết quả cho thấy kiến trúc AES đề xuất đạt được thông lượng 105,7 Gbps, với độ hiệu quả là 31,48 Mbps/Slice trên Virtex-7. Độ trễ của thiết kế giảm 32% so với [38], 53% so với [37], 42% so với [35]. Kiến trúc Ascon đạt tốc độ tối đa 13,312 Gbps, loại bỏ nhu cầu bộ nhớ đệm lớn và giảm độ phức tạp so với LWC API. Những kết quả này không chỉ có giá trị về mặt lý thuyết mà còn mở ra tiềm năng ứng dụng thực tiễn trong các hệ thống mã hóa hiệu năng cao. Chương tiếp theo sẽ trình bày việc áp dụng các kiến trúc này vào giao thức IPSec, một giao thức quan trọng trong việc đảm bảo an toàn thông tin trên các mạng IP.

CHƯƠNG 4

NGHIÊN CỨU ĐỀ XUẤT KIẾN TRÚC HMS-IPSEC

Chương 4 của luận án tập trung đề xuất kiến trúc phần cứng HMS-IPSec, sử dụng các kết quả nghiên cứu từ 2 chương trước (thành phần mật mã, kiến trúc phần cứng) để nâng cao thông lượng, giảm độ trễ và cải thiện tài nguyên trên FPGA. Kiến trúc được thiết kế với các thuật toán xử lý gói tin hiệu quả và được đánh giá thực nghiệm trên thực tế. Kết quả được công bố trong công trình [J2].

4.1. Động lực thiết kế kiến trúc HMS-IPSec

Dựa trên các phân tích về hạn chế của các kiến trúc IPSec trên FPGA trong phần 1.4.3, bao gồm:

- không phân tách rõ ràng giữa đường dữ liệu và đường điều khiển,
- tiêu thụ tài nguyên lớn khi sử dụng nhiều lõi mã hóa,
- thiếu hỗ trợ các tính năng triển khai thực tế như vượt NAT và ESN,
- chưa đánh giá hiệu quả của thuật toán mã hoá xác thực hạng nhẹ.

Trên cơ sở đó, luận án xác định các mục tiêu thiết kế cho kiến trúc HMS-IPSec như sau:

(1) Giảm độ trễ bằng phân tách rõ ràng đường dữ liệu và đường điều khiển: Trong các kiến trúc trước, đường dữ liệu xử lý gói tin và đường điều khiển cập nhật cấu hình thường chia sẻ cùng tài nguyên bộ nhớ và bus truy cập. Khi số giao diện mạng tăng, số điểm truy cập đồng thời vào các bảng cấu hình (SPD, SAD, bảng định tuyến, ARP) cũng tăng, dẫn đến độ trễ xử lý gói tin tăng theo số lượng giao diện.

Khi đó cần thiết kế kiến trúc HMS-IPSec mà trong đó:

- đường dữ liệu gói tin được ánh xạ vào một bus dữ liệu duy nhất với độ rộng cố định, được tối ưu cho truyền dữ liệu liên tục;

- đường điều khiển được tách ra thành một kênh cấu hình riêng, chỉ phục vụ cập nhật bảng và tham số;

- loại bỏ nút thất tra cứu các bảng, tham số bằng CAM/TCAM.

Nhờ đó, các thao tác tra cứu trên đường dữ liệu cho mỗi gói tin không phụ thuộc vào số giao diện hay tần suất cập nhật cấu hình. Đây là cơ sở để kiến trúc đạt được độ trễ ổn định, phù hợp với các ứng dụng thời gian thực.

(2) Nâng cao hiệu quả cài đặt bằng kiến trúc đường ống AES thay vì nhiều lõi: Các thiết kế IPSec tốc độ cao như [58], [64] thường sử dụng nhiều lõi AES song song để tăng thông lượng. Thông lượng tổng khi đó có thể tỉ lệ với số lõi, nhưng tài nguyên FPGA và độ phức tạp điều phối cũng tăng tương ứng. Hiệu quả cài đặt vì vậy bị giới hạn.

HMS-IPSec thay đổi hướng tiếp cận: thay vì tăng số lõi, tập trung tăng tốc một lõi duy nhất bằng:

- kiến trúc đường ống AES-P đã được cải tiến trong luận án;
- sử dụng các chế độ hoạt động GCM/CTR nhằm song song hoá và tận dụng tối đa hiệu năng của đường ống.

Điều này cho phép:

- giảm số lượng LUT/Slice dành cho khối mật mã,
- đơn giản hóa logic điều phối gói tin giữa các khối,
- tăng hiệu quả cài đặt của toàn bộ kiến trúc IPSec.

(3) Đảm bảo khả năng triển khai thực tế bằng ESN và vượt NAT: các công trình IPSec trên FPGA hiện tại chưa hỗ trợ đầy đủ các tính năng thiết yếu như vượt NAT và ESN. Việc thiếu cơ chế vượt NAT khiến các giải pháp này khó áp dụng trong các môi trường mạng thực tế, nơi thiết bị thường nằm sau các lớp NAT. Bên cạnh đó, việc không triển khai ESN 64-bit làm cho hệ thống dễ gặp vấn

đề tràn sequence number trong bối cảnh mạng tốc độ cao, ảnh hưởng đến cơ chế chống phát lại và độ tin cậy của kênh bảo mật.

(4) Đánh giá khả năng ứng dụng của mã hóa xác thực hạng nhẹ trong IPSec: Xu hướng chuẩn hóa mật mã hạng nhẹ cho các thiết bị tài nguyên hạn chế đặt ra câu hỏi quan FPGA trọng: các thuật toán này có thể thay thế hoặc bổ sung cho AES trong bối cảnh IPSec trên FPGA hay không. Các công trình hiện có tập trung chủ yếu vào AES và các biến thể của AES, trong khi chưa đánh giá một cách có hệ thống thuật toán Ascon trong giao thức IPSec. Điều này tạo ra khoảng trống nghiên cứu: cần một khung IPSec đủ tổng quát để hỗ trợ cả thuật toán chuẩn lẫn thuật toán hạng nhẹ, và đủ chặt chẽ để so sánh chúng trên cùng điều kiện.

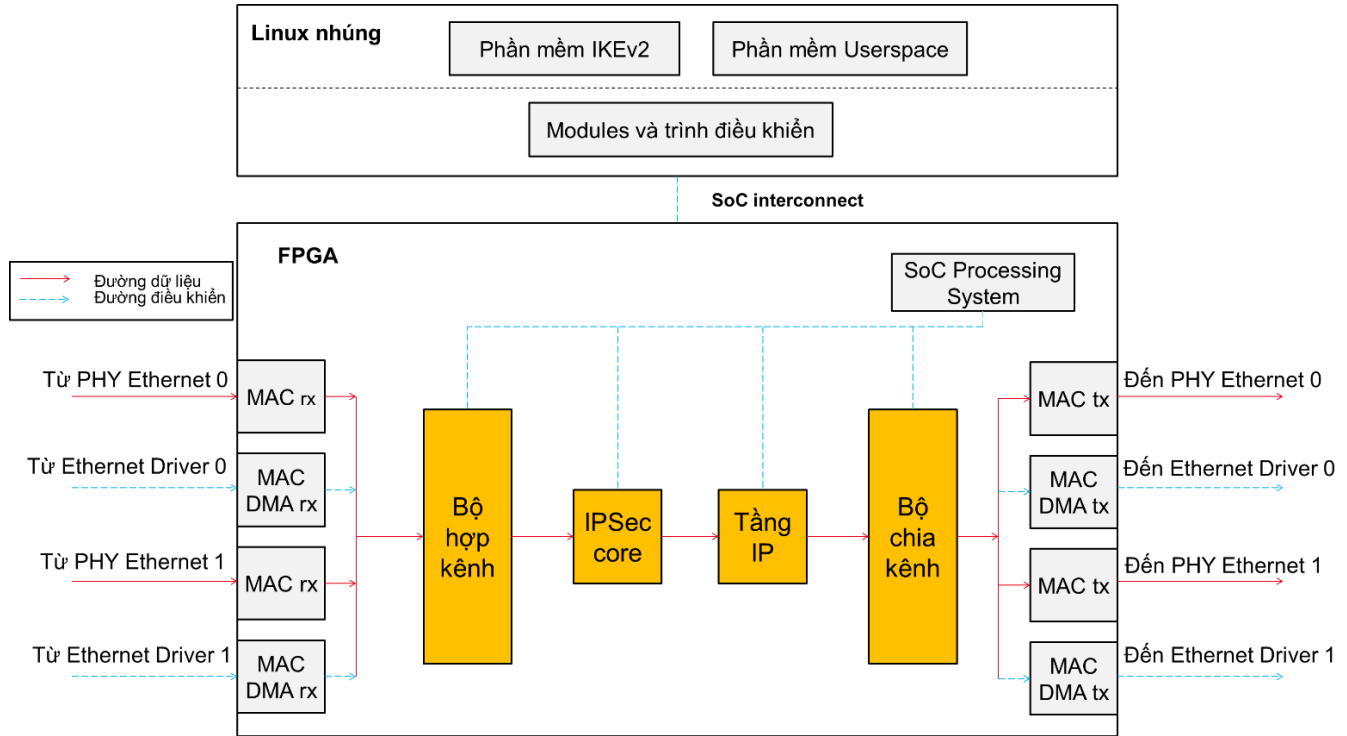
Trên cơ sở đó, HMS-IPSec được nghiên cứu nhằm giải quyết một cách hệ thống các hạn chế nêu trên trong các kiến trúc IPSec trên FPGA.

4.2. Nghiên cứu đề xuất kiến trúc phân cứng tổng thể

Kiến trúc tổng thể của HMS-IPSec được minh họa trong Hình 4.1, trong đó luồng dữ liệu được xử lý qua bốn thành phần chính: bộ hợp kênh, tầng IPSec, tầng IP, và bộ chia kênh. Luận án xây dựng các thuật toán cho từng thành phần này nhằm tối ưu độ trễ. Để đơn giản và đồng nhất, kiến trúc tập trung vào hai giao diện chính: Ethernet 0 (WAN) và Ethernet 1 (LAN), các giao diện LAN bổ sung được xử lý tương tự.

Tầng MAC (Medium Access Controller) thực hiện truyền nhận Ethernet Frame theo chuẩn IEEE 802.3-2008. Gói tin có thể đến hoặc đi từ PHY Ethernet hoặc Ethernet trên hệ điều hành nhúng. Để phân biệt gói tin đến từ cổng nào, các gói tin cần được gắn nhãn nguồn tương ứng. Bộ hợp kênh chuyển các gói tin từ các nguồn khác nhau vào một đường bus dữ liệu duy nhất. Tiếp theo IPSec core thực hiện các dịch vụ ESP cho toàn bộ phần Payload của Ethernet Frame. Sau đó các hoạt động ở tầng IP sẽ được thực hiện như tra bảng định tuyến, bảng ARP

(Address Resolution Protocol), tính IP checksum, xác định cổng ra của gói tin và gắn nhãn đích cho gói tin. Bộ chia kênh sẽ chuyển gói tin đến PHY Ethernet hoặc trình điều khiển Ethernet dựa trên nhãn đích tương ứng.



Hình 4.1. Kiến trúc HMS-IPSec

Kiến trúc HMS-IPSec sẽ có cả phiên bản tương đương của tầng IP và IPSec trên FPGA và hệ điều hành nhúng. Tất cả các thay đổi của tầng IP trên hệ điều hành nhúng sẽ đều được ánh xạ tức thời lên phần cứng FPGA bao gồm địa chỉ MAC, địa chỉ IP, bảng ARP, bảng định tuyến. Điều này tương tự với IPSec, các bảng cơ sở dữ liệu SPD, SAD có cả trên FPGA và nhân hệ điều hành, được cập nhật đồng bộ một cách tức thời. Các gói tin đến và đi từ thiết bị sẽ được xử lý bởi phần mềm như các giải pháp phần mềm khác. Các gói tin bảo mật sẽ được xử lý bởi phần cứng. Đường điều khiển được sử dụng để tương tác giữa phần mềm và các thành phần trên FPGA thông qua ánh xạ bộ nhớ của CPU (Memory Map). Các cấu hình, dữ liệu, các bảng trên tầng IP và IPSec giữa FPGA và hệ điều hành

nhúng sẽ được đồng bộ ngay lập tức qua đường điều khiển mỗi khi có thay đổi. Để phân biệt các gói tin giữa các giao diện mạng, HMS-IPSec sử dụng một trường thông tin gắn cho mỗi gói tin khi đi vào bus dữ liệu duy nhất. Định dạng của trường thông tin như sau:

Nhãn đích (4 bit)	Nhãn nguồn (4 bit)	Độ dài gói tin (16 bit)
----------------------	-----------------------	----------------------------

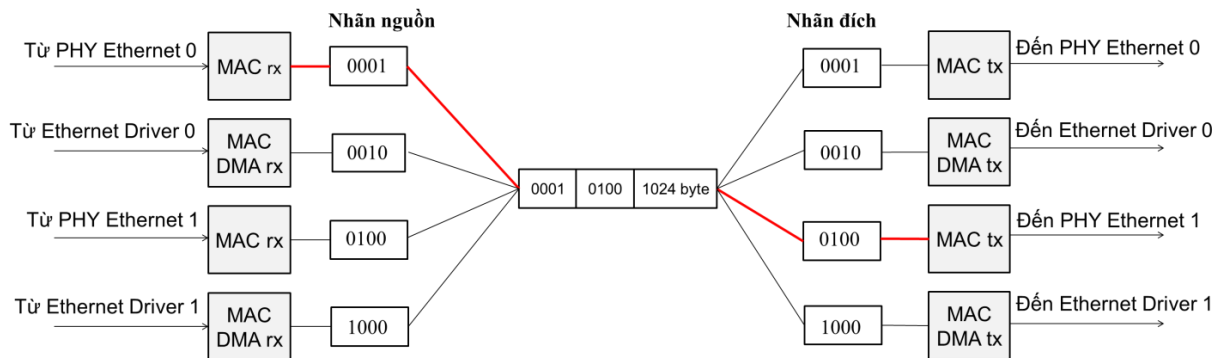
Hình 4.2. Cấu trúc trường thông tin

Vị trí bit được thiết lập giá trị 1 sẽ tương ứng với giao diện mạng, cụ thể bit chẵn là các giao diện mạng vật lý trên FPGA, bit lẻ là giao diện mạng trên hệ điều hành nhúng:

Bảng 4.1. Vị trí thiết lập các bit

Vị trí	Giao diện mạng
Bit 0	PHY Ethernet 0
Bit 1	DMA 0
Bit 2	PHY Ethernet 1
Bit 3	DMA 1

Dưới đây là trường hợp gói tin kích thước 1024 byte truyền trực tiếp từ giao diện mạng vật lý 0 sang giao diện mạng vật lý 1:



Hình 4.3. Gói tin đi qua các giao diện mạng

Bộ hợp kênh bao gồm 4 FIFO lưu các gói tin từ 4 giao diện mạng và chuyển các gói tin vào một đường dữ liệu duy nhất theo Thuật toán 4.1 dưới đây:

Thuật toán 4.1. Thuật toán bộ hợp kênh

1. Đọc trạng thái của các giao diện mạng
 2. **if** (gói tin có sẵn tại giao diện i) **then**
 3. **While** (gói tin chưa kết thúc)
 4. Đọc dữ liệu từ giao diện i
 5. Gán nhãn nguồn tương ứng cho gói tin
 6. Ghi dữ liệu vào bus dữ liệu chung
 7. Nhảy đến bước 1
 8. **else**
 9. Nhảy đến bước 1
 10. **end if**
-

Thuật toán tầng IP thực hiện việc tra cứu các bảng, chỉnh sửa IP Header và chuyển gói tin ra giao diện mạng tương ứng. Các giao diện mạng và các bảng trên phần cứng FPGA được cập nhập đồng bộ với phiên bản trên hệ điều hành. Do quá trình tìm kiếm bảng trên phần cứng có thể mất rất nhiều chu kỳ xung nhịp nếu các bảng này lớn. Do đó thuật toán đề xuất sử dụng kỹ thuật tìm kiếm CAM/TCAM [75] để tăng tốc khả năng tìm kiếm trong 1 chu kỳ. Thuật toán đề xuất cho tầng IP như sau:

Thuật toán 4.2. Thuật toán tầng IP

MAC 0: Địa chỉ MAC của giao diện mạng 0
MAC 1: Địa chỉ MAC của giao diện mạng 1
IPv4_0, IPv6_0: Địa chỉ IP của giao diện mạng 0
IPv4_1, IPv6_1: Địa chỉ IP của giao diện mạng 1
ARP_table: Bảng ARP
Neighbor_Cache: Bảng Neighbor
Route_table: Bảng định tuyến

Đầu vào: Gói tin IP

Đầu ra: Gói tin IP được định tuyến

1. Đọc nhãn nguồn và header gói tin từ bus dữ liệu.
2. Xác định phiên bản IP (IPv4 hoặc IPv6) từ trường version trong header.
3. **if** ((nhãn nguồn == 0010) || (nhãn nguồn == 1000)) **then**
4. // Gói tin đã được xử lý trên hệ điều hành nhúng.

```

5.      Gán nhãn đích giá trị 0001 (Ethernet 0) hoặc 0100 (Ethernet 1).
6.      Chuyển gói tin đi.
7.      Nhảy đến bước 1.
8.  else if (gói tin là IPv4) then
9.      // Xử lý gói tin IPv4.
10.     Đọc địa chỉ MAC đích, IP đích từ header IPv4.
11.     if ((MAC đích == MAC_0) || (MAC đích == MAC_1)) then
12.         if ((IP đích == IPv4_0) || (IP đích == IPv4_1)) then
13.             // Gói tin gửi đến hệ điều hành nhúng.
14.             Gán nhãn đích giá trị 0010 (DMA 0) hoặc 1000 (DMA 1).
15.             Chuyển gói tin đi.
16.             Nhảy đến bước 1.
17.         else
18.             // Tra bảng định tuyến cho IPv4.
19.             Tra bảng Route_table với IP đích bằng TCAM để xác định giao
20.             diện đích và next hop IPv4.
21.             Gán nhãn đích giá trị 0001 hoặc 0100 tương ứng với giao diện
22.             đích.
23.             // Tra bảng ARP cho next hop IPv4.
24.             Tra bảng ARP_table với next hop IPv4 sử dụng CAM để xác
25.             định địa chỉ MAC đích.
26.             Tạo MAC header mới với MAC nguồn (MAC_0 hoặc MAC_1)
27.             và MAC đích.
28.             Tạo IP header mới: giảm TTL đi 1, tính lại checksum.
29.             Thay thế MAC header và IP header mới cho gói tin IP.
30.             Chuyển gói tin đi.
31.             Nhảy đến bước 1.
32.         end if
33.     end if
34.     else
35.         Loại bỏ gói tin.
36.         Nhảy đến bước 1.
37.     end if
38. else if (gói tin là IPv6) then
39.     // Xử lý gói tin IPv6.

```

```

35.   Đọc địa chỉ MAC đích, IPv6 đích từ header IPv6.
36.   if ((MAC đích == MAC_0) || (MAC đích == MAC_1)) then
37.       if ((IPv6 đích == IPv6_0) || (IPv6 đích == IPv6_1) || (IPv6 đích là
38.           multicast)) then
39.           // Gói tin gửi đến hệ điều hành nhúng.
40.           Gán nhãn đích giá trị 0010 (DMA 0) hoặc 1000 (DMA 1).
41.           Chuyển gói tin đi.
42.           Nhảy đến bước 1.
43.       else
44.           // Tra bảng định tuyến cho IPv6.
45.           Tra bảng Route_table với IPv6 đích sử dụng TCAM để xác định
46.           giao diện đích và next hop IPv6.
47.           Gán nhãn đích giá trị 0001 hoặc 0100 tương ứng với giao diện
48.           đích.
49.           // Tra Neighbor Cache cho next hop IPv6.
50.           Tra bảng Neighbor_Cache với next hop IPv6 bằng CAM để xác
51.           định MAC đích.
52.           Tạo MAC header mới với MAC nguồn (MAC_0 hoặc MAC_1)
53.           và MAC đích.
54.           Tạo IPv6 header mới: giảm Hop Limit đi 1.
55.           Thay thế MAC header và IPv6 header mới cho gói tin.
56.           Chuyển gói tin đi.
57.           Nhảy đến bước 1.
58.       end if
59.   else
60.       Loại bỏ gói tin.
61.       Nhảy đến bước 1.
62.   end if

```

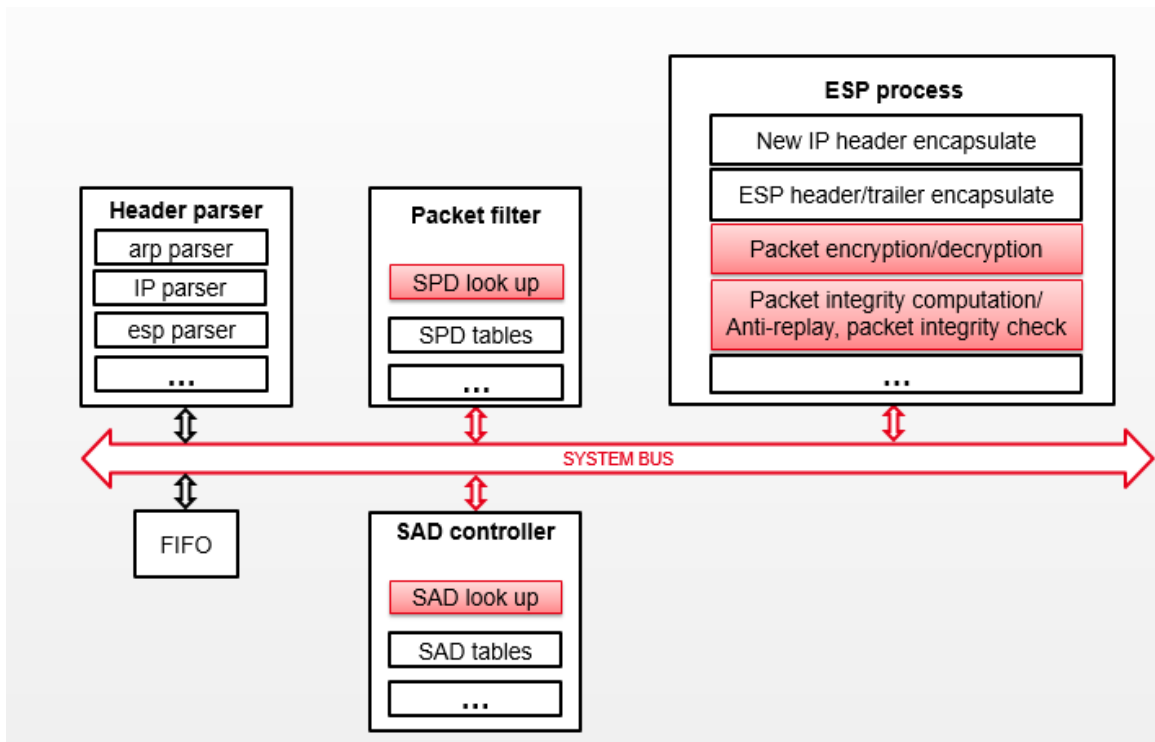
Bộ chia kênh cũng bao gồm FIFO lưu các gói tin và chuyển các gói tin vào giao diện mạng tương ứng với nhãn đích theo Thuật toán 4.3:

Thuật toán 4.3. Thuật toán bộ chia kênh

1. Đọc nhãn đích từ gói tin và xác định giao diện i .
 2. **While** (gói tin chưa kết thúc)
 3. Ghi dữ liệu vào $FIFO$ của giao diện i .
 4. **if** ($FIFO_i$ sẵn sàng) **then**
 5. Đọc dữ liệu từ $FIFO_i$.
 6. Ghi dữ liệu vào giao diện mạng i .
 7. **end if**
 8. Nhảy đến bước 1.
-

Đối với phần xử lý IPSec, tốc độ của giao thức này phụ thuộc phần lớn vào tốc độ tính toán mật mã trong đó là thuật toán mã hóa và thuật toán xác thực toàn vẹn. Trong các kết quả cài đặt IPSec tốc độ cao [58], [64], để nâng cao hiệu năng mã hóa, giải mã, nhiều khối AES được kết hợp lại với nhau. Chẳng hạn [58], [64] đều sử dụng đến 4 khối AES làm tăng số tài nguyên cài đặt và làm phức tạp hệ thống. Vấn đề này có thể được khắc phục bằng cách sử dụng kiến trúc tốc độ cao AES-P đã đề xuất ở Chương 3 của luận án. Khi đó chỉ cần 1 khối AES-P thay vì 4 khối, để cung cấp dịch vụ mã hóa cho toàn bộ thiết kế. Ngoài ra các quá trình xử lý giao thức: phân tích header, tra bảng cơ sở dữ liệu SPD, SAD, đóng gói gói tin ESP cũng chiếm khoảng 30% hiệu suất của IPSec. Chính vì lý do đó, một kiến trúc IPSec với thuật toán xử lý hiệu quả trên FPGA sẽ giúp cải thiện được hiệu năng.

Trong công trình công bố [J2], NCS đưa ra một kiến trúc IPSec tốc độ cao cho phần cứng bao gồm các thành phần chính sau:



Hình 4.4. Kiến trúc IPsec tốc độ cao trên phần cứng

Các thành phần chính gồm có:

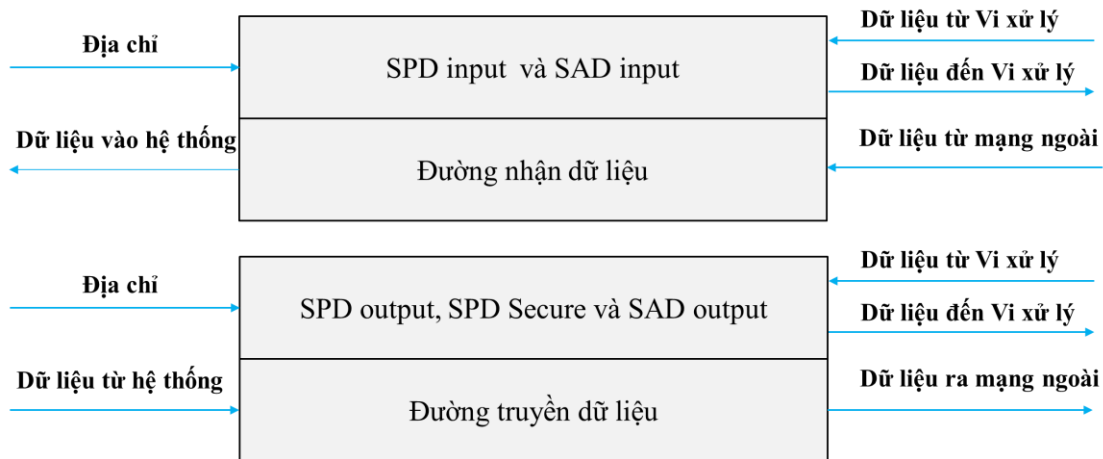
- **Header parser**: thực hiện phân tích header của gói tin và xác định loại gói tin, chẳng hạn ARP, IP, ESP hay các gói tin khác.
- **FIFO**: lưu trữ các gói tin, dữ liệu trong bộ nhớ khi chờ xử lý.
- **Packet filter**: kiểm tra các gói tin và áp dụng các chính sách cho phép, bảo vệ hay loại bỏ gói tin dựa trên các quy tắc đã được xác định từ trước trong cơ sở dữ liệu chính sách bảo mật SPD.
- **SAD controller**: lấy các khóa và tham số mật mã cần thiết để bảo vệ gói tin.
- **ESP process**: thực hiện các dịch vụ bí mật, xác thực, toàn vẹn của ESP.

Phần dưới đây trình bày chi tiết về kiến trúc IPsec tốc độ cao được đề xuất trên cơ sở của kiến trúc này.

4.3. Nghiên cứu đề xuất kiến trúc IPsec tốc độ cao

4.3.1. Đề xuất kiến trúc IPsec tốc độ cao

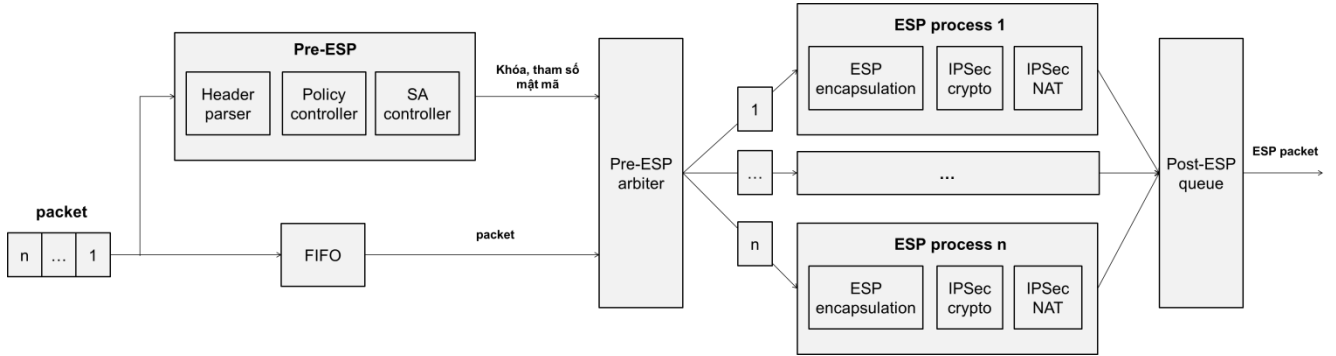
Về mặt giao tiếp, kiến trúc IPsec đề xuất có các các đường truyền, nhận dữ liệu và đường điều khiển (Hình 4.5). Đường truyền, nhận dữ liệu làm việc trực tiếp với các gói tin IP thông qua bus dữ liệu duy nhất sau khi đi qua bộ hợp kênh (Hình 4.1). Đường truyền, nhận dữ liệu có sử dụng bus có độ rộng 128 bit. NCS đề xuất sử dụng 5 bảng cơ sở dữ liệu cho IPsec trên FPGA bao gồm SPD output, SPD input, SPD secure, SAD output và SAD input. Các bảng này được cập nhập để đồng bộ với phiên bản trên hệ điều hành qua đường điều khiển có độ rộng 32 bit. Để tìm kiếm chính sách một cách hiệu quả cho gói tin trên các bảng SPD output, SPD input và SPD secure, TCAM được sử dụng, trong khi việc tìm kiếm các tham số mật mã trên bảng SAD output và SAD input được thực hiện bởi CAM. Cấu tạo của các bảng có thể xem phụ lục 2.



Hình 4.5. Các đường dữ liệu của kiến trúc IPsec

Kiến trúc thực hiện xử lý theo gói tin được chia làm hai giai đoạn *Pre-ESP* và *ESP process*. Giai đoạn *Pre-ESP* bao gồm ba thành phần: Header parser, Policy controller và SA controller. Giai đoạn *ESP process* gồm các thành phần: ESP

encapsulation, IPsec crypto và IPsec NAT. Kiến trúc cài đặt IPsec tốc độ cao đề xuất như sau:



Hình 4.6. Kiến trúc IPsec tốc độ cao đề xuất

Gói tin đi đến trước tiên sẽ được xử lý bởi *Pre-ESP*, đồng thời cũng được lưu ở FIFO. Giai đoạn này được thực hiện dựa vào các thông tin header của gói tin. Kết quả của giai đoạn này sẽ quyết định về chính sách của gói tin, cũng như khóa, tham số mật mã tương ứng cho gói tin cần bảo mật. Bởi quá trình tính toán được tập trung chủ yếu ở giai đoạn *ESP process*, nên tùy thuộc vào năng lực của các thuật toán mật mã, kiến trúc sẽ có một hoặc nhiều khối *ESP process*. Số khối *ESP process* cần đảm bảo được sự cân bằng giữa tài nguyên và tốc độ và có thể được tính theo công thức:

$$n = \frac{n_{ESP_process} \times 16}{MTU} \quad (4.1)$$

trong đó 16 là số byte mỗi chu kỳ xung nhịp đồng hồ tương ứng với bus 128-bit, *MTU* là kích thước lớn nhất của gói tin, $n_{ESP_process}$ là số chu kỳ xung nhịp đồng hồ cần thiết để thực hiện toàn bộ quá trình *ESP process* cho gói tin.

Khối *Pre-ESP arbiter* sử dụng thuật toán round-robin để lần lượt điều phối các gói tin cùng với khóa, tham số mật mã vào khối *ESP process* đã sẵn sàng. Các gói tin sau khi được xử lý bởi *ESP process* sẽ được gửi ra hàng đợi nhờ *Post-ESP queue*. Hoạt động của các thành phần trong hai giai đoạn như sau:

- **Header parser**: phân tích kiểu gói tin ARP, gói tin được bảo mật ESP hoặc gói tin không bảo mật cũng như nguồn của gói tin đến từ LAN, WAN hay hệ điều hành nhúng.

- **Policy controller**: thực hiện lọc các gói tin theo chính sách bằng cách kiểm tra sự trùng khớp của địa chỉ IP nguồn, địa chỉ IP đích, trường Protocol trong IP header, cổng UDP/TCP nguồn, cổng UDP/TCP đích. Ba bảng SPD được sử dụng bao gồm: *SPD output*, *SPD input* và *SPD secure* tương ứng với gói tin đi ra, gói tin đến và gói tin bảo mật. Policy controller sẽ tìm kiếm trong các SPD một chính sách khớp với gói tin. Một trong ba chính sách sẽ được áp dụng cho gói tin: BYPASS, DISCARD, hoặc PROTECT. Nếu không có chính sách nào được tìm thấy, IPSec sẽ loại bỏ gói tin và xử lý gói tiếp theo. Nếu gói tin khớp với chính sách BYPASS, nó sẽ được chuyển đi mà không cần xử lý thêm. Cuối cùng, nếu chính sách PROTECT được tìm thấy, gói tin sẽ được bảo vệ bằng IPSec ESP ở chế độ tunnel. Chính sách mặc định trong kiến trúc đề xuất là loại bỏ các gói tin. Chính sách này được coi là an toàn hơn chính sách chấp nhận mặc định. Policy controller sẽ chặn mọi gói tin không mong muốn, do đó ngăn chặn các tấn công từ chối dịch vụ. Để tìm kiếm hiệu quả chính sách trong SPD, bộ nhớ TCAM được sử dụng.

- **SA controller**: thực hiện tìm khóa, tham số mật mã cho gói tin từ SAD để mã hóa, giải mã, xác thực toàn vẹn gói tin. Nó cũng được sử dụng để cập nhật lại các giá trị trong SAD khi có yêu cầu (tăng số sequence number hoặc cập nhật lại giá trị của cửa sổ trượt chống tấn công phát lại). Hai bảng SAD được sử dụng bao gồm: *SAD input* và *SAD output* tương ứng với gói tin đến và gói tin đi ra. Khi một gói tin đi ra khớp với chính sách PROTECT trong SPD secure, một liên kết giữa SAD output và SPD secure được tạo ra thông qua giá trị SAD_ID. Giá trị này giúp cho SA controller có thể tìm được khóa, tham số mật mã tương ứng trong SAD

output. Đối với gói tin đi vào, nếu gói tin được xác định là ESP, SA controller sẽ tìm kiếm trong SAD input dựa vào giá trị SPI (Security Parameter Index) trong ESP header. Sau đó, gói tin sẽ được kiểm tra tính toàn vẹn và chống phát lại (anti-replay). Bởi mạng tốc độ cao có khả năng truyền được hàng triệu gói tin trong một giây, dẫn đến việc tràn giá trị 32 bit Sequence Number trong một khoảng thời gian rất ngắn. Kiến trúc đề xuất sử dụng 64 bit ESN (Extended Sequence Number) làm mặc định để tránh vấn đề này.

- **ESP encapsulation:** gói tin sẽ được đóng gói ESP ở chế độ tunnel. Đầu tiên nó tạo IP header mới với địa chỉ nguồn và đích của hai bên đường hầm. Sau đó ESP header và ESP trailer được tạo ra và chèn vào gói tin. Định dạng của ESP header và ESP trailer như sau:

Bảng 4.2. Định dạng gói ESP header và ESP trailer

Loại	Trường	Kích thước
ESP Header	Security parameter index (SPI)	32-bit
	Sequence number	32-bit
	IV	64-bit
	Payload	Thay đổi
ESP trailer	Padding	0-3 bytes
	Padding length	8-bit
	Next header	8-bit
	ICV	128-bit

- **IPSec crypto:** Thuật toán mã hóa AES-P và Ascon sử dụng các kiến trúc phần cứng của Chương 3. Đối với AES-P sử dụng chế độ GCM hoặc CTR để tận dụng khả năng xử lý song song của phần cứng. Các chế độ này cho phép tạo ra các khóa dòng 128-bit bằng cách mã hóa khối block counter 128-bit. Mỗi 128-bit block counter được cấu tạo bởi 32-bit nonce (CTR) hoặc 32-bit salt (GCM), 64-bit IV, và 32-bit counter [70], [71]. Giá trị counter được tăng cho đến khi đủ khóa dòng để mã hóa toàn bộ gói tin. Giá trị 32 bit nonce/salt được tạo ra cho mỗi SA

bởi IKEv2. Giá trị IV được sinh ngẫu nhiên khi SA được thiết lập ban đầu, sau đó cộng XOR với sequence number cho mỗi gói tin. Đối với thuật toán xác thực, toàn vẹn, kiến trúc đề xuất sử dụng GHASH cho AES-P-GCM hoặc HMAC-SHA256 cho AES-P-CTR.

- **IPSec NAT**: Các kết quả cài đặt IPSec tốc độ cao [58], [64] đều không hỗ trợ vượt NAT. Kiến trúc đề xuất khắc phục được hạn chế này bằng cách thêm thành phần xử lý NAT. Nếu không phát hiện NAT, gói tin ESP sẽ được chuyển đi mà không cần xử lý thêm. Ngược lại, nếu NAT được phát hiện, gói tin ESP sẽ được đóng gói UDP để có thể vượt NAT. Cổng nguồn và cổng đích trong UDP header được cung cấp bởi SA controller từ SAD output. IP header mới của gói ESP sẽ cần được cập nhập kích thước, trường protocol và tính lại checksum cho gói tin mới.

Thuật toán xử lý IPSec của kiến trúc đề xuất được thực hiện như sau:

Thuật toán 4.4. Thuật toán xử lý IPSec

SPD output: Bảng chính sách cho gói tin đi ra

SPD input: Bảng chính sách cho gói tin đến

SPD secure: Bảng chính sách cho gói tin bảo mật

SAD output: Bảng khóa, tham số mật mã cho gói tin đi ra

SAD input: Bảng khóa, tham số mật mã cho gói tin đến

Đầu vào: Gói tin IP

Đầu ra: Gói tin được xử lý IPSec

1. *Header parser* xác định loại gói tin, từ LAN/WAN hay từ thiết bị.
2. *Policy controller* tạo tuple gồm địa chỉ IP nguồn, IP đích, port nguồn, port đích, protocol của gói tin.
3. Tra tuple với bảng *SPD output*, *SPD input*, *SPD secure* sử dụng TCAM.
4. Lưu các kết quả của bước 1 và bước 3.
5. Đọc kết quả bước 4.
6. **if** ((gói tin là ARP) || (gói tin từ thiết bị) || (gói tin đến thiết bị)) **then**
7. Chuyển gói tin đi không cần xử lý gì thêm.
8. Nhảy đến bước 1.

```

9.  else if (gói tin từ LAN) then
10.      if (match SPD output) then
11.          Chuyển gói tin đi không cần xử lý gì thêm.
12.          Nhảy đến bước 1.
13.      else if (match SPD secure) then
14.          SA controller thực hiện tra bảng SAD output sử dụng sa_id từ
15.          SPD secure để lấy khóa, tham số mật mã.
16.          ESP encapsulation tạo IP header mới và chèn vào gói tin.
17.          ESP encapsulation tạo ESP header và ESP trailer và chèn vào
18.          gói tin.
19.          IPSec crypto thực hiện mã hóa gói tin.
20.          IPSec crypto thực hiện tính giá trị toàn vẹn ICV của gói tin và
21.          chèn vào gói tin.
22.          IPSec NAT kiểm tra thiết bị có nằm sau NAT.
23.          if (NAT) then
24.              Cập nhập IP header mới cho gói tin UDP ESP.
25.              Tạo và chèn UDP header.
26.              Chuyển gói tin UDP ESP đi.
27.              Nhảy đến bước 1.
28.          else
29.              Chuyển gói tin ESP đi.
30.              Nhảy đến bước 1.
31.          end if
32.      else
33.          Loại bỏ gói tin.
34.          Nhảy đến bước 1.
35.      end if
36.  else if (gói tin từ WAN) then
37.      if (match SPD input) then
38.          if (gói tin là ESP) then
39.              Lấy các giá trị spi, esn, icv từ gói tin.
40.              SA controller thực hiện tra bảng SAD input sử dụng spi để
41.              lấy khóa, tham số mật mã.
42.              if (match spi) then

```

38.			<i>SA controller</i> thực hiện kiểm tra <i>esn</i> với cửa sổ trượt và bitmap trong <i>SAD input</i> .
39.			if (gói tin phát lại) then
40.			Loại bỏ gói tin.
41.			Nhảy đến bước 1.
42.			else
43.			<i>IPSec crypto</i> thực hiện giải mã gói tin.
44.			<i>IPSec crypto</i> thực hiện tính giá trị toàn vẹn <i>icv_c</i> của gói tin.
45.			if (<i>icv_c</i> == <i>icv</i>) then
46.			<i>SA controller</i> cập nhập cửa sổ trượt và bitmap trong <i>SAD input</i> .
47.			Chuyển gói tin ESP đi.
48.			Nhảy đến bước 1.
49.			else
50.			Loại bỏ gói tin.
51.			Nhảy đến bước 1.
52.			end if
53.			end if
54.			else
55.			Loại bỏ gói tin.
56.			Nhảy đến bước 1.
57.			end if
58.			else if (gói tin là ESP_UDP) then
59.			Gỡ bỏ UDP header.
60.			Nhảy đến bước 35.
61.			else
62.			Chuyển gói tin đi không cần xử lý gì thêm.
63.			Nhảy đến bước 1.
64.			end if
65.		else	
66.			Loại bỏ gói tin.
67.			Nhảy đến bước 1.
68.		end if	
69.	else		

```

70. |      | Loại bỏ gói tin.
71. |      | Nhảy đến bước 1.
72. | end if

```

4.3.2. So sánh với các kết quả khác

NCS thực hiện cài đặt kiến trúc IPsec đã đề xuất với 1 khối *ESP process* trên chip Xilinx Zynq-7000 SoC, cho từng bộ thuật toán AES-P-GCM-16, AES-P-CTR-HMAC-SHA256 và Ascon-AEAD128. Tần số tối đa mà kiến trúc IPsec có thể đạt được đối với từng thuật toán là:

Bảng 4.3. Tần số tối đa của kiến trúc IPsec trên Zynq-7000 SoC

IPsec sử dụng thuật toán mật mã	Tần số tối đa của kiến trúc IPsec
AES-P-CTR-HMAC-SHA256	150 MHz
AES-P-GCM-16	185 MHz
Ascon-AEAD128	98,2 MHz

Bảng 4.4 là sự phân bố số chu kỳ xung nhịp đồng hồ cần thiết của các thành phần trong kiến trúc đối với gói tin có kích thước 1446 byte:

Bảng 4.4. Phân bố chu kỳ xung nhịp đồng hồ của các thành phần

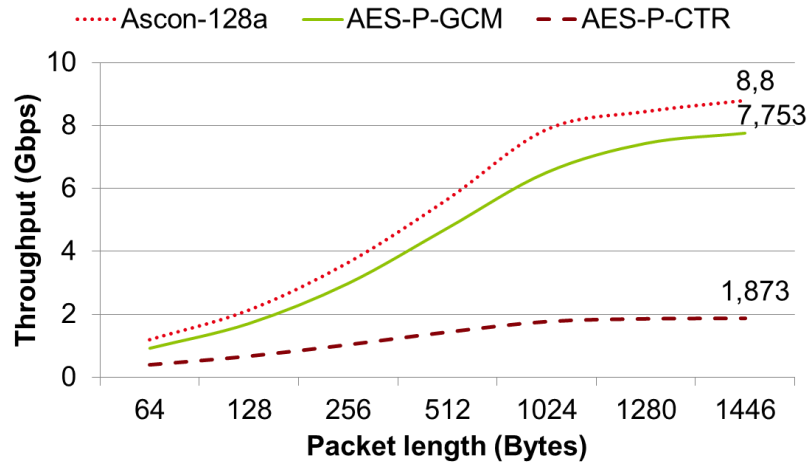
Quá trình	Số chu kỳ xung nhịp đồng hồ
Header parser, SPD, SAD lookup	16
New IP header and ESP header, ESP trailer encapsulation	11
IPsec-ESP state machine	6
Ascon-AEAD128	96
AES-P-GCM-16	243
AES-P-CTR-HMAC-SHA256	893

Tốc độ và độ trễ của kiến trúc được tính bằng công thức:

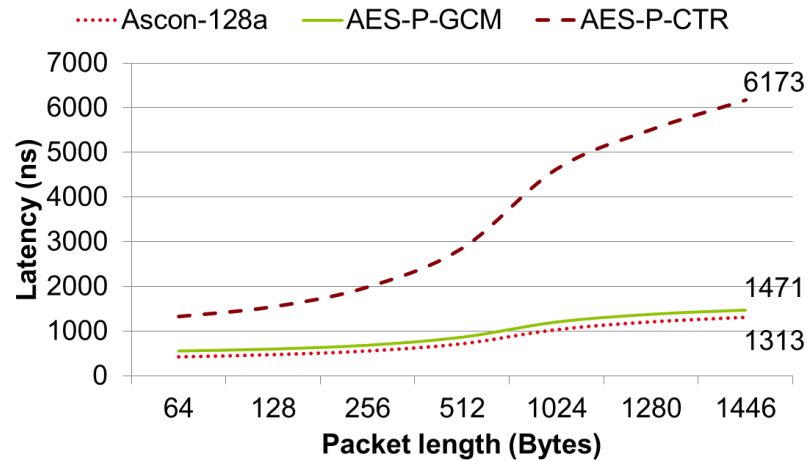
$$throughput = \frac{packet_size \times f_{max}}{n_{packet}} \quad (4.2)$$

$$Latency = \frac{n_{packet}}{f_{max}} \quad (4.3)$$

với $packet\ size$ là kích thước của gói tin, f_{max} là tần số tối đa của thiết kế và n_{packet} là số chu kỳ xung nhịp đồng hồ cần để thực hiện toàn bộ quá trình IPsec. Sử dụng công thức (4.2) và (4.3), khi đó tốc độ và độ trễ của kiến trúc IPsec đã đề xuất cho các gói tin có kích thước khác nhau từ 64 byte đến 1446 bytes được thể hiện ở Hình 4.7 và Hình 4.8.



Hình 4.7. Tốc độ với các gói tin kích thước khác nhau



Hình 4.8. Độ trễ với các gói tin kích thước khác nhau

Sự khác biệt đáng kể về hiệu năng giữa AES-P-CTR-HMAC-SHA256 và AES-P-GCM-16 chủ yếu bắt nguồn từ đặc tính của khối xác thực. Cụ thể, GHASH trong GCM có thể được pipeline và xử lý song song ở mức khối dữ liệu, trong khi

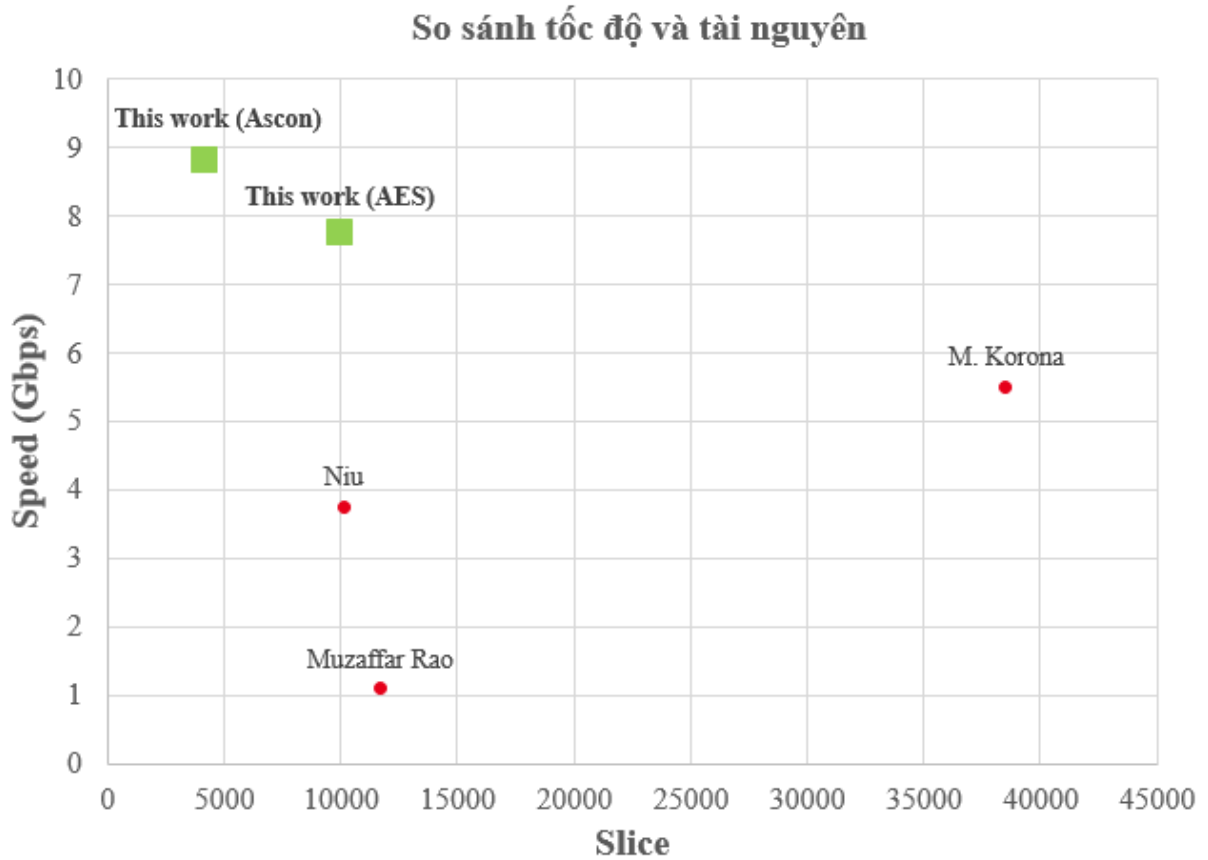
HMAC-SHA256 có sự phụ thuộc dữ liệu, dẫn đến quá trình xử lý mang tính tuần tự và làm suy giảm thông lượng toàn hệ thống.

Các tốc độ và độ trễ cho thấy sự hiệu quả của kiến trúc IPsec đã đề xuất. Bảng dưới đây là tổng hợp so sánh kiến trúc IPsec đề xuất với các công trình IPsec tốc độ cao. Cần lưu ý rằng Bảng 4.5 thực hiện so sánh các cài đặt IPsec trên nhiều nền tảng FPGA/SoC khác nhau (Zynq-7000, Virtex-5, Virtex-6, DE5/Stratix). Do sự khác biệt về công nghệ chế tạo và kiến trúc logic đặc thù của từng dòng chip, cấu tạo và năng lực xử lý của các đơn vị tài nguyên như LUT hay Slice là không đồng nhất. Để đảm bảo tính khách quan và giảm thiểu ảnh hưởng từ sự khác biệt nền tảng, luận án sử dụng chỉ số hiệu quả cài đặt Mbps/Slice:

Bảng 4.5. So sánh các kết quả cài đặt IPsec tốc độ cao

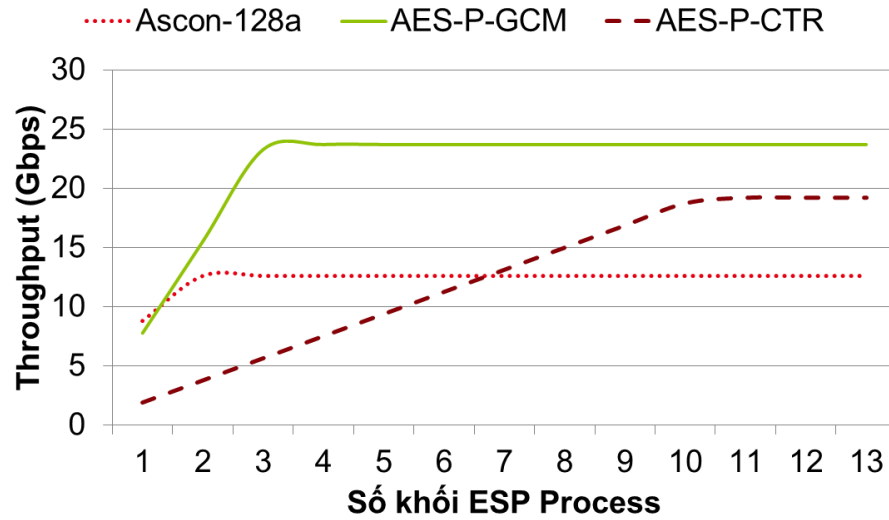
Cài đặt	Thiết bị	Số LUT	Số Slice	Throughput (Mbps)	Efficient (Mbps/slice)	Chế độ	Thuật toán	IKE	Mã hóa	Toàn vẹn	ESN	NAT
This work	Zynq-7000	30.035	9.948	7.753	0,779	tunnel	AES-P-GCM	có	có	có	có	có
This work	Zynq-7000	13.584	4.224	8.806	2,08	tunnel	Ascon-AEAD128	có	có	có	có	có
Niu [58]	Virtex-5	30.366	10.122	3.760	0,37	transport	AES	n/a	có	có	n/a	không
M. Korona [63]	DE5	115.600	38.533	5.518	0,14	tunnel	AES-CBC	n/a	có	có	n/a	không
Muzaffar Rao [64]	Virtex-6	35.079	11.693	1.110	0,094	tunnel	AES-CTR	n/a	có	không	n/a	không
Tuan [61]	Virtex-6	21.306	8.983	394	0,04	tunnel	AES	có	có	có	n/a	không
Agarwal [68]	NXP LX2160A SoC	n/a	n/a	1.400	n/a	tunnel	AES	có	có	có	n/a	không
Một số công trình khác												
Binh [60]	StratixV	n/a	637	4.874	7,65	không	SHA256	không	không	có	không	không
Kermiche [62]	Virtex-6	n/a	2.238	27.820	12,43	không	SHA-3	không	không	có	không	không

Lưu ý: Kết quả so sánh có sự khác biệt về nền tảng phần cứng (họ FPGA/SoC), phiên bản công cụ thiết kế và chiến lược tổng hợp/tối ưu hóa.



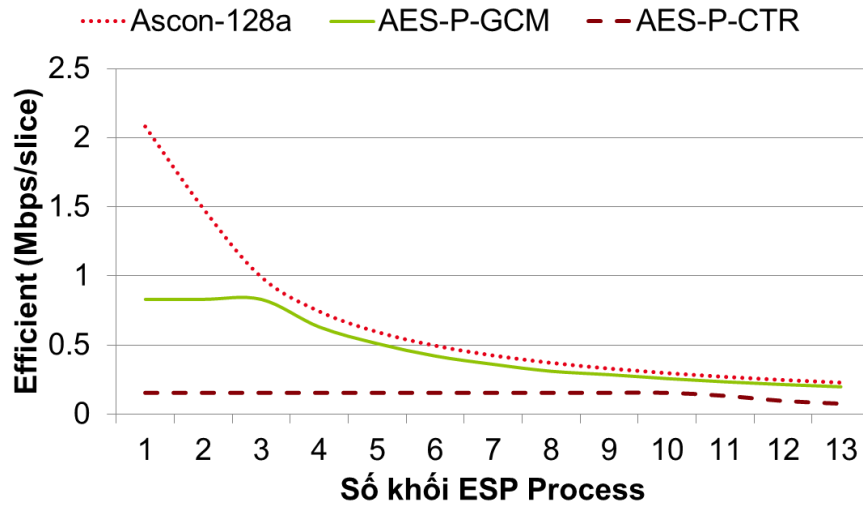
Hình 4.9. So sánh tốc độ và tài nguyên của kiến trúc IPsec sử dụng 1 khối ESP với các các thiết kế IPsec tốc độ cao khác

So sánh với các kết quả cài đặt IPsec khác, kiến trúc đề xuất có độ hiệu quả vượt trội chỉ với 1 khối *ESP process*, gấp 2,1 lần so với Niu [58], và 5,56 lần so với M. Korona [63]. Khi tăng số khối *ESP process*, tốc độ của kiến trúc sẽ tiệm cận tốc độ tối đa của xử lý rõ ở tần số f_{max} (Bảng 4.3) và độ trễ của kiến trúc sẽ không đổi. Hình 4.10 là biểu diễn sự phụ thuộc của tốc độ lên số khối *ESP Process* cho từng thuật toán.



Hình 4.10. Sự phụ thuộc của tốc độ IPsec lên số khối ESP Process

Tuy nhiên khi tăng số khối *ESP process*, độ hiệu quả cài đặt cũng sẽ giảm xuống. Hình 4.11 là biểu diễn sự phụ thuộc của hiệu quả cài đặt lên số khối *ESP Process* cho từng thuật toán.



Hình 4.11. Sự phụ thuộc của hiệu quả cài đặt lên số khối ESP Process

Số khối *ESP process* để kiến trúc có thể đạt được hiệu quả cài đặt cao nhất được thể hiện ở bảng Bảng 4.6:

Bảng 4.6. Số khối ESP tối ưu nhất

Thuật toán	Tốc độ tối đa (Gbps)	Số khối ESP process
AES-P-GCM	23,26	3
AES-P-CTR-HMAC-SHA256	18,73	10
Ascon-AEAD128	8,806	1

Độ trễ của kiến trúc được so sánh với các công trình khác trong Bảng 4.7:

Bảng 4.7. So sánh độ trễ của các công trình

Công trình	Độ trễ
Luận án (Ascon)	1,313 μs
Luận án (AES-P-GCM)	1,471 μs
Muzaffar Rao [64]	4,166 μs
Lastinec [66]	1,5 ms
Wang [67]	250,63 μs
Liu [69]	15 ms

Kiến trúc đề xuất đạt độ trễ 1,313 μs (Ascon-AEAD128) và 1,471 μs (AES-P-GCM), thấp hơn đáng kể so với [66], [67], [69]. Độ trễ này đáp ứng tốt các yêu cầu nghiêm ngặt của các ứng dụng thời gian thực, vốn đòi hỏi độ trễ dưới ngưỡng 1 ms để đảm bảo độ tin cậy, như đã được chỉ ra trong nghiên cứu của Jiang [9]. Đây là nghiên cứu đầu tiên đánh giá hiệu quả của thuật toán mã hóa xác thực hạng nhẹ Ascon-AEAD128 trong giao thức IPSec.

4.4. Đánh giá thực nghiệm

Dữ liệu trình bày từ Bảng 4.4 đến Bảng 4.7 phản ánh năng lực xử lý lý thuyết của khối IPSec core (được xác định thông qua các công thức (4.2) và (4.3)), không phải là hiệu năng thực tế đo đạc theo mô hình end-to-end.

Luận án tiến hành đánh giá thực nghiệm sử dụng các công nghệ phần cứng, IPCore và phần mềm sau:

Bảng 4.8. Các công nghệ sử dụng trong thực nghiệm

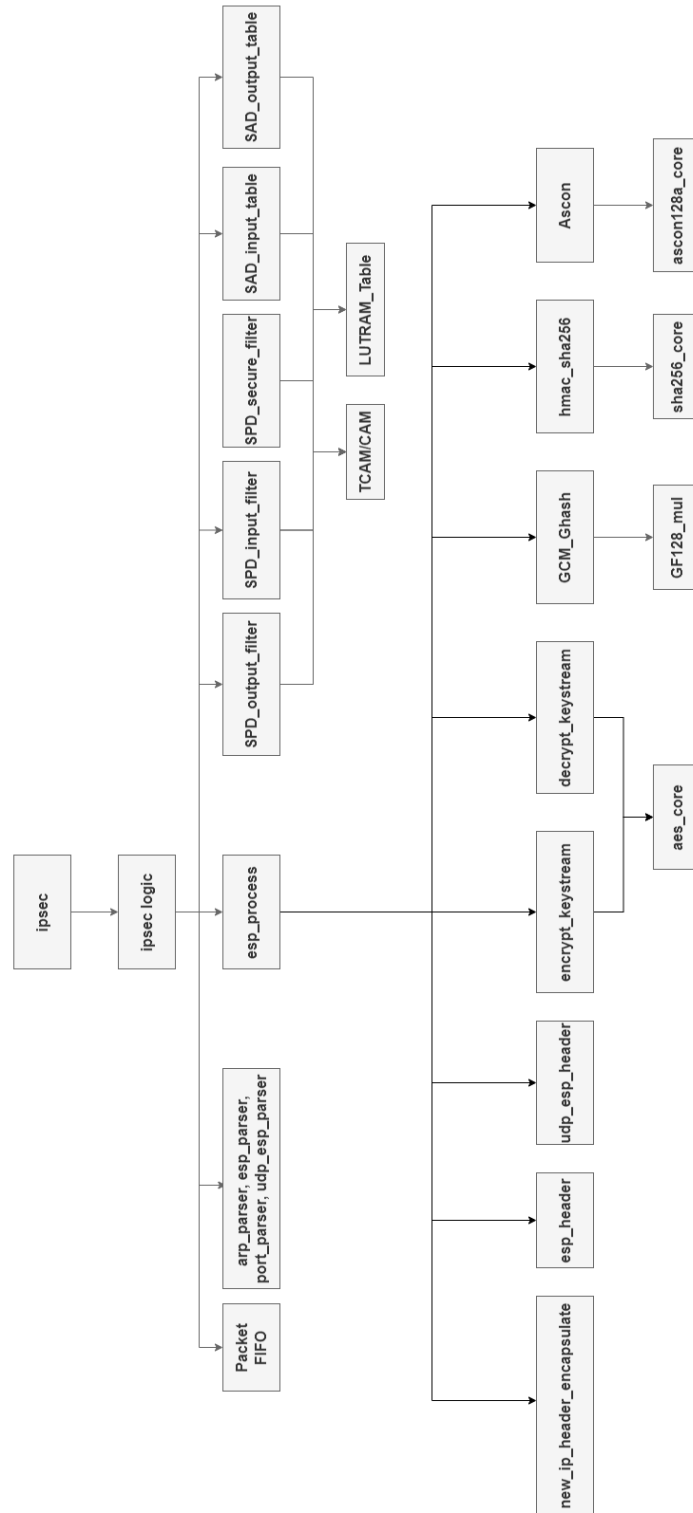
Công nghệ	Tên
Phần cứng	ZC706 Evaluation Kit
	Ethernet FMC
IPCore	Zynq7 processing system
	AXI 1G/2.5G Ethernet Subsystem
	AXI Direct Memory Access
OS và Driver	Petalinux 2015.4, Linux AXI Ethernet driver
Phần mềm	Strongswan 5.8.4

Việc sử dụng đồng bộ PetaLinux 2015.4 và StrongSwan 5.8.4 nhằm đảm bảo tính tương thích tối đa giữa hệ điều hành nhúng với các IP Core của Xilinx trên dòng chip Zynq-7000 (Kit ZC706). NCS thực hiện cài đặt HMS-IPSec với các thành phần như trên Bảng 4.9:

Bảng 4.9. Các IPCore sử dụng trong thiết kế

STT	Thành phần	IPCore	Nguồn gốc
0	SoC Processing System	ZYNQ7 Processing System	Xilinx
1	MAC rx và MAC tx	AXI 1G/2.5G Ethernet Subsystem	Xilinx
2	MAC DMA rx và MAC DMA tx	AXI Direct Memory Access	Xilinx
3	Bộ hợp kênh	port_mux_0	NCS cài đặt theo Thuật toán 4.1
4	IPSec	ipsec_0	NCS cài đặt theo Thuật toán 4.4
5	Tầng IP	IP_port_lookup_0	NCS cài đặt theo Thuật toán 4.2
6	Bộ chia kênh	port_demux_0	NCS cài đặt theo Thuật toán 4.3

Các khối từ 3 đến 6 được cài đặt theo các thuật toán bằng ngôn ngữ Verilog, sau đó được đóng gói thành các IPCore và ghép vào thiết kế. Các IPCore sử dụng bus giao tiếp chung AXI Stream cho đường dữ liệu, bus AXI Lite cho đường điều khiển. Để minh chứng đầy đủ các thuật toán, khối IPSec được cài đặt với kích thước bảng SPD/SAD là 64, sử dụng 1 khối *ESP process*, trong đó kết hợp cả ba thuật toán AES-P-GCM, AES-P-CTR-HMAC-SHA256 và Ascon-AEAD128. Dưới đây là phân lớp các modules của khối IPSec:



Thiết kế HMS-IPSec được tổng hợp và triển khai thành công dưới dạng bitstream, sau đó được nạp vào Kit phát triển ZC706 (Phụ lục 5). Phần mềm quản lý khóa IKE, dựa trên nền tảng StrongSwan, đã được tùy chỉnh để đảm bảo đồng bộ hóa các tham số từ các bảng SPD và SAD giữa nhân Linux nhúng và phần cứng FPGA mỗi khi có cập nhật. Hệ thống được triển khai trên hai Kit ZC706, thiết lập các kết nối IPSec tunnel giữa các mạng. Quá trình cấu hình các Kit ZC706 sau khi nạp thiết kế FPGA tương tự như các giải pháp IPSec dựa trên phần mềm, đảm bảo tính tương thích và linh hoạt.

Hiệu năng thực tế theo mô hình end-to-end đạt được 912 Mbps, tiệm cận với mạng 1Gbps. Tốc độ này đã tính cả sự ảnh hưởng của quá trình packet I/O, quá trình chuyển gói tin giữa các tầng mạng trong nhân của hệ điều hành trên các máy đầu cuối. Thực hiện tương tự với cài đặt IPSec mềm trong nhân Linux trên Kit ZC706 chỉ đạt được tốc độ 47,7 Mbps. Điều này minh chứng cho tính khả thi và sự hiệu quả của các giải pháp đã đề xuất trong luận án.

4.5. Kết luận chương 4

Chương 4 đã trình bày nghiên cứu đề xuất và triển khai kiến trúc phần cứng HMS-IPSec, đạt được những kết quả vượt trội trong việc nâng cao hiệu năng của giao thức IPSec trên FPGA. Kiến trúc HMS-IPSec có thể bảo mật luồng IP với các thuật toán AES-P-GCM, AES-P-CTR-HMAC-SHA256 và Ascon-AEAD128, trong đó thông lượng tối đa lên đến 23,26 Gbps, độ trễ 1.471 ns cho AES-P-GCM. Thuật toán Ascon-AEAD128 có độ trễ thấp 1.313 ns và hiệu quả cài đặt rất cao (2,08 Mbps/Slice), gấp 2,1 lần so với [69] và 5,56 lần so với [71]. Điều cho thấy HMS-IPSec không chỉ đáp ứng các yêu cầu về tốc độ và độ trễ thấp cho các ứng dụng thời gian thực, bảo mật video, mà còn phù hợp với các hệ thống tính toán biên nhờ mức tiêu thụ tài nguyên hợp lý.

KẾT LUẬN

Mạng IP đóng vai trò quan trọng trong cuộc cách mạng công nghiệp 4.0, giúp kết nối hàng tỷ thiết bị thông minh, từ cảm biến công nghiệp, thiết bị gia dụng, đến các phương tiện tự hành. Tuy nhiên nó cũng đang đặt ra các thách thức lớn đối với các giải pháp bảo mật mạng truyền thống. Một trong thách thức lớn là vấn đề hiệu suất, chẳng hạn mạng 6G được kỳ vọng mang lại tốc độ truyền dữ liệu cao hơn nhiều lần so với 5G, đồng thời giảm độ trễ xuống mức cực thấp, tiệm cận thời gian thực. Những yêu cầu này phải được đáp ứng, đặc biệt là trên những thiết bị trung tâm, nơi xử lý số lượng lớn dữ liệu giữa các hệ thống. Luận án này tập trung nghiên cứu các giải pháp nâng cao hiệu năng của một số thuật toán mã hoá trong bảo mật luồng IP, đáp ứng nhu cầu bảo mật cho các hệ thống mạng hiện đại.

I. Đóng góp mới của luận án

1) *Nâng cao hiệu năng của thuật toán mã hoá AES-P và ASCON*: Luận án đề xuất một kiến trúc AES-P đường ống 4 tầng, trong đó sử dụng các thành phần mật mã mới, bao gồm S-box S_p và ma trận MDS M_p , được thiết kế theo hướng thân thiện với phần cứng. Kiến trúc AES-P đạt thông lượng 105,7 Gbps với độ trễ là 48,4 ns, cải thiện đến 53% về độ trễ so với các công trình liên quan trên cùng nền tảng FPGA Virtex-7. Độ an toàn của AES-P vẫn được bảo toàn đối với các tấn công vi sai và tuyến tính, đồng thời nâng cao khả năng kháng tấn công đại số, hướng tới chống lại các tấn công đại số có thể thực thi trên máy tính lượng tử. Bên cạnh đó, luận án đề xuất một kiến trúc mới cho Ascon, bao gồm lõi thuật toán và khối giao tiếp, giúp đạt thông lượng 13,312 Gbps trong 1 chu kỳ, loại bỏ sự phụ thuộc vào bộ nhớ và phù hợp cho các hệ thống IoT thời gian thực. Các kết quả này được công bố trong các công trình [J1], [J2], [C1], [C2].

2) Đề xuất kiến trúc hệ thống HMS-IPSec trên thiết bị SoC: Luận án thiết kế và xây dựng kiến trúc phần cứng HMS-IPSec tích hợp thống nhất các bộ thuật toán AES-P-GCM, AES-P-CTR-HMAC-SHA256 và Ascon-AEAD128. Điểm nổi bật của kiến trúc là sự phân tách triệt để giữa đường dữ liệu và đường điều khiển, kết hợp với việc hỗ trợ đầy đủ các tính năng quan trọng như cơ chế vượt NAT, ESN 64-bit, giúp khắc phục các hạn chế về khả năng triển khai thực tế trong các công trình trước đây. Kiến trúc cho phép chuyển đổi linh hoạt giữa AES-P và Ascon mà không làm thay đổi cấu trúc xử lý gói tin, qua đó đảm bảo đồng thời yêu cầu về thông lượng lớn, độ trễ thấp cho các ứng dụng thời gian thực. Đây là nghiên cứu đầu tiên đánh giá một cách hệ thống tính khả thi và hiệu quả của việc tích hợp thuật toán mã hoá xác thực hạng nhẹ Ascon-AEAD128 vào giao thức IPSec trên nền tảng phần cứng SoC. Các kết quả nghiên cứu này đã được công bố trong công trình [J2].

II. Hướng nghiên cứu tiếp theo

Mặc dù luận án đã đạt được những kết quả đáng kể trong việc cải thiện hiệu năng và khả năng triển khai thực tế của các thuật toán mã hoá và giao thức IPSec trên phần cứng, vẫn còn một số vấn đề mở cần tiếp tục được nghiên cứu và hoàn thiện trong tương lai:

1. Nghiên cứu và tích hợp các kỹ thuật chống tấn công kênh kề:

Các kiến trúc phần cứng đề xuất trong luận án hiện tập trung cải thiện về tốc độ, độ trễ và tài nguyên. Việc triển khai trên thiết bị vật lý luôn tiềm ẩn nguy cơ rò rỉ thông tin qua điện năng tiêu thụ hoặc bức xạ điện từ. Hướng nghiên cứu tiếp theo sẽ tập trung phân tích và tích hợp các kỹ thuật phòng chống tấn công kênh kề (như kỹ thuật che giấu - masking, hoặc làm phẳng năng lượng - hiding) vào kiến trúc AES và Ascon. Mục tiêu là đạt được sự cân bằng giữa hiệu năng xử

lý cao và khả năng kháng lại các tấn công phân tích năng lượng (DPA/CPA) mà không làm gia tăng quá mức tài nguyên phần cứng.

2. Mở rộng hỗ trợ các thuật toán mật mã hậu lượng tử:

Trước sự phát triển của máy tính lượng tử, bên cạnh các thuật toán mã hoá, cần nghiên cứu tích hợp các thuật toán chữ ký số và trao đổi khóa hậu lượng tử (như Kyber, Dilithium) vào giao thức trao đổi khóa IKEv2 trong kiến trúc HMS-IPSec. Việc này nhằm đảm bảo tính bảo mật dài hạn cho hệ thống trước các mối đe dọa lượng tử trong tương lai.

3. Nghiên cứu phát triển các kiến trúc trên các nền tảng phần cứng ASIC và công nghệ mới:

Hiện tại, các thiết kế mới chỉ được đánh giá và triển khai trên nền tảng FPGA. Hướng phát triển tiếp theo sẽ là tiếp tục nghiên cứu, phát triển kiến trúc HMS-IPSec trên công nghệ ASIC (Application-Specific Integrated Circuit) để khẳng định tính khả thi trong sản xuất chip chuyên dụng. Đồng thời, nghiên cứu khả năng thích ứng của kiến trúc với các công nghệ mạng mới như 5G/6G hoặc các hệ thống tính toán biên (Edge Computing) với yêu cầu khắt khe hơn về năng lượng và độ trễ cũng là một hướng đi tiềm năng.

DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC ĐÃ CÔNG BỐ

I. CÔNG TRÌNH CÔNG BỐ LIÊN QUAN ĐẾN LUẬN ÁN

- [J1] Nam, T. S., Dung, L. T., & Long, N. V. (2024). *A High Throughput, Low Latency 105 Gbps Four-Pipeline Stage AES*. Journal of Science and Technology on Information Security, 1(21), 58-66. (HĐGSNN, 0.75)
- [J2] Nam, Tran Sy, Hoang Van Thuc, and Bui Duy Hieu. "A Hardware Architecture of NIST Lightweight Cryptography applied in IPsec to Secure High-throughput Low-latency IoT Networks." IEEE Access (2023). (Q1, IF = 3.4).
- [J3] Nam, T. S., Long, N. V., & Cương, N. B. (2023). *An efficient and secure linear diffusion layer for 256-bit block cipher based on FLC structure*. Journal of Science and Technology on Information Security, 2(16), 31-38. (HĐGSNN, 0.75)
- [C1] Nam, Tran Sy, Le Quoc Dat, Nguyen Van Long, and Nguyen Bui Cuong. "An Optimized Bit-Slice Implementation of Secure 8-Bit Sbox Based on Butterfly Structure." 2023 15th International Conference on Knowledge and Systems Engineering (KSE). IEEE, 2023. (ISBN 979-8-3503-2974-2)
- [C2] Nguyen, Cuong, Nam Tran, and Long Nguyen. "FLC: A New Secure and Efficient SPN-Based Scheme for Block Ciphers." 2022 9th NAFOSTED Conference on Information and Computer Science (NICS). IEEE, 2022. (ISBN 978-1-6654-5422-3)

II. CÔNG TRÌNH CÔNG BỐ KHÁC TRONG QUÁ TRÌNH THỰC HIỆN LUẬN ÁN

- [C3] Van Nghi, Nguyen, Tran Sy Nam, Tran Thi Hanh, and Hoang Van Thuc. "A Flexible and Balanced Hardware Architecture of RSA Modulo Exponentiation on FPGA." In The International Conference on Intelligent Systems & Networks, pp. 416-422. Singapore: Springer Nature Singapore, 2024. (ISBN 978-981-97-5504-2)

MỤC LỤC THAM KHẢO

Tài liệu tiếng Việt

- [1] Nguyễn Bùi Cương (2018). *Nghiên cứu xây dựng các thành phần mật mã cho thuật toán mã khối hạng nhẹ* (Luận án Tiến sĩ).
- [2] Đồng Phạm Khôi (2024). *Giải pháp kiến trúc phần cứng bảo mật AES hiệu quả cao, công suất thấp dùng cho các thiết bị internet vạn vật* (Luận án Tiến sĩ).

Tài liệu tiếng nước ngoài

- [3] Skiadopoulos, Athinagoras, et al. "High-throughput and flexible host networking for accelerated computing." *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 2024.
- [4] Zhao, Zhipeng, et al. "Achieving 100gbps intrusion prevention on a single server." *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 2020.
- [5] Hauger, Simon, et al. "Packet processing at 100 Gbps and beyond-challenges and perspectives." *2009 ITG Symposium on Photonic Networks*. VDE, 2009.
- [6] National Institute of Standards and Technology, "Ascon-Based Lightweight Cryptography Standards for Constrained Devices," *NIST Special Publication 800-232*, Aug. 2025. doi: 10.6028/NIST.SP.800-232.
- [7] Rukhin, Andrew, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *No. NISTSP80002*. 2001.
- [8] Burek, Elżbieta, et al. "Algebraic attacks on block ciphers using quantum annealing." *IEEE Transactions on Emerging Topics in Computing* 10.2 (2022): 678-689.
- [9] Jiang, Xiaolin, et al. "Low-latency networking: Where latency lurks and how to tame it." *Proceedings of the IEEE* 107.2 (2018): 280-306.
- [10] Vansteenkiste, Elias. New FPGA design tools and architectures. *Diss. Ghent University*, 2016.
- [11] National Institute of Standards and Technology. Advanced Encryption Standard (AES). *FIPS PUB 197, 2001*. U.S. Department of Commerce.
- [12] Daemen, Joan, and Vincent Rijmen. The design of Rijndael. Vol. 2. *New York: Springer-verlag*, 2002.
- [13] Hatzivasilis, George, et al. "A review of lightweight block ciphers." *Journal of cryptographic Engineering* 8.2 (2018): 141-184.
- [14] Gaj, Kris, and Paweł Chodowiec. "FPGA and ASIC implementations of AES." *Cryptographic engineering*. Boston, MA: Springer US, 2009. 235-294.
- [15] Canright, David. "A very compact S-box for AES." *International Workshop on Cryptographic Hardware and Embedded Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

- [16] Nakashima, Ayano, Rei Ueno, and Naofumi Homma. "AES S-box hardware with efficiency improvement based on linear mapping optimization." *IEEE Transactions on Circuits and Systems II: Express Briefs* 69.10 (2022): 3978-3982.
- [17] Ivanov, Georgi, Nikolay Nikolov, and Svetla Nikova. "Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties." *Cryptography and Communications* 8.2 (2016): 247-276.
- [18] Gorbenko, Ivan, et al. "Random S-boxes generation methods for symmetric cryptography." *2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON)*. IEEE, 2019.
- [19] Mahboob, Abid, Muhammad Nadeem, and Muhammad Waheed Rasheed. "A study of text-theoretical approach to S-box construction with image encryption applications." *Scientific Reports* 13.1 (2023): 21081.
- [20] Long, Min, and Longlong Wang. "S-box design based on discrete chaotic map and improved artificial bee colony algorithm." *IEEE Access* 9 (2021): 86144-86154.
- [21] Zhang, Lijun, et al. "A novel dynamic S-box generation scheme based on quantum random walks controlled by a hyper-chaotic map." *Mathematics* 12.1 (2023): 84.
- [22] Nizam Chew, Liyana Chew, and Eddie Shahril Ismail. "S-box construction based on linear fractional transformation and permutation function." *Symmetry* 12.5 (2020): 826.
- [23] Corona-Bermúdez, Erendira, et al. "Chaos meets cryptography: Developing an S-box design with the Rössler attractor." *Mathematics* 11.22 (2023): 4575.
- [24] Baowidan, Souad Ahmad, et al. "Group-Action-Based S-box Generation Technique for Enhanced Block Cipher Security and Robust Image Encryption Scheme." *Symmetry* (20738994) 16.8 (2024).
- [25] Su, Yuyue, et al. "A new S-box three-layer optimization method and its application." *nonlinear Dynamics* 111.3 (2023): 2841-2867.
- [26] Izbenko, Yuriy, Vladislav Kovtun, and Alexandr Kuznetsov. "The design of boolean functions by modified hill climbing method." *2009 Sixth International Conference on Information Technology: New Generations*. IEEE, 2009.
- [27] Fomin, Denis Bonislavovich. "New classes of 8-bit permutations based on a butterfly structure." *Математические вопросы криптографии* 10.2 (2019): 169-180.
- [28] Perrin, Léo, Aleksei Udovenko, and Alex Biryukov. "Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem." *Annual International Cryptology Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.
- [29] Biryukov, Alex, Léo Perrin, and Aleksei Udovenko. "Reverse-engineering the S-box of Streebog, Kuznyechik and STRIBOBr1." *Annual international conference on the theory and applications of cryptographic techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.
- [30] Teng, You-Tun, et al. "VLSI architecture of S-box with high area efficiency based on composite field arithmetic." *IEEE Access* 10 (2021): 2721-2728.

- [31] Li, Yongqiang, and Mingsheng Wang. "On the construction of lightweight circulant involutory MDS matrices." *International Conference on Fast Software Encryption*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.
- [32] Pehlivanoglu, Meltem Kurt, et al. "Generalisation of Hadamard matrix to generate involutory MDS matrices for lightweight cryptography." *IET Information Security* 12.4 (2018): 348-355.
- [33] Gupta, Kishan Chand, and Indranil Ghosh Ray. "On constructions of MDS matrices from companion matrices for lightweight cryptography." *International Conference on Availability, Reliability, and Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.
- [34] Liu, Qiang, Zhenyu Xu, and Ye Yuan. "A 66.1 Gbps single-pipeline AES on FPGA." *2013 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2013.
- [35] Liu, Qiang, Zhenyu Xu, and Ye Yuan. "High throughput and secure advanced encryption standard on field programmable gate array with fine pipelining and enhanced key expansion." *IET Computers & Digital Techniques* 9.3 (2015): 175-184.
- [36] Soltani, Abolfazl, and Saeed Sharifian. "An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA." *Microprocessors and Microsystems* 39.7 (2015): 480-493.
- [37] Oukili, Soufiane, and Seddik Bri. "Hardware implementation of AES algorithm with logic S-box." *Journal of Circuits, Systems and Computers* 26.09 (2017): 1750141.
- [38] Baby Chellam, Manjith, and Ramasubramanian Natarajan. "AES hardware accelerator on FPGA with improved throughput and resource efficiency." *Arabian Journal for Science and Engineering* 43.12 (2018): 6873-6890.
- [39] Visconti, Paolo, et al. "High-performance AES-128 algorithm implementation by FPGA-based SoC for 5G communications." *International Journal of Electrical and Computer Engineering (IJECE)* 11.5 (2021): 4221-4232.
- [40] Kumar, Thanikodi Manoj, et al. "A low area high speed FPGA implementation of AES architecture for cryptography application." *Electronics* 10.16 (2021): 2023.
- [41] Priya, S. Sridevi Sathya, P. Karthigaikumar, and Narayana Ravi Teja. "FPGA implementation of AES algorithm for high speed applications." *Analog integrated circuits and signal processing* 112.1 (2022): 115-125.
- [42] Rahim, Usva, et al. "Architectural implementation of AES based 5G security protocol on FPGA." *2022 32nd International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2022.
- [43] Savalam, Chandrasekhar, and Prasanti Korapati. "Implementation and design of AES S-box on FPGA." *Int. J. Res. Eng. Sci.* 3.1 (2015): 1-9.
- [44] Groß, Hannes, et al. "Suit up!--made-to-measure hardware implementations of Ascon." *2015 Euromicro Conference on Digital System Design*. IEEE, 2015.
- [45] Yalla, Panasayya, and Jens-Peter Kaps. "Evaluation of the CAESAR hardware API for lightweight implementations." *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017.

- [46] Rezvani, Behnaz, et al. "Hardware implementations of NIST lightweight cryptographic candidates: A first look." *Cryptology ePrint Archive* (2019).
- [47] Samir, Nagham, et al. "ASIC and FPGA comparative study for IoT lightweight hardware security algorithms." *Journal of Circuits, Systems and Computers* 28.12 (2019): 1930009.
- [48] Soliman, Shady, et al. "FPGA implementation of dynamically reconfigurable IoT security module using algorithm hopping." *Integration* 68 (2019): 108-121.
- [49] Abebe, Abiy Tadesse, Yalemzewd Negash Shiferaw, and PGV Suresh Kumar. "Efficient reconfigurable integrated cryptosystems for cybersecurity protection." *Advances in Cyber Security Analytics and Decision Systems*. Cham: Springer International Publishing, 2020. 57-77.
- [50] Khan, Safiullah, Wai-Kong Lee, and Seong Oun Hwang. "Scalable and efficient hardware architectures for authenticated encryption in IoT applications." *IEEE Internet of Things Journal* 8.14 (2021): 11260-11275.
- [51] Kaur, Jasmin, Mehran Mozaffari Kermani, and Reza Azarderakhsh. "Hardware constructions for error detection in lightweight authenticated cipher ASCON benchmarked on FPGA." *IEEE Transactions on Circuits and Systems II: Express Briefs* 69.4 (2021): 2276-2280.
- [52] Wei, Xiangdong, et al. "Reco-hcon: A high-throughput reconfigurable compact ascon processor for trusted iot." *2022 IEEE 35th International System-on-Chip Conference (SOCC)*. IEEE, 2022.
- [53] Khan, Safiullah, Wai-Kong Lee, and Seong Oun Hwang. "Evaluating the performance of ascon lightweight authenticated encryption for ai-enabled iot devices." *2022 TRON Symposium (TRONSHOW)*. IEEE, 2022.
- [54] ATHENA, Database of FPGA Results for Authenticated Ciphers. [Online]. Available: https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/results/468
- [55] Kaps, J.P., Diehl, W., Tempelmeier, M., Homsirikamol, E., Gaj, K.: *Hardware API for Lightweight Cryptography* (Oct 2019), https://cryptography.gmu.edu/athena/LWC/LWC_HW_API.pdf
- [56] Ascon128a Repository on GitHub. [Online]. Available: <https://github.com/synam/Ascon128a>
- [57] Kent, Stephen, and Karen Seo. *Security architecture for the internet protocol*. No. rfc4301. 2005.
- [58] Niu, Yun, Liji Wu, and Xiangmin Zhang. "An ipsec accelerator design for a 10gbps in-line security network processor." *J. Comput.* 8.2 (2013): 319-325.
- [59] Rao, Muzaffar, Joseph Coleman, and Thomas Newe. "An FPGA based reconfigurable IPSec ESP core suitable for IoT applications." *2016 10th International Conference on Sensing Technology (ICST)*. IEEE, 2016.
- [60] Kieu-Do-Nguyen, Binh, et al. "High-performance multi-function HMAC-SHA2 FPGA implementation." *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*. IEEE, 2022.
- [61] Nguyen, Trong-Tuan, et al. "Performance enhancement of encryption and authentication IP cores for IPSec based on multiple-core architecture and dynamic partial reconfiguration on FPGA." *2018 2nd International Conference on Recent Advances in Signal Processing, Telecommunications & Computing (SigTelCom)*. IEEE, 2018.

- [62] Kermiche, Akram, Mohamed Saoudi, and Amine Drouiche. "High throughput pipelined implementation of SHA3 hash algorithm on FPGA." *Journal of Cryptographic Engineering* 15.2 (2025): 15.
- [63] Korona, Mateusz, et al. "FPGA implementation of IPsec protocol suite for multigigabit networks." *2017 International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, 2017.
- [64] Rao, Muzaffar, et al. "An efficient implementation of FPGA based high speed IPsec (AH/ESP) core." *International Journal of Internet Protocol Technology* 11.2 (2018): 97-109.
- [65] Ullah, Sami, Joontae Choi, and Heekuck Oh. "IPsec for high speed network links: Performance analysis and enhancements." *Future Generation Computer Systems* 107 (2020): 112-125.
- [66] Lastinec, Jan, and Ladislav Hudec. "A study of securing in-vehicle communication using IPSEC protocol." *Journal of Electrical Engineering* 72.2 (2021): 89-98.
- [67] Wang, Qianran, Jinhua Wang, and Chengbin Huang. "The Optimization of IPsec VPN in 5G Mobile Communication Network." *Proceedings of the 2023 2nd International Conference on Networks, Communications and Information Technology*. 2023.
- [68] Agarwal, Ayushi, et al. "APPAMM: Memory Management for IPsec Application on Heterogeneous SoCs." *2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2024.
- [69] Liu, Jihong, Chenyang Tu, and Yifei Zhang. "Hardware assisted security gateway system: combined with FPGA shielding protection." *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2024.
- [70] Housley, Russell. *Using advanced encryption standard (aes) counter mode with ipsec encapsulating security payload (esp)*. No. rfc3686. 2004.
- [71] Viega, John, and David McGrew. *The use of Galois/counter mode (GCM) in IPsec encapsulating security payload (ESP)*. No. rfc4106. 2005.
- [72] Bellare, Steven M. "Problem Areas for the IP Security Protocols." *USENIX Security Symposium*. 1996.
- [73] Paterson, Kenneth G., and Arnold KL Yau. "Cryptography in theory and practice: The case of encryption in IPsec." *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.
- [74] Degabriele, Jean Paul, and Kenneth G. Paterson. "Attacking the IPsec standards in encryption-only configurations." *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 2007.
- [75] Brelet, Jean-Louis, and Lakshmi Gopalakrishnan. "Using Virtex-II block RAM for high performance read/write CAMs." *Xilinx Application Note XAPP260* (2002).
- [76] Degabriele, Jean Paul, and Kenneth G. Paterson. "On the (in) security of IPsec in MAC-then-encrypt configurations." *Proceedings of the 17th ACM conference on Computer and communications security*. 2010.

- [77] Ivanov, Georgi, Nikolay Nikolov, and Svetla Nikova. "Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties." *Cryptography and Communications* 8.2 (2016): 247-276.
- [78] Avraamova, Ol'ga Dmitrievna, et al. "A compact bit-sliced representation of Kuznyechik S-box." *Математические вопросы криптографии* 12.2 (2021): 21-38.
- [79] Z'aba, Muhammad Reza. *Analysis of linear relationships in block ciphers*. Diss. Queensland University of Technology, 2010.
- [80] Feistel, Horst. "Cryptography and computer privacy." *Scientific american* 228.5 (1973): 15-23.
- [81] Webster, A. F. "SE Tavares On the Design of S-Boxes." *Advances in Cryptology: Crypto*. Vol. 85. 1986.
- [82] Bogdanov, Andrey, Dmitry Khovratovich, and Christian Rechberger. "Biclique cryptanalysis of the full AES." *International conference on the theory and application of cryptology and information security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
- [83] Biryukov, Alex, et al. "Security and performance analysis of ARIA." *Final report, KU Leuven ESAT/SCD-COSIC* 3 (2004): 4.
- [84] Collard, Baudoin, and F-X. Standaert. "A statistical saturation attack against the block cipher PRESENT." *Cryptographers' Track at the RSA Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.
- [85] Luong, Tran Thi, and Truong Minh Phuong. "Generating efficient circulant-like MDS matrices for implementation." *Journal of Science and Technology on Information security* (2024): 58-68.
- [86] Castro, Julio Cesar Hernandez, et al. "The strict avalanche criterion randomness test." *Mathematics and Computers in Simulation* 68.1 (2005): 1-7.
- [87] Zodpe, Harshali, and Ashok Sapkal. "An efficient AES implementation using FPGA with enhanced security features." *Journal of King Saud University-Engineering Sciences* 32.2 (2020): 115-122.
- [88] Abikoye, Oluwakemi Christiana, et al. "Modified advanced encryption standard algorithm for information security." *Symmetry* 11.12 (2019): 1484.
- [89] Standaert, François-Xavier, Siddika Berna Örs, and Bart Preneel. "Power analysis of an FPGA: Implementation of rijndael: Is pipelining a DPA countermeasure?." *International Workshop on Cryptographic Hardware and Embedded Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.
- [90] Sasaki, Yu, et al. "Peigen—a platform for evaluation, implementation, and generation of s-boxes." (2019).

PHỤ LỤC

Phụ lục 1. Phần mềm kiểm tra tính chất mật mã của S-box

```

import numpy as np
import random
from sympy import symbols, Poly, GF
from collections import defaultdict
from itertools import combinations

def read_sboxes(filename, n):
    """Read S-boxes from a text file and validate for n-bit size."""
    sboxes = {}
    current_sbox = None
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'):
                continue
            if not line.startswith('0x'): # S-box name
                current_sbox = line
                sboxes[current_sbox] = []
            else: # S-box values
                if line.endswith(','):
                    line = line[:-1]
                values = [int(x, 16) for x in line.split(',')]
                sboxes[current_sbox].extend(values)

    # Validate each S-box
    for name, sbox in sboxes.items():
        if len(sbox) != 2**n:
            raise ValueError(f"S-box {name} has {len(sbox)} entries, expected {2**n} for n={n}")
        if sorted(sbox) != list(range(2**n)):
            raise ValueError(f"S-box {name} is not a valid permutation for n={n}")
    return sboxes

def hamming_distance(a, b):
    """Compute Hamming distance between two integers."""
    return bin(a ^ b).count('1')

def apply_sbox(input_val, sbox, n):
    """Apply S-box to an input value."""
    if input_val >= 2**n:
        raise ValueError(f"Input {input_val} exceeds {n}-bit range")
    return sbox[input_val]

def walsh_component_function(c, sbox, n):

```

```

        """Compute the maximum absolute Walsh coefficient of the component
        function defined by output mask c."""
        max_walsh = 0
        for a in range(2**n):
            walsh_sum = 0
            for x in range(2**n):
                fx = dot_product(c, apply_sbox(x, sbox, n), n)
                wx = dot_product(a, x, n)
                walsh_sum += (-1) ** (fx ^ wx)
            max_walsh = max(max_walsh, abs(walsh_sum))
        return max_walsh

def compute_nonlinearity_sbox(sbox, n):
    """Compute the nonlinearity of the S-box."""
    min_nl = 2**(n-1)
    for c in range(1, 2**n):
        max_walsh = walsh_component_function(c, sbox, n)
        nl = (2**(n-1)) - (max_walsh // 2)
        min_nl = min(min_nl, nl)
    return min_nl

def compute_differential_uniformity_sbox(sbox, n):
    """Compute differential uniformity of S-box."""
    diff_table = np.zeros((2**n, 2**n), dtype=int)

    for input_diff in range(2**n):
        for x in range(2**n):
            y = x ^ input_diff
            output_diff = apply_sbox(x, sbox, n) ^ apply_sbox(y, sbox, n)
            diff_table[input_diff, output_diff] += 1

    max_diff = np.max(diff_table[1:, :])
    return max_diff

def compute_mdp_sbox(sbox, n):
    """Compute maximum differential probability for S-box."""
    diff_table = np.zeros((2**n, 2**n), dtype=int)

    for input_diff in range(2**n):
        for x in range(2**n):
            y = x ^ input_diff
            output_diff = apply_sbox(x, sbox, n) ^ apply_sbox(y, sbox, n)
            diff_table[input_diff, output_diff] += 1

    max_diff = np.max(diff_table[1:, :])
    mdp = max_diff / (2**n)
    return mdp

def compute_mlp_sbox(sbox, n):
    """Compute maximum linear probability for S-box using Walsh transform."""

```

```

max_walsh = 0
for input_mask in range(1, 2**n):
    for output_mask in range(2**n):
        walsh_sum = 0
        for x in range(2**n):
            sbox_output = apply_sbox(x, sbox, n)
            parity = dot_product(input_mask, x, n) ^
dot_product(output_mask, sbox_output, n)
            walsh_sum += (-1) ** parity
        max_walsh = max(max_walsh, abs(walsh_sum))

mlp = (max_walsh / (2**n)) ** 2
return mlp

def Monomials(n):
    """Generate monomials for degree <= 2 and <= 3 over 2n variables (n input
+ n output bits)."""
    total_vars = 2 * n

    linear_terms = total_vars
    quadratic_terms = (total_vars * (total_vars - 1)) // 2
    num_terms_d2 = 1 + linear_terms + quadratic_terms

    cubic_terms = (total_vars * (total_vars - 1) * (total_vars - 2)) // 6
    num_terms_d3 = num_terms_d2 + cubic_terms

    A = np.zeros((2, num_terms_d3), dtype=np.uint32)

    A[0, 0] = 0
    for i in range(total_vars):
        A[0, i + 1] = 1 << (total_vars - 1 - i)
    idx = total_vars + 1
    for j, k in combinations(range(total_vars), 2):
        A[0, idx] = (1 << (total_vars - 1 - j)) ^ (1 << (total_vars - 1 - k))
        idx += 1

    A[1, :num_terms_d2] = A[0, :num_terms_d2]
    for j, k, l in combinations(range(total_vars), 3):
        A[1, idx] = (1 << (total_vars - 1 - j)) ^ (1 << (total_vars - 1 - k))
^ (1 << (total_vars - 1 - l))
        idx += 1

    return A, num_terms_d2, num_terms_d3

def Equations(sbox, n, A, num_terms_d2, num_terms_d3):
    """Generate equation matrices for degree <= 2 and <= 3 over input-output
pairs."""
    M = np.zeros((2, 2**n, num_terms_d3), dtype=np.uint8)
    total_vars = 2 * n

```

```

for i in range(2**n):
    input_bits = [(i >> j) & 1 for j in range(n-1, -1, -1)]
    output = sbox[i]
    output_bits = [(output >> j) & 1 for j in range(n-1, -1, -1)]
    io_bits = input_bits + output_bits

    M[0, i, 0] = 1
    M[1, i, 0] = 1

    for j in range(1, num_terms_d2):
        active = True
        for k in range(total_vars):
            if (A[0, j] >> (total_vars - 1 - k)) & 1 and io_bits[k] == 0:
                active = False
                break
        M[0, i, j] = 1 if active else 0

    for j in range(1, num_terms_d3):
        active = True
        for k in range(total_vars):
            if (A[1, j] >> (total_vars - 1 - k)) & 1 and io_bits[k] == 0:
                active = False
                break
        M[1, i, j] = 1 if active else 0

return M

def swap_io(N_bien, Mi, Mj):
    """Swap rows Mi and Mj."""
    for i in range(N_bien):
        temp = Mi[i]
        Mi[i] = Mj[i]
        Mj[i] = temp

def gauss_elim_io(N_bien, M, n):
    """Perform Gaussian elimination on matrix M."""
    M = M.copy()
    num_rows = 2**n
    column = 0
    row = 0
    while row < num_rows:
        j = 0
        dk = 0
        for jj in range(row, num_rows):
            if M[jj, column] == 1:
                j = jj
                dk = 1
                break

```

```

    if dk == 1:
        if j != row:
            swap_io(N_bien, M[row], M[j])
        for j in range(row + 1, num_rows):
            if M[j, column] == 1:
                for k in range(N_bien):
                    M[j, k] ^= M[row, k]

        row += 1
        column += 1
    else:
        column += 1
    if column >= N_bien:
        break
return M

def rank(N_bien, M, n):
    """Compute rank of matrix M."""
    M = gauss_elim_io(N_bien, M, n)
    num_rows = 2**n
    i = 0
    dk = 0
    while dk == 0:
        dk = 1
        for j in range(i, N_bien):
            if M[i, j] == 1:
                i += 1
                dk = 0
                break
        if i >= num_rows:
            dk = 1
    return i

def degIO_sbox(sbox, name, n):
    """Compute degIO for the S-box."""
    A, num_terms_d2, num_terms_d3 = Monomials(n)
    M = Equations(sbox, n, A, num_terms_d2, num_terms_d3)

    d0 = rank(num_terms_d2, M[0], n)
    d1 = rank(num_terms_d3, M[1], n)

    if d0 < num_terms_d2:
        deg_io = 2
        ret_val = 0
    elif d1 < num_terms_d3:
        deg_io = 3
        ret_val = 1
    else:
        deg_io = "> 3"
        ret_val = 2

```

```

print(f" degIO: {deg_io}")
return ret_val

def compute_algebraic_degree_sbox(sbox, n):
    """Compute the algebraic degree of the S-box."""
    degrees = []

    for bit in range(n):
        f = [0] * (2**n)
        for x in range(2**n):
            output = apply_sbox(x, sbox, n)
            f[x] = (output >> bit) & 1

        x_vars = symbols('x0:%d' % n)
        terms = []
        for i in range(2**n):
            if f[i] == 1:
                term = 1
                for j in range(n):
                    bit_val = (i >> j) & 1
                    if bit_val == 0:
                        term *= (1 - x_vars[j])
                    else:
                        term *= x_vars[j]
                terms.append(term)

        if terms:
            poly = sum(terms)
            poly = Poly(poly, *x_vars, domain=GF(2))
            degree = poly.total_degree()
        else:
            degree = 0
        degrees.append(degree)

    return max(degrees)

def compute_fixed_points_sbox(sbox, n):
    """Compute the fixed points of the S-box."""
    fixed_points = []
    for x in range(2**n):
        if apply_sbox(x, sbox, n) == x:
            fixed_points.append(x)
    return fixed_points, len(fixed_points)

def hadamard_transform(truth_table, n):
    """Compute the Walsh-Hadamard transform of a Boolean function."""
    vec = np.array([1 - 2 * b for b in truth_table], dtype=int)
    h = vec.copy()
    m = n
    for i in range(m):

```



```

    step = 2 ** i
    for j in range(0, 2**n, 2 * step):
        for k in range(step):
            u = h[j + k]
            v = h[j + k + step]
            h[j + k] = u + v
            h[j + k + step] = u - v
    return h

def component_function(v, sbbox, n):
    """Compute the component function  $f_v(x) = v \cdot S(x)$ ."""
    return [bin(apply_sbox(x, sbbox, n) & v).count("1") % 2 for x in
            range(2**n)]

def check_linear_redundancy(sbbox, n):
    """Check for complete linear redundancy by comparing squared spectra."""
    seen_spectra = defaultdict(list)

    for v in range(1, 2**n):
        comp = component_function(v, sbbox, n)
        spectrum = hadamard_transform(comp, n)
        squared = tuple(sorted([x * x for x in spectrum]))
        seen_spectra[squared].append(v)

    complete_lr = len(seen_spectra) == 1

    print(f"Sbox affinely equivalent (Complete LR)? {complete_lr}")

def compute_sac_sbox(sbbox, n):
    """Compute  $n \times n$  SAC probability matrix for S-box."""
    change_counts = np.zeros((n, n), dtype=np.float64)

    for x in range(2**n):
        for i in range(n):
            y = x ^ (1 << i)
            output_x = apply_sbox(x, sbbox, n)
            output_y = apply_sbox(y, sbbox, n)
            diff = output_x ^ output_y
            for j in range(n):
                change_counts[i, j] += (diff >> j) & 1

    probabilities = change_counts / (2**n)
    mean_prob = np.mean(probabilities)
    return probabilities, mean_prob

def compute_bic_sac_sbox(sbbox, n):
    """Compute  $n \times n$  BIC-SAC correlation matrix for S-box."""
    correlations = np.zeros((n, n), dtype=np.float64)

```

```

for i in range(n):
    for x in range(2**n):
        y = x ^ (1 << i)
        dx = apply_sbox(x, sbox, n) ^ apply_sbox(y, sbox, n)

        for j in range(n):
            for k in range(j + 1, n):
                bit_j = (dx >> j) & 1
                bit_k = (dx >> k) & 1
                if bit_j == bit_k:
                    correlations[j, k] += 1

correlations /= (2**n * n)

for j in range(n):
    for k in range(j + 1, n):
        correlations[k, j] = correlations[j, k]

mean_corr = np.mean([correlations[j, k] for j in range(n) for k in range(j
+ 1, n)])
return correlations, mean_corr

def dot_product(a, x, n):
    """Compute the dot product (XOR of ANDed bits) for two integers."""
    return bin(a & x).count('1') % 2

def compute_nonlinearity_jk(j, k, sbox, n):
    """Compute nonlinearity of  $f(x) = S_j(x) \text{ XOR } S_k(x)$  over all linear input
    masks."""
    max_walsh = 0
    for a in range(1, 2**n):
        wht = 0
        for x in range(2**n):
            s = apply_sbox(x, sbox, n)
            bit_j = (s >> j) & 1
            bit_k = (s >> k) & 1
            fx = bit_j ^ bit_k
            wx = dot_product(a, x, n)
            wht += (-1) ** (fx ^ wx)
        max_walsh = max(max_walsh, abs(wht))
    return (2**(n-1)) - (max_walsh // 2)

def compute_bic_nl_sbox(sbox, n):
    """Compute the BIC-NL matrix for an S-box."""
    nl_matrix = np.zeros((n, n), dtype=np.int32)

    for j in range(n):
        for k in range(j + 1, n):
            nl = compute_nonlinearity_jk(j, k, sbox, n)
            nl_matrix[j, k] = nl

```

```

        nl_matrix[k, j] = nl

    mean_nl = np.sum(nl_matrix) / (n * (n - 1))
    return nl_matrix, mean_nl

def main(n=8, filename='sboxes.txt'):
    """Main function to evaluate n-bit S-boxes."""
    sboxes = read_sboxes(filename, n)

    for sbox_name, sbox in sboxes.items():
        print(f"\n***** Evaluating {sbox_name} S-box (n={n})
        *****")

        nonlinearity = compute_nonlinearity_sbox(sbox, n)
        print("\n=== Nonlinearity ===")
        print(f"Nonlinearity (S-box): {nonlinearity} (Higher is better, max
        {2**(n-1)-2**(n//2-1)} for n={n})")

        diff_uniformity = compute_differential_uniformity_sbox(sbox, n)
        print(f"\n=== Differential Uniformity ===")
        print(f"Differential Uniformity (S-box): {diff_uniformity} (Lower is
        better, optimal ~4 for n=8)")

        mdp = compute_mdp_sbox(sbox, n)
        print(f"\n=== Maximum Differential Probability (MDP) ===")
        print(f"Maximum Differential Probability (MDP) (S-box): {mdp:.6f}
        (Lower is better)")

        mlp = compute_mlp_sbox(sbox, n)
        print(f"\n=== Maximum Linear Probability (MLP) ===")
        print(f"Maximum Linear Probability (MLP) (S-box): {mlp:.6f} (Lower is
        better)")

        print("\n=== DegIO ===")
        degIO_sbox(sbox, sbox_name, n)

        alg_degree = compute_algebraic_degree_sbox(sbox, n)
        print("\n=== Algebraic Degree ===")
        print(f"Algebraic Degree (S-box): {alg_degree} (Higher is better, max
        {n-1} for n={n})")

        fixed_points, num_fixed_points = compute_fixed_points_sbox(sbox, n)
        print("\n=== Fixed Points ===")
        print(f"Fixed Points (S-box): {fixed_points}")
        print(f"Number of Fixed Points: {num_fixed_points} (Lower is better,
        ideally 0)")

        print("\n=== Linear Redundancy ===")
        check_linear_redundancy(sbox, n)

```

```

sac_matrix, sac_mean = compute_sac_sbox(sbox, n)
print("\n=== SAC table ===")
print(sac_matrix)
print(f"\nMean SAC Probability: {sac_mean:.4f} (Ideal: 0.5)")

bic_sac_matrix, bic_sac_mean = compute_bic_sac_sbox(sbox, n)
print("\n=== BIC-SAC table ===")
np.set_printoptions(precision=4, suppress=True)
print(bic_sac_matrix)
print(f"\nMean BIC-SAC Correlation: {bic_sac_mean:.4f} (Ideal: 0.5)")

bic_nl_matrix, bic_nl_mean = compute_bic_nl_sbox(sbox, n)
print("\n=== BIC-NL ===")
print(bic_nl_matrix.astype(int))
print(f"\nMean BIC-NL Nonlinearity: {bic_nl_mean:.2f}")

if __name__ == "__main__":
    main(n=8, filename='sboxes.txt')

```

Phụ lục 2. Tập sboxes.txt sử dụng trong luận án

AES

0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,
0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,
0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,
0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,
0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,
0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,
0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,
0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,
0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,
0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,
0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,
0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,
0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16

S_p

0x01,0x11,0x91,0xE1,0xD1,0xB1,0x71,0x61,0xF1,0x21,0xC1,0x51,0xA1,0x41,0x31,0x81,
0x00,0x10,0xE3,0x92,0xB5,0xD4,0x77,0x66,0x89,0x38,0xAB,0x4A,0xCD,0x5C,0x2F,0xFE,
0x08,0x5F,0x3E,0xB0,0x1C,0xC2,0x83,0xDD,0xE8,0xF6,0x47,0x79,0x95,0x2B,0xAA,0x64,
0x0F,0x48,0xD0,0x29,0xA3,0x1A,0xF2,0xBB,0x65,0xCC,0xE4,0x3D,0x57,0x7E,0x86,0x9F,
0x0C,0x2A,0xF4,0x1F,0x5B,0x90,0xEE,0xC5,0x36,0x6D,0x73,0x88,0xBC,0xA7,0x49,0xD2,
0x0A,0x3C,0x18,0x85,0xE0,0x4D,0x99,0xA4,0xB3,0x5E,0xDA,0xC7,0x72,0xFF,0x6B,0x26,
0x06,0x76,0xCF,0xA8,0x4E,0x59,0x60,0x17,0xDC,0x9B,0x32,0xF5,0x23,0x84,0xED,0xBA,
0x07,0x67,0x2D,0x3B,0xFA,0x8C,0x16,0x70,0x54,0xA2,0x98,0xBE,0xEF,0xD9,0xC3,0x45,
0x0E,0xA9,0x62,0x5A,0x27,0xBF,0x34,0x9C,0xFD,0xD5,0x8E,0xE6,0x1B,0x43,0x78,0xC0,
0x03,0xB2,0x87,0xC4,0x9D,0x6E,0x4B,0xF8,0x7A,0xE9,0x2C,0xAF,0xD6,0x15,0x50,0x33,
0x0D,0xFB,0x56,0xEC,0x3F,0x75,0xB8,0x42,0x1E,0x24,0xC9,0x93,0x80,0x6A,0xD7,0xAD,
0x04,0xE5,0xB9,0x7D,0x82,0xA6,0xCA,0x2E,0x97,0x13,0x6F,0xDB,0x44,0x30,0xFC,0x58,
0x0B,0x8D,0x9A,0x46,0x74,0x28,0xDF,0x53,0xCB,0xB7,0xF0,0x6C,0xAE,0xE2,0x35,0x19,
0x05,0x94,0x7B,0xDE,0xC6,0xF3,0xAC,0x39,0x4F,0x8A,0x55,0x20,0x68,0xBD,0x12,0xE7,
0x02,0xD3,0xA5,0xF7,0x69,0xEB,0x5D,0x8F,0x22,0x40,0xB6,0x14,0x3A,0xC8,0x9E,0x7C,
0x09,0xCE,0x4C,0x63,0xD8,0x37,0x25,0xEA,0xA0,0x7F,0x1D,0x52,0xF9,0x96,0xB4,0x8B

Kuznyechik

0xFC,0xEE,0xDD,0x11,0xCF,0x6E,0x31,0x16,0xFB,0xC4,0xFA,0xDA,0x23,0xC5,0x04,0x4D,
0xE9,0x77,0xF0,0xDB,0x93,0x2E,0x99,0xBA,0x17,0x36,0xF1,0xBB,0x14,0xCD,0x5F,0xC1,
0xF9,0x18,0x65,0x5A,0xE2,0x5C,0xEF,0x21,0x81,0x1C,0x3C,0x42,0x8B,0x01,0x8E,0x4F,
0x05,0x84,0x02,0xAE,0xE3,0x6A,0x8F,0xA0,0x06,0x0B,0xED,0x98,0x7F,0xD4,0xD3,0x1F,
0xEB,0x34,0x2C,0x51,0xEA,0xC8,0x48,0xAB,0xF2,0x2A,0x68,0xA2,0xFD,0x3A,0xCE,0xCC,
0xB5,0x70,0x0E,0x56,0x08,0x0C,0x76,0x12,0xBF,0x72,0x13,0x47,0x9C,0xB7,0x5D,0x87,
0x15,0xA1,0x96,0x29,0x10,0x7B,0x9A,0xC7,0xF3,0x91,0x78,0x6F,0x9D,0x9E,0xB2,0xB1,
0x32,0x75,0x19,0x3D,0xFF,0x35,0x8A,0x7E,0x6D,0x54,0xC6,0x80,0xC3,0xBD,0x0D,0x57,
0xDF,0xF5,0x24,0xA9,0x3E,0xA8,0x43,0xC9,0xD7,0x79,0xD6,0xF6,0x7C,0x22,0xB9,0x03,
0xE0,0x0F,0xEC,0xDE,0x7A,0x94,0xB0,0xBC,0xDC,0xE8,0x28,0x50,0x4E,0x33,0x0A,0x4A,
0xA7,0x97,0x60,0x73,0x1E,0x00,0x62,0x44,0x1A,0xB8,0x38,0x82,0x64,0x9F,0x26,0x41,
0xAD,0x45,0x46,0x92,0x27,0x5E,0x55,0x2F,0x8C,0xA3,0xA5,0x7D,0x69,0xD5,0x95,0x3B,
0x07,0x58,0xB3,0x40,0x86,0xAC,0x1D,0xF7,0x30,0x37,0x6B,0xE4,0x88,0xD9,0xE7,0x89,

0xE1, 0x1B, 0x83, 0x49, 0x4C, 0x3F, 0xF8, 0xFE, 0x8D, 0x53, 0xAA, 0x90, 0xCA, 0xD8, 0x85, 0x61, 0x20, 0x71, 0x67, 0xA4, 0x2D, 0x2B, 0x09, 0x5B, 0xCB, 0x9B, 0x25, 0xD0, 0xBE, 0xE5, 0x6C, 0x52, 0x59, 0xA6, 0x74, 0xD2, 0xE6, 0xF4, 0xB4, 0xC0, 0xD1, 0x66, 0xAF, 0xC2, 0x39, 0x4B, 0x63, 0xB6

Camellia

0x70, 0x82, 0x2c, 0xec, 0xb3, 0x27, 0xc0, 0xe5, 0xe4, 0x85, 0x57, 0x35, 0xea, 0x0c, 0xae, 0x41, 0x23, 0xef, 0x6b, 0x93, 0x45, 0x19, 0xa5, 0x21, 0xed, 0x0e, 0x4f, 0x4e, 0x1d, 0x65, 0x92, 0xbd, 0x86, 0xb8, 0xaf, 0x8f, 0x7c, 0xeb, 0x1f, 0xce, 0x3e, 0x30, 0xdc, 0x5f, 0x5e, 0xc5, 0x0b, 0x1a, 0xa6, 0xe1, 0x39, 0xca, 0xd5, 0x47, 0x5d, 0x3d, 0xd9, 0x01, 0x5a, 0xd6, 0x51, 0x56, 0x6c, 0x4d, 0x8b, 0x0d, 0x9a, 0x66, 0xfb, 0xcc, 0xb0, 0x2d, 0x74, 0x12, 0x2b, 0x20, 0xf0, 0xb1, 0x84, 0x99, 0xdf, 0x4c, 0xcb, 0xc2, 0x34, 0x7e, 0x76, 0x05, 0x6d, 0xb7, 0xa9, 0x31, 0xd1, 0x17, 0x04, 0xd7, 0x14, 0x58, 0x3a, 0x61, 0xde, 0x1b, 0x11, 0x1c, 0x32, 0x0f, 0x9c, 0x16, 0x53, 0x18, 0xf2, 0x22, 0xfe, 0x44, 0xcf, 0xb2, 0xc3, 0xb5, 0x7a, 0x91, 0x24, 0x08, 0xe8, 0xa8, 0x60, 0xfc, 0x69, 0x50, 0xaa, 0xd0, 0xa0, 0x7d, 0xa1, 0x89, 0x62, 0x97, 0x54, 0x5b, 0x1e, 0x95, 0xe0, 0xff, 0x64, 0xd2, 0x10, 0xc4, 0x00, 0x48, 0xa3, 0xf7, 0x75, 0xdb, 0x8a, 0x03, 0xe6, 0xda, 0x09, 0x3f, 0xdd, 0x94, 0x87, 0x5c, 0x83, 0x02, 0xcd, 0x4a, 0x90, 0x33, 0x73, 0x67, 0xf6, 0xf3, 0x9d, 0x7f, 0xbf, 0xe2, 0x52, 0x9b, 0xd8, 0x26, 0xc8, 0x37, 0xc6, 0x3b, 0x81, 0x96, 0x6f, 0x4b, 0x13, 0xbe, 0x63, 0x2e, 0xe9, 0x79, 0xa7, 0x8c, 0x9f, 0x6e, 0xbc, 0x8e, 0x29, 0xf5, 0xf9, 0xb6, 0x2f, 0xfd, 0xb4, 0x59, 0x78, 0x98, 0x06, 0x6a, 0xe7, 0x46, 0x71, 0xba, 0xd4, 0x25, 0xab, 0x42, 0x88, 0xa2, 0x8d, 0xfa, 0x72, 0x07, 0xb9, 0x55, 0xf8, 0xee, 0xac, 0x0a, 0x36, 0x49, 0x2a, 0x68, 0x3c, 0x38, 0xf1, 0xa4, 0x40, 0x28, 0xd3, 0x7b, 0xbb, 0xc9, 0x43, 0xc1, 0x15, 0xe3, 0xad, 0xf4, 0x77, 0xc7, 0x80, 0x9e

SEED_S0

0xA9, 0x85, 0xD6, 0xD3, 0x54, 0x1D, 0xAC, 0x25, 0x5D, 0x43, 0x18, 0x1E, 0x51, 0xFC, 0xCA, 0x63, 0x28, 0x44, 0x20, 0x9D, 0xE0, 0xE2, 0xC8, 0x17, 0xA5, 0x8F, 0x03, 0x7B, 0xBB, 0x13, 0xD2, 0xEE, 0x70, 0x8C, 0x3F, 0xA8, 0x32, 0xDD, 0xF6, 0x74, 0xEC, 0x95, 0x0B, 0x57, 0x5C, 0x5B, 0xBD, 0x01, 0x24, 0x1C, 0x73, 0x98, 0x10, 0xCC, 0xF2, 0xD9, 0x2C, 0xE7, 0x72, 0x83, 0x9B, 0xD1, 0x86, 0xC9, 0x60, 0x50, 0xA3, 0xEB, 0x0D, 0xB6, 0x9E, 0x4F, 0xB7, 0x5A, 0xC6, 0x78, 0xA6, 0x12, 0xAF, 0xD5, 0x61, 0xC3, 0xB4, 0x41, 0x52, 0x7D, 0x8D, 0x08, 0x1F, 0x99, 0x00, 0x19, 0x04, 0x53, 0xF7, 0xE1, 0xFD, 0x76, 0x2F, 0x27, 0xB0, 0x8B, 0x0E, 0xAB, 0xA2, 0x6E, 0x93, 0x4D, 0x69, 0x7C, 0x09, 0x0A, 0xBF, 0xEF, 0xF3, 0xC5, 0x87, 0x14, 0xFE, 0x64, 0xDE, 0x2E, 0x4B, 0x1A, 0x06, 0x21, 0x6B, 0x66, 0x02, 0xF5, 0x92, 0x8A, 0x0C, 0xB3, 0x7E, 0xD0, 0x7A, 0x47, 0x96, 0xE5, 0x26, 0x80, 0xAD, 0xDF, 0xA1, 0x30, 0x37, 0xAE, 0x36, 0x15, 0x22, 0x38, 0xF4, 0xA7, 0x45, 0x4C, 0x81, 0xE9, 0x84, 0x97, 0x35, 0xCB, 0xCE, 0x3C, 0x71, 0x11, 0xC7, 0x89, 0x75, 0xFB, 0xDA, 0xF8, 0x94, 0x59, 0x82, 0xC4, 0xFF, 0x49, 0x39, 0x67, 0xC0, 0xCF, 0xD7, 0xB8, 0x0F, 0x8E, 0x42, 0x23, 0x91, 0x6C, 0xDB, 0xA4, 0x34, 0xF1, 0x48, 0xC2, 0x6F, 0x3D, 0x2D, 0x40, 0xBE, 0x3E, 0xBC, 0xC1, 0xAA, 0xBA, 0x4E, 0x55, 0x3B, 0xDC, 0x68, 0x7F, 0x9C, 0xD8, 0x4A, 0x56, 0x77, 0xA0, 0xED, 0x46, 0xB5, 0x2B, 0x65, 0xFA, 0xE3, 0xB9, 0xB1, 0x9F, 0x5E, 0xF9, 0xE6, 0xB2, 0x31, 0xEA, 0x6D, 0x5F, 0xE4, 0xF0, 0xCD, 0x88, 0x16, 0x3A, 0x58, 0xD4, 0x62, 0x29, 0x07, 0x33, 0xE8, 0x1B, 0x05, 0x79, 0x90, 0x6A, 0x2A, 0x9A

Kalyna_Pi0

0xa8, 0x43, 0x5f, 0x6, 0x6b, 0x75, 0x6c, 0x59, 0x71, 0xdf, 0x87, 0x95, 0x17, 0xf0, 0xd8, 0x9, 0x6d, 0xf3, 0x1d, 0xcb, 0xc9, 0x4d, 0x2c, 0xaf, 0x79, 0xe0, 0x97, 0xfd, 0x6f, 0x4b, 0x45, 0x39, 0x3e, 0xdd, 0xa3, 0x4f, 0xb4, 0xb6, 0x9a, 0xe, 0x1f, 0xbf, 0x15, 0xe1, 0x49, 0xd2, 0x93, 0xc6, 0x92, 0x72, 0x9e, 0x61, 0xd1, 0x63, 0xfa, 0xee, 0xf4, 0x19, 0xd5, 0xad, 0x58, 0xa4, 0xbb, 0xa1, 0xdc, 0xf2, 0x83, 0x37, 0x42, 0xe4, 0x7a, 0x32, 0x9c, 0xcc, 0xab, 0x4a, 0x8f, 0x6e, 0x4, 0x27, 0x2e, 0xe7, 0xe2, 0x5a, 0x96, 0x16, 0x23, 0x2b, 0xc2, 0x65, 0x66, 0xf, 0xbc, 0xa9, 0x47, 0x41, 0x34, 0x48, 0xfc, 0xb7, 0x6a, 0x88, 0xa5, 0x53, 0x86, 0xf9, 0x5b, 0xdb, 0x38, 0x7b, 0xc3, 0x1e, 0x22, 0x33, 0x24, 0x28, 0x36, 0xc7, 0xb2, 0x3b, 0x8e, 0x77, 0xba, 0xf5, 0x14, 0x9f, 0x8, 0x55, 0x9b, 0x4c, 0xfe, 0x60, 0x5c, 0xda, 0x18, 0x46, 0xcd, 0x7d, 0x21, 0xb0, 0x3f, 0x1b, 0x89, 0xff, 0xeb, 0x84, 0x69, 0x3a, 0x9d, 0xd7, 0xd3, 0x70, 0x67, 0x40, 0xb5, 0xde, 0x5d, 0x30, 0x91, 0xb1, 0x78, 0x11, 0x1, 0xe5, 0x0, 0x68, 0x98, 0xa0, 0xc5, 0x2, 0xa6, 0x74, 0x2d, 0xb, 0xa2, 0x76,

0xb3, 0xbe, 0xce, 0xbd, 0xae, 0xe9, 0x8a, 0x31, 0x1c, 0xec, 0xf1, 0x99, 0x94, 0xaa, 0xf6, 0x26, 0x2f, 0xef, 0xe8, 0x8c, 0x35, 0x3, 0xd4, 0x7f, 0xfb, 0x5, 0xc1, 0x5e, 0x90, 0x20, 0x3d, 0x82, 0xf7, 0xea, 0xa, 0xd, 0x7e, 0xf8, 0x50, 0x1a, 0xc4, 0x7, 0x57, 0xb8, 0x3c, 0x62, 0xe3, 0xc8, 0xac, 0x52, 0x64, 0x10, 0xd0, 0xd9, 0x13, 0xc, 0x12, 0x29, 0x51, 0xb9, 0xcf, 0xd6, 0x73, 0x8d, 0x81, 0x54, 0xc0, 0xed, 0x4e, 0x44, 0xa7, 0x2a, 0x85, 0x25, 0xe6, 0xca, 0x7c, 0x8b, 0x56, 0x80

[19]

0x05, 0xf8, 0x92, 0xf4, 0x55, 0xfb, 0x6e, 0xe1, 0xbb, 0x8a, 0xfe, 0x54, 0x1e, 0xd0, 0x6d, 0x7d, 0x82, 0x2e, 0x36, 0x40, 0xee, 0xba, 0xea, 0x67, 0x1d, 0x31, 0x4a, 0x2d, 0x07, 0xbc, 0x44, 0x4f, 0xc1, 0xd4, 0x5f, 0xde, 0xc2, 0xd1, 0x4c, 0xb0, 0x6f, 0x3d, 0x9c, 0xd7, 0x70, 0x9f, 0x3e, 0x30, 0x80, 0x17, 0x26, 0x37, 0xc3, 0x96, 0xd5, 0x74, 0xe0, 0xb8, 0x13, 0x32, 0x14, 0x0c, 0x94, 0x83, 0x01, 0xf0, 0xb6, 0xca, 0x21, 0x64, 0x50, 0x79, 0x4d, 0xd6, 0xce, 0xbd, 0xab, 0xa9, 0x86, 0xdd, 0xcb, 0x1a, 0x0b, 0x56, 0x9b, 0x72, 0x89, 0xf7, 0xf1, 0x7b, 0x1f, 0x9a, 0x00, 0x2a, 0x41, 0x53, 0x02, 0xdb, 0xda, 0x27, 0xf9, 0x97, 0xfa, 0xeb, 0x81, 0x7e, 0xec, 0x3b, 0x10, 0x87, 0xcd, 0x61, 0x25, 0xe3, 0x22, 0xd8, 0x75, 0x9d, 0x12, 0x8c, 0x43, 0x47, 0x95, 0x0d, 0xa8, 0x49, 0x04, 0x16, 0x3f, 0x35, 0x69, 0x1c, 0x6a, 0xa6, 0xff, 0x76, 0x73, 0xcc, 0xe2, 0xb3, 0xd3, 0x2b, 0xb9, 0xa5, 0xe7, 0x63, 0x78, 0x9e, 0x09, 0x66, 0xef, 0x85, 0x03, 0x0f, 0x4b, 0xf5, 0x3a, 0x65, 0x48, 0x39, 0xe8, 0x42, 0x28, 0x0a, 0x84, 0x29, 0xf6, 0xb4, 0xe6, 0xd2, 0x4e, 0xc4, 0xdc, 0x58, 0xc6, 0xb5, 0xa2, 0xe5, 0x62, 0x5e, 0x51, 0xc7, 0xae, 0x7c, 0xd9, 0x19, 0x99, 0x7a, 0xbf, 0xa7, 0x7f, 0xb1, 0x68, 0x20, 0xc5, 0x8e, 0x77, 0xfc, 0xfd, 0xa0, 0x38, 0x15, 0x60, 0xad, 0x5c, 0xc8, 0x91, 0x2c, 0x88, 0xe4, 0x59, 0x5a, 0x8f, 0xc9, 0xa3, 0xf2, 0x46, 0xac, 0x5d, 0xe9, 0xb7, 0x71, 0xc0, 0x23, 0x45, 0x52, 0xdf, 0x8b, 0x1b, 0x24, 0x2f, 0x8d, 0x5b, 0xaf, 0x6c, 0x0e, 0x90, 0x34, 0x3c, 0xed, 0xcf, 0x18, 0xaa, 0xf3, 0x33, 0x93, 0x08, 0xa4, 0x11, 0xbe, 0x6b, 0x98, 0x06, 0xa1, 0x57, 0xb2

[20]

0x71, 0xea, 0x7b, 0xf3, 0x67, 0xf1, 0x10, 0x5c, 0x22, 0xf9, 0x79, 0xb1, 0xef, 0x5f, 0xb5, 0x6b, 0xc5, 0x05, 0xd4, 0xfa, 0xe7, 0xd2, 0x59, 0x66, 0xbc, 0x4e, 0x25, 0xbd, 0x8e, 0x2c, 0x63, 0x44, 0x3f, 0xfe, 0x17, 0x29, 0x2b, 0xbb, 0x7f, 0xcc, 0x2a, 0x3c, 0x7c, 0x76, 0x72, 0xc6, 0x32, 0x1a, 0xdc, 0x5d, 0x31, 0x55, 0x82, 0x0f, 0x18, 0x87, 0x19, 0x03, 0x04, 0x65, 0xf5, 0x39, 0x27, 0x7a, 0x90, 0x15, 0xa8, 0x83, 0x93, 0xe9, 0xc3, 0x24, 0x43, 0xb3, 0x4f, 0x37, 0xb0, 0x75, 0xb9, 0x0c, 0x53, 0x56, 0x00, 0x12, 0x20, 0xee, 0x36, 0xd3, 0xe1, 0xd5, 0xaf, 0xb2, 0xc1, 0xc8, 0xc2, 0xdd, 0x46, 0x01, 0xa5, 0xf7, 0x51, 0xd8, 0x33, 0x1c, 0x58, 0xf6, 0xfd, 0xfb, 0x42, 0xaa, 0x9f, 0xa4, 0x52, 0x35, 0x40, 0x9d, 0x07, 0x9e, 0xa2, 0x7e, 0xae, 0x2f, 0xc7, 0x30, 0x02, 0xff, 0x77, 0x47, 0x08, 0x88, 0x13, 0xec, 0xdb, 0x1f, 0x48, 0x1b, 0x4c, 0x3d, 0xeb, 0xba, 0x68, 0xda, 0x92, 0x73, 0x60, 0x14, 0xd9, 0xce, 0x21, 0xa1, 0x06, 0x84, 0x49, 0xed, 0xa6, 0x0a, 0xcf, 0xcb, 0x91, 0xd7, 0x64, 0x23, 0xa3, 0x81, 0xd0, 0x09, 0x28, 0xca, 0x45, 0x57, 0xac, 0x61, 0x16, 0x1e, 0x6c, 0xf2, 0x7d, 0xc4, 0x3b, 0xf8, 0x9c, 0x5e, 0xc9, 0xb4, 0xe8, 0x4b, 0x6e, 0xe5, 0x78, 0xe3, 0xad, 0x80, 0xa7, 0xe2, 0x38, 0xa9, 0x8a, 0x2d, 0x2e, 0x4d, 0xe4, 0xde, 0x6a, 0x9b, 0xd1, 0xbe, 0x95, 0x85, 0x62, 0xab, 0x3e, 0x69, 0xc0, 0x11, 0x6f, 0x89, 0x70, 0x3a, 0x5b, 0xb6, 0x0d, 0x54, 0x9a, 0x8f, 0x74, 0xe6, 0x86, 0xfc, 0xf0, 0xd6, 0x8d, 0x0e, 0x97, 0x8b, 0x1d, 0xf4, 0xcd, 0x5a, 0xa0, 0xdf, 0x8c, 0x34, 0x94, 0x98, 0xbf, 0x6d, 0x41, 0xe0, 0x50, 0x96, 0xb8, 0x99, 0x26, 0x4a, 0xb7, 0x0b

[21]

0x91, 0x81, 0x46, 0x56, 0xa5, 0x12, 0x5b, 0x54, 0x1d, 0x03, 0x58, 0x84, 0x2f, 0x14, 0x11, 0x31, 0xd9, 0x9e, 0x6b, 0xdb, 0x93, 0x0a, 0x27, 0x82, 0x3c, 0xb3, 0xca, 0x22, 0xbe, 0xb9, 0x80, 0x3f, 0x4f, 0xd5, 0x29, 0x72, 0x61, 0x4a, 0xa6, 0xa9, 0xe6, 0x76, 0xc7, 0x87, 0x8e, 0x1f, 0x4b, 0x7c, 0x49, 0x5d, 0x06, 0x53, 0x97, 0x63, 0x78, 0xcb, 0x13, 0x7a, 0x40, 0xf8, 0x6a, 0x04, 0xb7, 0xfa, 0xbb, 0x94, 0x74, 0x8f, 0xa4, 0x7d, 0x51, 0x23, 0x8a, 0xa1, 0x09, 0x6f, 0xdf, 0x9f, 0xb0, 0x6c, 0x9d, 0x02, 0x32, 0x2d, 0x9b, 0xb2, 0x71, 0x07, 0x0e, 0xb5, 0xc2, 0xae, 0xf4, 0xef, 0x36, 0x00, 0xb4, 0xe5, 0x4c, 0x95, 0x48, 0x8d, 0x75, 0xcf, 0x33, 0x1b, 0xe7, 0x9a, 0x10, 0x6e, 0x7e, 0xec, 0x1c, 0xf2, 0xc5, 0x79, 0x19, 0x68, 0xac, 0xf0, 0x25, 0xdd, 0x0d, 0x18, 0x5e, 0x67, 0xc6, 0xcd, 0xf3, 0x21, 0xaf, 0xab, 0x50, 0xc1, 0x99, 0xf5, 0x98, 0x0b, 0x05, 0x35, 0xd0, 0x89, 0x5a, 0xea,

0x26,0x2a,0xb1,0x43,0x55,0x1a,0x65,0xc9,0xd7,0xba,0x7f,0x4e,0x96,0xd8,0x60,0xda,
 0xfc,0x41,0x3e,0x2b,0xb8,0xdc,0x70,0x34,0xe1,0x42,0xe0,0xb6,0x24,0x73,0xed,0x62,
 0x4d,0xbf,0x39,0x7b,0x5f,0x08,0xa7,0x37,0xe3,0x38,0x83,0x1e,0xde,0xc8,0x88,0xeb,
 0x45,0xce,0x17,0xff,0x66,0x16,0xc4,0xfd,0x5c,0xd3,0xaa,0xfe,0x85,0x28,0xf9,0x0f,
 0xa0,0xfb,0x44,0xd6,0xc0,0x6d,0x57,0x30,0xbd,0xc3,0xbc,0xe4,0x0c,0x8b,0x69,0xa3,
 0x3d,0x20,0x3a,0xf6,0xf1,0x2e,0xd1,0x86,0x01,0x92,0xcc,0xa8,0xe2,0xad,0xee,0x59,
 0x15,0x90,0x77,0x52,0xe9,0x9c,0x8c,0xa2,0x2c,0x3b,0xe8,0xd4,0x64,0xf7,0xd2,0x47

[22]

0xAC,0x76,0xA6,0x5F,0xCE,0x02,0x55,0x00,0xFA,0xC8,0xFB,0x77,0x37,0xF6,0x9D,0x73,
 0x87,0x5C,0xF9,0xA3,0x60,0x4E,0x41,0x65,0x61,0x88,0xE6,0xCA,0x53,0x1A,0x24,0x82,
 0x39,0x49,0xD4,0x5B,0x06,0x7D,0x4C,0x7B,0xAD,0xCF,0xBA,0xB0,0x5A,0x71,0x2F,0xC2,
 0x4A,0xAB,0xD9,0xFC,0xEE,0x56,0x9F,0xAF,0x34,0xEB,0xF7,0x93,0xE8,0x8E,0xBB,0x0D,
 0x66,0x89,0x5E,0x45,0x90,0x80,0x30,0x4D,0x9E,0x2A,0x23,0x6E,0xDC,0x46,0x20,0xA0,
 0x4B,0x0A,0x1D,0x14,0xA7,0x9C,0xC5,0xE5,0xD7,0x52,0x42,0x81,0x6D,0x91,0x9B,0x05,
 0xDF,0xBF,0x48,0xB5,0x16,0x44,0x84,0x75,0xEA,0x1E,0xB6,0x08,0x35,0x03,0x2D,0xE3,
 0xE0,0x33,0x68,0x70,0x01,0x86,0x54,0xE7,0xE9,0xEC,0xDA,0x72,0x40,0xE2,0x1B,0x74,
 0xAE,0xE4,0x3E,0xC1,0xB2,0x83,0x3F,0x69,0xD5,0x64,0xDD,0xD6,0x85,0x19,0x3B,0xF1,
 0xD2,0x22,0xEF,0x26,0x0C,0xA2,0xAA,0x59,0x94,0x6A,0xBE,0x17,0x8D,0x67,0xCD,0x0F,
 0x38,0x51,0xFF,0x18,0x13,0xE1,0xF0,0xCC,0x29,0xA1,0xB4,0x79,0x7F,0x28,0x6F,0xF4,
 0x11,0x2C,0xD3,0xFE,0xDB,0x5D,0x3A,0x36,0x2B,0xC9,0x32,0x3D,0x50,0x8A,0x1F,0xB3,
 0x15,0x21,0x25,0xF5,0xB9,0xB1,0x47,0x7E,0x96,0xF3,0xD8,0xC7,0xB8,0x3C,0x27,0xF2,
 0x8F,0xA8,0x92,0x2E,0xC6,0x63,0x99,0x4F,0x07,0x78,0x1C,0x6B,0x0B,0x57,0x6C,0xF8,
 0xED,0xCB,0x04,0x95,0xA9,0xA4,0xA5,0xFD,0x7A,0x12,0xDE,0x43,0xD0,0xB7,0x09,0x58,
 0x0E,0x98,0x8C,0x31,0x97,0xC4,0xC3,0x7C,0xBD,0xD1,0x8B,0x9A,0x10,0x62,0xC0,0xBC

[23]

0x9f,0x73,0x33,0xcf,0xdf,0x1f,0x53,0x03,0x87,0x0c,0xa5,0x80,0x81,0x19,0x77,0xd4,
 0x63,0x86,0x92,0xb7,0x29,0xb4,0xe0,0xfd,0xd5,0x46,0xc3,0xe5,0xf1,0x89,0xb2,0xf8,
 0x44,0xcd,0xeb,0x6a,0x67,0xee,0x08,0x9c,0x4d,0xc7,0x6c,0x09,0x75,0x99,0x17,0x2a,
 0x85,0xca,0xd8,0xd9,0x04,0x37,0xea,0x59,0xd6,0x45,0xd2,0x94,0x1b,0x40,0xbd,0x4a,
 0x0b,0x7a,0xe3,0xa3,0xaa,0xae,0x5d,0xb1,0x98,0x35,0xdc,0x3c,0x8a,0x21,0x9d,0xbf,
 0x5e,0xb6,0xcb,0xd1,0xf4,0x4c,0x4b,0x1a,0xab,0x27,0x65,0x23,0x7e,0x1e,0xb9,0x51,
 0x14,0xad,0xce,0x8f,0x3e,0x12,0x70,0xe9,0xda,0xfa,0x47,0xa7,0x84,0xc5,0x6d,0x5c,
 0x3b,0x90,0x68,0x48,0xed,0x7f,0xe8,0x9a,0xb5,0xf6,0xbe,0x8e,0x49,0x0f,0xec,0xb3,
 0xa4,0xc1,0x4f,0x95,0x8c,0xac,0x38,0xd3,0xdd,0xe6,0xc2,0x62,0x9b,0xef,0xf2,0x3a,
 0x0e,0x2c,0xc0,0x83,0x57,0xcc,0xdb,0x30,0x7c,0x82,0x18,0x5b,0x36,0x88,0x91,0x05,
 0x43,0x79,0x3f,0xe1,0x72,0xde,0x55,0xf7,0x6b,0x02,0x15,0x07,0xc9,0xa2,0x78,0x93,
 0x58,0x32,0x50,0x64,0xf5,0x7d,0x61,0x6f,0x69,0x60,0xfb,0x39,0x1c,0x7b,0x96,0xa0,
 0xa9,0x66,0x2f,0x2d,0xc6,0xc4,0x20,0xb0,0x13,0x56,0xe2,0xbc,0x8d,0x9e,0xe4,0xd7,
 0x1d,0x54,0x00,0x5a,0x06,0xb8,0xff,0x3d,0x34,0x2b,0x42,0x4e,0x0d,0xf3,0xfe,0xf0,
 0x01,0xa6,0x71,0x76,0x6e,0xa1,0xbb,0x0a,0x25,0x28,0x41,0xaf,0x10,0x26,0x97,0xe7,
 0x24,0x11,0x22,0x52,0xd0,0xf9,0x2e,0x31,0x5f,0xfc,0x8b,0x74,0xc8,0x16,0xa8,0xba

[24]

0xE1,0x3D,0xF5,0xF9,0xB3,0xE9,0xED,0x47,0x30,0x40,0xE5,0xE8,0xBF,0xD7,0xEA,0xB5,
 0x8B,0x82,0x4B,0x7D,0xBB,0x59,0xC5,0x33,0x6E,0x17,0xA2,0xEE,0x1E,0x26,0xB1,0x03,
 0xF6,0x7F,0xD2,0xA4,0xB4,0xFC,0xF7,0x0F,0x34,0x66,0x67,0x73,0x71,0x1B,0x70,0x54,
 0x04,0xC7,0xE0,0xC3,0x18,0x96,0x44,0x9A,0xC4,0x90,0x02,0xA3,0xEB,0xE4,0xB2,0x75,
 0x48,0xC2,0x2C,0x98,0xD8,0xAD,0x99,0x22,0x91,0xF8,0x97,0xF2,0x68,0xE3,0xEC,0x06,
 0xD1,0x53,0x00,0x6F,0x20,0x3F,0x72,0xD9,0xA9,0xCB,0xBE,0x78,0x89,0x0D,0x19,0xCF,
 0x13,0xEF,0xAA,0xFB,0xC1,0x4D,0xF0,0x46,0x45,0x7B,0x80,0xFD,0x11,0x3C,0xDE,0x2A,

0x51,0xE2,0x01,0xCE,0x92,0x5E,0x38,0x77,0x3E,0xB6,0x9B,0x60,0x10,0xFF,0xF3,0x93,
 0x4F,0x0C,0xD0,0x2F,0xDD,0xD6,0x05,0xD4,0x07,0xE6,0xBD,0x7C,0x25,0x5D,0x58,0xF1,
 0x21,0x42,0xCD,0x1F,0xA0,0xA8,0x12,0x0A,0x85,0xAF,0x3A,0x14,0x9F,0x9D,0xC8,0xDB,
 0x23,0xB0,0xB8,0x88,0x49,0x84,0x24,0x1D,0x83,0xD3,0x2E,0xA1,0x52,0x56,0x27,0x79,
 0xE7,0x0B,0xF4,0x6D,0x4E,0x57,0x8D,0x6A,0x2D,0x95,0x37,0xAB,0x65,0xB9,0xAE,0x08,
 0xBA,0x39,0x64,0xAC,0x1C,0xA6,0x36,0x87,0x2B,0x5F,0x35,0xDC,0xC9,0x7E,0xCA,0x8A,
 0x31,0xBC,0x76,0xA5,0x09,0xC0,0xB7,0x8C,0x61,0x74,0xD5,0x7A,0x86,0x43,0x5C,0x9E,
 0x63,0x3B,0x1A,0x50,0x69,0x5B,0x8E,0x16,0xDA,0x9C,0xC6,0x6B,0x8F,0x55,0x28,0xDF,
 0x0E,0x62,0x4A,0x4C,0xFE,0xA7,0x81,0x29,0x41,0x5A,0x6C,0xCC,0x32,0x15,0xFA,0x94

[25]

0x3E,0xEE,0xC5,0xBA,0x7B,0xA0,0x95,0x07,0xAC,0x22,0x56,0x6A,0x55,0x64,0xF6,0xC6,
 0x33,0x7D,0x2C,0xFC,0x35,0x31,0x28,0x84,0xC9,0x17,0x16,0xAD,0x30,0xAE,0x97,0x88,
 0xC7,0xBD,0x6E,0x00,0xA6,0xC4,0x62,0x9B,0x8E,0x1B,0x79,0x0B,0x0C,0x69,0xDA,0xE8,
 0x20,0xA5,0x5C,0xE2,0x54,0x44,0x36,0xB6,0x77,0x5B,0xC2,0x47,0xD8,0xCF,0x18,0x5E,
 0x5A,0xD0,0xE5,0x93,0xE1,0x10,0x45,0x04,0xF5,0x59,0x9E,0x8B,0xB9,0xD9,0x9A,0x21,
 0x1D,0x74,0x15,0xB0,0x25,0xD4,0x57,0x29,0xC1,0x73,0x72,0xD1,0x12,0x4A,0xE4,0x7E,
 0x23,0xA4,0xEC,0xE7,0xB1,0xD5,0x1E,0x40,0x8A,0x99,0x75,0xFE,0x65,0x14,0x1C,0xF2,
 0xAF,0xBC,0xB3,0xF4,0x85,0x41,0xB2,0x09,0x0E,0x90,0x7F,0x52,0xFD,0x51,0x1A,0xDE,
 0x87,0x8D,0x37,0xA9,0x89,0xA2,0xE3,0x68,0xF3,0xF9,0x43,0x78,0xF8,0x8F,0x27,0xAA,
 0xAB,0xA3,0x60,0x2B,0x82,0x7C,0x19,0x92,0x32,0x2D,0xEA,0x13,0xE9,0xEB,0xD6,0x4D,
 0xB7,0x2E,0xCE,0x0D,0x02,0x6C,0x05,0x94,0x50,0x01,0x9C,0x61,0x67,0x98,0x34,0x70,
 0x24,0xD7,0x2A,0x76,0x53,0xD3,0x96,0x49,0x39,0xB8,0xBE,0x06,0x9D,0x0A,0x80,0xC3,
 0xDB,0xFA,0xDC,0x63,0xFB,0xF0,0x26,0xE0,0x81,0xCD,0xA8,0xB5,0xBB,0x46,0x38,0x4F,
 0xC0,0x08,0xC8,0x3C,0x4C,0x6D,0xCB,0x6B,0x86,0xF1,0xA7,0x83,0x3D,0x8C,0xED,0xF7,
 0x42,0xA1,0x58,0x48,0xB4,0xCC,0xDF,0x4B,0xD2,0xCA,0x3A,0x66,0x4E,0xE6,0xFF,0x11,
 0x71,0x7A,0x91,0x2F,0xEF,0xBF,0x6F,0x9F,0x0F,0x5F,0x1F,0xDD,0x03,0x3F,0x3B,0x5D

Phụ lục 3. Cấu tạo của các bảng IPSec

Trường (32 bit)	Chức năng	Yêu cầu	Sử dụng trong bảng
src_ip	Địa chỉ nguồn của gói IP	Bắt buộc	SPD output, SPD input và SPD Secure
src_ip_mask	Mặt nạ của địa chỉ nguồn		
dst_ip	Địa chỉ đích của gói IP		
dst_ip_mask	Mặt nạ của địa chỉ đích		
src_port	Cổng nguồn (UDP, TCP)		
src_port_mask	Mặt nạ của cổng nguồn		
dst_port	Cổng đích (UDP, TCP)		
dst_port_mask	Mặt nạ của cổng đích		
protocol	Trường giao thức của gói IP		
protocol_mask	Mặt nạ của trường giao thức		
sa_id	Địa chỉ liên kết của SAD output	Bắt buộc	SPD Secure
src_tunnel	Địa chỉ nguồn của gói tin bảo mật		
dst_tunnel	Địa chỉ đích của gói tin bảo mật		
spi	Giá trị SPI	Bắt buộc	SAD output và SAD input
key_cipher_0	Khóa mã hóa		
key_cipher_1			
key_cipher_2			
key_cipher_3			
key_cipher_4			
key_cipher_5			
key_cipher_6			
key_cipher_7			
iv_0	IV		
iv_1			
iv_2			
iv_3			
key_hmac_0	Khóa xác thực	Tùy chọn	
key_hmac_1			
key_hmac_2			

key_hmac_3			
key_hmac_4			
key_hmac_5			
key_hmac_6			
key_hmac_7			
cipher_suite	Bộ thuật toán		
esn_hi	Giá trị ESN gói tin đi	Bắt buộc	SAD output
esn_low			
udp_encap	Đóng gói UDP ESP	Tùy chọn	
udp_src_port	Cổng nguồn UDP ESP		
udp_dst_port	Cổng đích UDP ESP		
seq_cnt_hi	Giá trị ESN gói tin đến	Bắt buộc	SAD input
seq_cnt_low			
bitmap_hi	Giá trị bitmap của cửa sổ trượt		
bitmap_low			
top_window_hi	Giá trị cao của cửa sổ trượt		
top_window_low			

Phụ lục 4. Module S-box và MDS lựa chọn cài đặt theo phương pháp biểu diễn logic

```

module S (clk, in, out);
    input clk;
    input [7:0] in;
    output reg [7:0] out;
    wire [3:0] xi,yi;
    wire [3:0] xo,yo;

    function [3 : 0] mul(input [3 : 0] x, input [3 : 0] y);
        reg x0,x1,x2,x3;
        reg y0,y1,y2,y3;
        reg z0,z1,z2,z3;
        reg P1,P2;
        begin
            {x3,x2,x1,x0} = x;
            {y3,y2,y1,y0} = y;
            P1 = x0 ^ x3;
            P2 = x2 ^ x3;
            z0 = x1 & y3 ^ x2 & y2 ^ x3 & y1 ^ x0 & y0;
            z1 = (x2 ^ x1) & y3 ^ P2 & y2 ^ P1 & y1 ^ x1 & y0;
            z2 = P2 & y3 ^ P1 & y2 ^ x1 & y1 ^ x2 & y0;
            z3 = P1 & y3 ^ x1 & y2 ^ x2 & y1 ^ x3 & y0;
            mul = {z3,z2,z1,z0};
        end
    endfunction

    function [3 : 0] mux(input [3 : 0] a, input [3 : 0] b, input [3 : 0]
y);
        reg a0,a1,a2,a3;
        reg b0,b1,b2,b3;
        reg y0,y1,y2,y3;
        reg x0,x1,x2,x3;
        begin
            {a3,a2,a1,a0} = a;
            {b3,b2,b1,b0} = b;
            {y3,y2,y1,y0} = y;
            x0 = (~(y0|y1|y2|y3)) & (a0^b0) ^ b0;
            x1 = (~(y0|y1|y2|y3)) & (a1^b1) ^ b1;
            x2 = (~(y0|y1|y2|y3)) & (a2^b2) ^ b2;
            x3 = (~(y0|y1|y2|y3)) & (a3^b3) ^ b3;
            mux = {x3,x2,x1,x0};
        end
    endfunction

    function [3 : 0] pi_1(input [3 : 0] x);
        reg x0,x1,x2,x3;
        reg y0,y1,y2,y3;
        reg P1,P2,P3,P4,P5;
        begin
            {x3,x2,x1,x0} = x;

```

```

        P1 = x0|x1;
        P2 = x1|x2;
        P3 = x0^x2;
        P4 = x0|x3;
        P5 = x2&P4;
        y0 = P3 ^ P5 ^ x0 & x1 & (x2^x3);
        y1 = x1 ^ x0 & P3 ^ (~x3) & (P1^P2);
        y2 = x0 ^ P1 ^ P5 ^ x3 & (x2^P2);
        y3 = x1 ^ P3 ^ P4 ^ x3 & P2;
        pi_1 = {y3,y2,y1,y0};
    end
endfunction

function [3 : 0] pi_2(input [3 : 0] x);
    reg x0,x1,x2,x3;
    reg y0,y1,y2,y3;
    reg P1,P2,P3,P4;
    begin
        {x3,x2,x1,x0} = x;
        P1 = x0|x2;
        P2 = x2&x3;
        P3 = x0^x1;
        P4 = x0|x3;
        y0 = x0 & x3 ^ x1 & P1 ^ (~P2) & P3;
        y1 = x0 & x1 ^ x0 & P2 ^ P1 ^ P4;
        y2 = (x0^x2) & (x1|x3) ^ P1 ^ P3;
        y3 = x3&(x1|x2) ^ P4 ^ x2 ^ P3;
        pi_2 = {y3,y2,y1,y0};
    end
endfunction

function [3 : 0] pi_inv(input [3 : 0] x);
    reg x0,x1,x2,x3;
    reg y0,y1,y2,y3;
    reg P1,P2;
    begin
        {x3,x2,x1,x0} = x;
        P1 = x0^x1;
        P2 = x1&x2;
        y0 = x3 ^ (x2|P1) ^ P2 & (x0^x3);
        y1 = x3 ^ x1 & (x0|x3) ^ x2 & P1;
        y2 = x0 & (x2|x3) ^ x0 & x1 ^ x2 ^ x3;
        y3 = (x3|(x1^x2)) ^ x3 & (x0^P2);
        pi_inv = {y3,y2,y1,y0};
    end
endfunction

assign {xi,yi} = in;
assign xo = mux(pi_inv(yi), pi_2(mul(yi, mux(pi_inv(xi), mul(pi_1(xi),
yi), yi))), mux(pi_inv(xi), mul(pi_1(xi), yi), yi));
assign yo = mux(pi_inv(xi), mul(pi_1(xi), yi), yi);

always @(posedge clk) begin
    out <= {xo,yo} ^ 1'h1;
end;

```

endmodule

module MixColumns (in, out);

function [7 : 0] x2(input [7 : 0] op);

begin

x2 = {op[6 : 0], 1'b0} ^ (8'h1b & {8{op[7]}});

end

endfunction

function [7 : 0] x2_inverse(input [7 : 0] op);

begin

x2_inverse = {1'b0, op[7 : 1]} ^ (8'h8d & {8{op[0]}});

end

endfunction

function [7 : 0] x3(input [7 : 0] op);

begin

x3 = x2(op) ^ op;

end

endfunction

function [7 : 0] x8c(input [7 : 0] op);

begin

x8c = x2_inverse(op) ^ op;

end

endfunction

assign out[127:120] = x2(in[127:120]) ^ in[119:112] ^ in[111:104] ^ in[103:96];

assign out[119:112] = in[127:120] ^ in[119:112] ^ x2(in[111:104]) ^ x3(in[103:96]);

assign out[111:104] = in[127:120] ^ x3(in[119:112]) ^ in[111:104] ^ x2(in[103:96]);

assign out[103:96] = in[127:120] ^ x2(in[119:112]) ^ x8c(in[111:104]) ^ in[103:96];

assign out[95:88] = x2(in[95:88]) ^ in[87:80] ^ in[79:72] ^ in[71:64];

assign out[87:80] = in[95:88] ^ in[87:80] ^ x2(in[79:72]) ^ x3(in[71:64]);

assign out[79:72] = in[95:88] ^ x3(in[87:80]) ^ in[79:72] ^ x2(in[71:64]);

assign out[71:64] = in[95:88] ^ x2(in[87:80]) ^ x8c(in[79:72]) ^ in[71:64];

assign out[63:56] = x2(in[63:56]) ^ in[55:48] ^ in[47:40] ^ in[39:32];

assign out[55:48] = in[63:56] ^ in[55:48] ^ x2(in[47:40]) ^ x3(in[39:32]);

assign out[47:40] = in[63:56] ^ x3(in[55:48]) ^ in[47:40] ^ x2(in[39:32]);

assign out[39:32] = in[63:56] ^ x2(in[55:48]) ^ x8c(in[47:40]) ^ in[39:32];

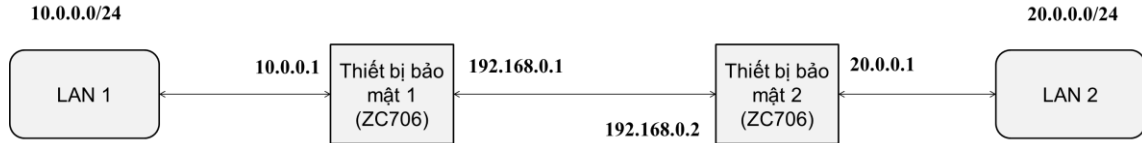
```

        assign out[31:24] = x2(in[31:24]) ^ in[23:16] ^ in[15:8] ^
in[7:0];
        assign out[23:16] = in[31:24] ^ in[23:16] ^ x2(in[15:8]) ^
x3(in[7:0]);
        assign out[15:8] = in[31:24] ^ x3(in[23:16]) ^ in[15:8] ^
x2(in[7:0]);
        assign out[7:0] = in[31:24] ^ x2(in[23:16]) ^ x8c(in[15:8]) ^
in[7:0];
endmodule

```

Phụ lục 5. Đánh giá thực nghiệm trên Kit ZC706

Mô hình thực nghiệm trong luận án được thể hiện ở hình dưới đây:



Trong đó 2 thiết bị bảo mật tương ứng với 2 Kit ZC706 cung cấp giải pháp IPsec ở chế độ tunnel cho hai mạng LAN1, LAN2. Như đã trình bày ở phần 4.4, việc sử dụng đồng bộ PetaLinux 2015.4 và StrongSwan 5.8.4 nhằm đảm bảo tính tương thích tối đa giữa hệ điều hành nhúng với các IP Core của Xilinx trên dòng chip Zynq-7000 (Kit ZC706).

Trong hệ điều hành PetaLinux 2015.4, khai báo cấu hình device tree và core IPsec ở dạng UIO device:

petalinux/subsystems/linux/configs/device-tree/system-top.dts

```

&axi_ethernet_0 {
    local-mac-address = [00 0a 35 00 01 22];
    phy-handle = <&phy0>;
    xlnx,has-mdio = <0x1>;
    phy-mode = "rgmii-rxid";
    mdio {
        #address-cells = <1>;
        #size-cells = <0>;
        phy0: phy@0 {
            compatible = "marvell,88e1510";
            device_type = "ethernet-phy";
            reg = <0>;
        };
    };
};

&axi_ethernet_1 {
    local-mac-address = [00 0a 35 00 01 23];
    phy-handle = <&phy1>;
    xlnx,has-mdio = <0x1>;
    phy-mode = "rgmii-rxid";
    mdio {
        #address-cells = <1>;
        #size-cells = <0>;
        phy1: phy@0 {
            compatible = "marvell,88e1510";
            device_type = "ethernet-phy";
            reg = <0>;
        };
    };
};
  
```



```

    };
};

};

petalinux/subsystems/linux/configs/device-tree/ pl.dtsi
/ {
    amba_pl: amba_pl {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "simple-bus";
        ranges ;
        axi_ethernet_0: ethernet@41000000 {
            axistream-connected = <&axi_ethernet_0_dma>;
            axistream-control-connected = <&axi_ethernet_0_dma>;
            clock-frequency = <100000000>;
            clock-names = "ref_clk";
            clocks = <&clkc 0>;
            compatible = "xlnx,axi-ethernet-1.00.a";
            device_type = "network";
            interrupt-parent = <&intc>;
            interrupts = <0 33 1 0 34 4>;
            phy-mode = "RGMII";
            reg = <0x41000000 0x40000>;
            xlnx = <0x0>;
            xlnx,axiliteclkrate = <0x0>;
            xlnx,axisclkrate = <0x0>;
            xlnx,phy-type = <0x3>;
            xlnx,phyaddr = <0x1>;
            xlnx,rable = <0x0>;
            xlnx,rxcsun = <0x0>;
            xlnx,rxmem = <0x4000>;
            xlnx,txcsun = <0x0>;
            axi_ethernet_0_mdio: mdio {
                #address-cells = <1>;
                #size-cells = <0>;
            };
        };
        axi_ethernet_0_dma: dma@40400000 {
            #dma-cells = <1>;
            axistream-connected = <&axi_ethernet_0>;
            axistream-control-connected = <&axi_ethernet_0>;
            compatible = "xlnx,axi-dma-1.00.a";
            interrupt-parent = <&intc>;
            interrupts = <0 29 4 0 30 4>;
            reg = <0x40400000 0x10000>;
            xlnx,include-sg ;
            xlnx,sg-include-stscntrl-strm ;
            dma-channel@40400000 {
                compatible = "xlnx,axi-dma-mm2s-channel";
                interrupts = <0 29 4>;
                xlnx,datawidth = <0x20>;
                xlnx,device-id = <0x0>;
                xlnx,include-dre ;
            };
            dma-channel@40400030 {

```

```

compatible = "xlnx,axi-dma-s2mm-channel";
interrupts = <0 30 4>;
xlnx,datawidth = <0x20>;
xlnx,device-id = <0x0>;
xlnx,include-dre ;
};

axi_ethernet_1: ethernet@41040000 {
    axistream-connected = <&axi_ethernet_1_dma>;
    axistream-control-connected = <&axi_ethernet_1_dma>;
    clock-frequency = <100000000>;
    clock-names = "ref_clk";
    clocks = <&clkc 0>;
    compatible = "xlnx,axi-ethernet-1.00.a";
    device_type = "network";
    interrupt-parent = <&intc>;
    interrupts = <0 35 1 0 36 4>;
    phy-mode = "RGMII";
    reg = <0x41040000 0x40000>;
    xlnx = <0x0>;
    xlnx,axiliteclkrate = <0x0>;
    xlnx,axisclkrate = <0x0>;
    xlnx,phy-type = <0x3>;
    xlnx,phyaddr = <0x1>;
    xlnx,rable = <0x0>;
    xlnx,rxcsun = <0x0>;
    xlnx,rxmem = <0x4000>;
    xlnx,txcsun = <0x0>;
    axi_ethernet_1_mdio: mdio {
        #address-cells = <1>;
        #size-cells = <0>;
    };
};

axi_ethernet_1_dma: dma@40410000 {
    #dma-cells = <1>;
    axistream-connected = <&axi_ethernet_1>;
    axistream-control-connected = <&axi_ethernet_1>;
    compatible = "xlnx,axi-dma-1.00.a";
    interrupt-parent = <&intc>;
    interrupts = <0 31 4 0 32 4>;
    reg = <0x40410000 0x10000>;
    xlnx,include-sg ;
    xlnx,sg-include-stscntrl-strm ;
    dma-channel@40410000 {
        compatible = "xlnx,axi-dma-mm2s-channel";
        interrupts = <0 31 4>;
        xlnx,datawidth = <0x20>;
        xlnx,device-id = <0x1>;
        xlnx,include-dre ;
    };
    dma-channel@40410030 {
        compatible = "xlnx,axi-dma-s2mm-channel";
        interrupts = <0 32 4>;
        xlnx,datawidth = <0x20>;
    };
};

```

```

        xlnx,device-id = <0x1>;
        xlnx,include-dre ;

    };

    IP_port_lookup_0: IP_port_lookup@600000000 {
        compatible = "generic-uio";
        reg = <0x600000000 0x20000>;
    };

    ipsec_0: ipsec@500000000 {
        compatible = "generic-uio";
        reg = <0x500000000 0x10000>;
    };

};
};

```

Thêm địa chỉ các thanh ghi vào *kernel_netlink_ipsec.h* trong phần mềm strongswan-5.8.4:

**strongswan-5.8.4\src\libcharon\plugins\kernel_netlink\
kernel_netlink_ipsec.h**

```

#define SPD_OUTPUT_TABLE_DEPTH          16
#define SPD_INPUT_TABLE_DEPTH           16
#define SPD_SECURE_TABLE_DEPTH          512
#define SAD_OUTPUT_TABLE_DEPTH          512
#define SAD_INPUT_TABLE_DEPTH           512

/* SPD output table */
#define SPD_OUTPUT_SRC_IP_OFFSET         0x0
#define SPD_OUTPUT_SRC_IP_MASK_OFFSET   0x4
#define SPD_OUTPUT_DST_IP_OFFSET        0x8
#define SPD_OUTPUT_DST_IP_MASK_OFFSET   0xc
#define SPD_OUTPUT_SRC_PORT_OFFSET       0x10
#define SPD_OUTPUT_SRC_PORT_MASK_OFFSET 0x14
#define SPD_OUTPUT_DST_PORT_OFFSET       0x18
#define SPD_OUTPUT_DST_PORT_MASK_OFFSET 0x1c
#define SPD_OUTPUT_PROTOCOL_OFFSET       0x20
#define SPD_OUTPUT_PROTOCOL_MASK_OFFSET 0x24
#define SPD_OUTPUT_SPD_WR_ADDR_OFFSET    0x28
#define SPD_OUTPUT_SPD_RD_ADDR_OFFSET    0x2c

/* SPD input table */
#define SPD_INPUT_SRC_IP_OFFSET          0x80
#define SPD_INPUT_SRC_IP_MASK_OFFSET     0x84
#define SPD_INPUT_DST_IP_OFFSET          0x88
#define SPD_INPUT_DST_IP_MASK_OFFSET     0x8c
#define SPD_INPUT_SRC_PORT_OFFSET        0x90
#define SPD_INPUT_SRC_PORT_MASK_OFFSET   0x94
#define SPD_INPUT_DST_PORT_OFFSET        0x98
#define SPD_INPUT_DST_PORT_MASK_OFFSET   0x9c

```

```

#define SPD_INPUT_PROTOCOL_OFFSET          0xa0
#define SPD_INPUT_PROTOCOL_MASK_OFFSET     0xa4
#define SPD_INPUT_SPD_WR_ADDR_OFFSET      0xa8
#define SPD_INPUT_SPD_RD_ADDR_OFFSET      0xac

/* SPD secure table */
#define SPD_SECURE_SRC_IP_OFFSET          0x100
#define SPD_SECURE_SRC_IP_MASK_OFFSET     0x104
#define SPD_SECURE_DST_IP_OFFSET          0x108
#define SPD_SECURE_DST_IP_MASK_OFFSET     0x10c
#define SPD_SECURE_SRC_PORT_OFFSET        0x110
#define SPD_SECURE_SRC_PORT_MASK_OFFSET   0x114
#define SPD_SECURE_DST_PORT_OFFSET        0x118
#define SPD_SECURE_DST_PORT_MASK_OFFSET   0x11c
#define SPD_SECURE_PROTOCOL_OFFSET        0x120
#define SPD_SECURE_PROTOCOL_MASK_OFFSET   0x124
#define SPD_SECURE_SA_ID_OFFSET           0x128
#define SPD_SECURE_SRC_TUNNEL_OFFSET       0x12c
#define SPD_SECURE_DST_TUNNEL_OFFSET       0x130
#define SPD_SECURE_SPD_WR_ADDR_OFFSET     0x134
#define SPD_SECURE_SPD_RD_ADDR_OFFSET     0x138

/* SAD output table */
#define SAD_OUTPUT_SPI_OFFSET              0x180
#define SAD_OUTPUT_ESN_HI_OFFSET           0x184
#define SAD_OUTPUT_ESN_LOW_OFFSET          0x188
#define SAD_OUTPUT_KEY_CIPHER_0_OFFSET     0x18c
#define SAD_OUTPUT_KEY_CIPHER_1_OFFSET     0x190
#define SAD_OUTPUT_KEY_CIPHER_2_OFFSET     0x194
#define SAD_OUTPUT_KEY_CIPHER_3_OFFSET     0x198
#define SAD_OUTPUT_KEY_CIPHER_4_OFFSET     0x19c
#define SAD_OUTPUT_KEY_CIPHER_5_OFFSET     0x1a0
#define SAD_OUTPUT_KEY_CIPHER_6_OFFSET     0x1a4
#define SAD_OUTPUT_KEY_CIPHER_7_OFFSET     0x1a8
#define SAD_OUTPUT_IV_0_OFFSET              0x1ac
#define SAD_OUTPUT_IV_1_OFFSET              0x1b0
#define SAD_OUTPUT_IV_2_OFFSET              0x1b4
#define SAD_OUTPUT_IV_3_OFFSET              0x1b8
#define SAD_OUTPUT_KEY_HMAC_0_OFFSET        0x1bc
#define SAD_OUTPUT_KEY_HMAC_1_OFFSET        0x1c0
#define SAD_OUTPUT_KEY_HMAC_2_OFFSET        0x1c4
#define SAD_OUTPUT_KEY_HMAC_3_OFFSET        0x1c8
#define SAD_OUTPUT_KEY_HMAC_4_OFFSET        0x1cc
#define SAD_OUTPUT_KEY_HMAC_5_OFFSET        0x1d0
#define SAD_OUTPUT_KEY_HMAC_6_OFFSET        0x1d4
#define SAD_OUTPUT_KEY_HMAC_7_OFFSET        0x1d8
#define SAD_OUTPUT_UDP_ENCAP_OFFSET         0x1dc
#define SAD_OUTPUT_UDP_SRC_PORT_OFFSET      0x1e0
#define SAD_OUTPUT_UDP_DST_PORT_OFFSET      0x1e4
#define SAD_OUTPUT_CIPHER_SUITE_OFFSET      0x1e8
#define SAD_OUTPUT_SAD_WR_ADDR_OFFSET       0x1ec
#define SAD_OUTPUT_SAD_RD_ADDR_OFFSET       0x1f0

```

```

/* SAD input table */
#define SAD_INPUT_SPI_OFFSET 0x200
#define SAD_INPUT_SEQ_CNT_HI_OFFSET 0x204
#define SAD_INPUT_SEQ_CNT_LOW_OFFSET 0x208
#define SAD_INPUT_BITMAP_HI_OFFSET 0x20c
#define SAD_INPUT_BITMAP_LOW_OFFSET 0x210
#define SAD_INPUT_TOP_WINDOW_HI_OFFSET 0x214
#define SAD_INPUT_TOP_WINDOW_LOW_OFFSET 0x218
#define SAD_INPUT_KEY_CIPHER_0_OFFSET 0x21c
#define SAD_INPUT_KEY_CIPHER_1_OFFSET 0x220
#define SAD_INPUT_KEY_CIPHER_2_OFFSET 0x224
#define SAD_INPUT_KEY_CIPHER_3_OFFSET 0x228
#define SAD_INPUT_KEY_CIPHER_4_OFFSET 0x22c
#define SAD_INPUT_KEY_CIPHER_5_OFFSET 0x230
#define SAD_INPUT_KEY_CIPHER_6_OFFSET 0x234
#define SAD_INPUT_KEY_CIPHER_7_OFFSET 0x238
#define SAD_INPUT_IV_0_OFFSET 0x23c
#define SAD_INPUT_IV_1_OFFSET 0x240
#define SAD_INPUT_IV_2_OFFSET 0x244
#define SAD_INPUT_IV_3_OFFSET 0x248
#define SAD_INPUT_KEY_HMAC_0_OFFSET 0x24c
#define SAD_INPUT_KEY_HMAC_1_OFFSET 0x250
#define SAD_INPUT_KEY_HMAC_2_OFFSET 0x254
#define SAD_INPUT_KEY_HMAC_3_OFFSET 0x258
#define SAD_INPUT_KEY_HMAC_4_OFFSET 0x25c
#define SAD_INPUT_KEY_HMAC_5_OFFSET 0x260
#define SAD_INPUT_KEY_HMAC_6_OFFSET 0x264
#define SAD_INPUT_KEY_HMAC_7_OFFSET 0x268
#define SAD_INPUT_CIPHER_SUITE_OFFSET 0x26c
#define SAD_INPUT_SAD_WR_ADDR_OFFSET 0x270
#define SAD_INPUT_SAD_RD_ADDR_OFFSET 0x274

#define MAP_SIZE 0x10000

```

Khi strongswan thiết lập SPD, SAD, đồng thời ghi dữ liệu vào các bảng theo các địa chỉ ở trên vào phần cứng (chỉnh sửa trong *kernel_netlink_ipsec.c* của strongswan-5.8.4).

```

strongswan-5.8.4\src\libcharon\plugins\kernel_netlink\
kernel_netlink_ipsec.c

METHOD(kernel_ipsec_t, add_sa, status_t,
        private_kernel_netlink_ipsec_t *this, kernel_ipsec_sa_id_t *id,
        kernel_ipsec_add_sa_t *data)
{
    netlink_buf_t request;
    const char *alg_name;
    char markstr[32] = "";
    struct nlmsgghdr *hdr;
    struct xfrm_usersa_info *sa;
    uint16_t icv_size = 64, ipcomp = data->ipcomp;

```

```

ipsec_mode_t mode = data->mode, original_mode = data->mode;
traffic_selector_t *first_src_ts, *first_dst_ts;
status_t status = FAILED;
rng_t *rng;
char tmp[1024];
char iv[8];
int i;

rng = lib->crypto->create_rng(lib->crypto, RNG_WEAK);
if (!rng)
{
    DBG1(DBG_KNL, "no RNG found !");
    goto failed;
}

rng->get_bytes(rng, 8, iv);

if (data->reqid > SAD_OUTPUT_TABLE_DEPTH - 1)
{
    goto failed;
}

this->mutex->lock(this->mutex);

if (data->inbound)
{
    fd = open("/dev/uiol", O_RDWR|O_SYNC);
    if (fd < 1) {
        this->mutex->unlock(this->mutex);
        printf("ERROR opening device /dev/uiol");
        goto failed;
    }
    ptr = mmap(NULL, MAP_SIZE, PROT_READ|PROT_WRITE,
               MAP_SHARED, fd, 0);
    if (ptr == MAP_FAILED) {
        perror("mmap");
        fprintf(stderr, "Couldn't mmap.\n");
        this->mutex->unlock(this->mutex);
        goto failed;
    }
}

if ((data->enc_alg == ENCR_AES_GCM_ICV8 ||
     data->enc_alg == ENCR_AES_GCM_ICV12 ||
     data->enc_alg == ENCR_AES_GCM_ICV16) && (data->reqid <=
SAD_OUTPUT_TABLE_DEPTH - 1))
{
    writeReg(ptr, SAD_INPUT_SPI_OFFSET, ntohl(id->spi));
    writeReg(ptr, SAD_INPUT_SEQ_CNT_HI_OFFSET, 0);
    writeReg(ptr, SAD_INPUT_SEQ_CNT_LOW_OFFSET, 0);
    writeReg(ptr, SAD_INPUT_BITMAP_HI_OFFSET, 0);
    writeReg(ptr, SAD_INPUT_BITMAP_LOW_OFFSET, 0);
    writeReg(ptr, SAD_INPUT_TOP_WINDOW_HI_OFFSET, 0);
    writeReg(ptr, SAD_INPUT_TOP_WINDOW_LOW_OFFSET, 0);
}

```

```

        writeReg(ptr, SAD_INPUT_KEY_CIPHER_0_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_1_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 4)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_2_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 8)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_3_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 12)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_4_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 16)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_5_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 20)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_6_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 24)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_7_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 28)));

        if (data->enc_key.len > 32)
            writeReg(ptr, SAD_INPUT_IV_0_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 32)));
        else
            writeReg(ptr, SAD_INPUT_IV_0_OFFSET, 0);
            writeReg(ptr, SAD_INPUT_IV_1_OFFSET, 0);
            writeReg(ptr, SAD_INPUT_IV_2_OFFSET, 0);
            writeReg(ptr, SAD_INPUT_IV_3_OFFSET, 0);

        writeReg(ptr, SAD_INPUT_KEY_HMAC_0_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_HMAC_1_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_HMAC_2_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_HMAC_3_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_HMAC_4_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_HMAC_5_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_HMAC_6_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_HMAC_7_OFFSET, 0);

        writeReg(ptr, SAD_INPUT_CIPHER_SUITE_OFFSET, 1);

        writeReg(ptr, SAD_INPUT_SAD_WR_ADDR_OFFSET, data->reqid - 1);

    }
    else
    {
        writeReg(ptr, SAD_INPUT_SPI_OFFSET, ntohl(id->spi));
        writeReg(ptr, SAD_INPUT_SEQ_CNT_HI_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_SEQ_CNT_LOW_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_BITMAP_HI_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_BITMAP_LOW_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_TOP_WINDOW_HI_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_TOP_WINDOW_LOW_OFFSET, 0);
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_0_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr)));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_1_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 4)));

```

```

        writeReg(ptr, SAD_INPUT_KEY_CIPHER_2_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 8))));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_3_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 12))));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_4_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 16))));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_5_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 20))));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_6_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 24))));
        writeReg(ptr, SAD_INPUT_KEY_CIPHER_7_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 28))));

        if (data->enc_key.len > 32)
            writeReg(ptr, SAD_INPUT_IV_0_OFFSET, ntohl(*(unsigned *)
(data->enc_key.ptr + 32))));
        else
            writeReg(ptr, SAD_INPUT_IV_0_OFFSET, 0);
            writeReg(ptr, SAD_INPUT_IV_1_OFFSET, 0);
            writeReg(ptr, SAD_INPUT_IV_2_OFFSET, 0);
            writeReg(ptr, SAD_INPUT_IV_3_OFFSET, 0);

        writeReg(ptr, SAD_INPUT_KEY_HMAC_0_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr))));
        writeReg(ptr, SAD_INPUT_KEY_HMAC_1_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr + 4))));
        writeReg(ptr, SAD_INPUT_KEY_HMAC_2_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr + 8))));
        writeReg(ptr, SAD_INPUT_KEY_HMAC_3_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr + 12))));
        writeReg(ptr, SAD_INPUT_KEY_HMAC_4_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr + 16))));
        writeReg(ptr, SAD_INPUT_KEY_HMAC_5_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr + 20))));
        writeReg(ptr, SAD_INPUT_KEY_HMAC_6_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr + 24))));
        writeReg(ptr, SAD_INPUT_KEY_HMAC_7_OFFSET, ntohl(*(unsigned *)
(data->int_key.ptr + 28))));

        writeReg(ptr, SAD_INPUT_CIPHER_SUITE_OFFSET, 2);
        writeReg(ptr, SAD_INPUT_SAD_WR_ADDR_OFFSET, data->reqid - 1);
    }

    munmap(ptr, MAP_SIZE);
    close(fd);
}
else
{
    fd = open("/dev/uiol", O_RDWR|O_SYNC);
    if (fd < 1) {
        printf("ERROR opening device /dev/uiol");

        this->mutex->unlock(this->mutex);
        goto failed;
    }
}

```



```

    }

    ptr = mmap(NULL, MAP_SIZE, PROT_READ|PROT_WRITE,
               MAP_SHARED, fd, 0);
    if (ptr == MAP_FAILED) {
        perror("mmap");
        fprintf(stderr, "Couldn't mmap.\n");

        this->mutex->unlock(this->mutex);
        goto failed;
    }

    if ((data->enc_alg == ENCR_AES_GCM_ICV8 ||
        data->enc_alg == ENCR_AES_GCM_ICV12 ||
        data->enc_alg == ENCR_AES_GCM_ICV16) && (data->reqid <=
SAD_OUTPUT_TABLE_DEPTH - 1))
    {
        writeReg(ptr, SAD_OUTPUT_SPI_OFFSET, ntohl(id->spi));
        writeReg(ptr, SAD_OUTPUT_ESN_HI_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_ESN_LOW_OFFSET, 1);
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_0_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_1_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 4))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_2_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 8))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_3_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 12))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_4_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 16))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_5_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 20))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_6_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 24))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_7_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 28))));

        if (data->enc_key.len > 32)
            writeReg(ptr, SAD_OUTPUT_IV_0_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 32))));
        else
            writeReg(ptr, SAD_OUTPUT_IV_0_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_IV_1_OFFSET, ntohl(iv));
        writeReg(ptr, SAD_OUTPUT_IV_2_OFFSET, ntohl(iv + 4));
        writeReg(ptr, SAD_OUTPUT_IV_3_OFFSET, 0);

        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_0_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_1_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_2_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_3_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_4_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_5_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_6_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_7_OFFSET, 0);
    }

```

```

        if (data->encap)
        {
            writeReg(ptr, SAD_OUTPUT_UDP_ENCAP_OFFSET, 1);
            writeReg(ptr, SAD_OUTPUT_UDP_SRC_PORT_OFFSET,      id->src-
>get_port(id->src));
            writeReg(ptr, SAD_OUTPUT_UDP_DST_PORT_OFFSET,      id->dst-
>get_port(id->dst));
        }
        else
        {
            writeReg(ptr, SAD_OUTPUT_UDP_ENCAP_OFFSET, 0);
            writeReg(ptr, SAD_OUTPUT_UDP_SRC_PORT_OFFSET, 0);
            writeReg(ptr, SAD_OUTPUT_UDP_DST_PORT_OFFSET, 0);
        }

        writeReg(ptr, SAD_OUTPUT_CIPHER_SUITE_OFFSET, 1);
        writeReg(ptr, SAD_OUTPUT_SAD_WR_ADDR_OFFSET, data->reqid - 1);
    }
    else
    {
        writeReg(ptr, SAD_OUTPUT_SPI_OFFSET, ntohl(id->spi));
        writeReg(ptr, SAD_OUTPUT_ESN_HI_OFFSET, 0);
        writeReg(ptr, SAD_OUTPUT_ESN_LOW_OFFSET, 1);
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_0_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_1_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 4))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_2_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 8))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_3_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 12))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_4_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 16))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_5_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 20))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_6_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 24))));
        writeReg(ptr, SAD_OUTPUT_KEY_CIPHER_7_OFFSET, ntohl(*((unsigned *)
(data->enc_key.ptr + 28))));

        if (data->enc_key.len > 32)
            writeReg(ptr, SAD_OUTPUT_IV_0_OFFSET,      ntohl(*((unsigned *)
(data->enc_key.ptr + 32))));
        else
        {
            writeReg(ptr, SAD_OUTPUT_IV_0_OFFSET, 0);
            writeReg(ptr, SAD_OUTPUT_IV_1_OFFSET, ntohl(iv));
            writeReg(ptr, SAD_OUTPUT_IV_2_OFFSET, ntohl(iv + 4));
            writeReg(ptr, SAD_OUTPUT_IV_3_OFFSET, 0);

            writeReg(ptr, SAD_OUTPUT_KEY_HMAC_0_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr))));
            writeReg(ptr, SAD_OUTPUT_KEY_HMAC_1_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr + 4))));

```

```

        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_2_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr + 8))));
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_3_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr + 12))));
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_4_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr + 16))));
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_5_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr + 20))));
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_6_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr + 24))));
        writeReg(ptr, SAD_OUTPUT_KEY_HMAC_7_OFFSET, ntohl(*((unsigned *)
(data->int_key.ptr + 28))));

        if (data->encap)
        {
            writeReg(ptr, SAD_OUTPUT_UDP_ENCAP_OFFSET, 1);
            writeReg(ptr, SAD_OUTPUT_UDP_SRC_PORT_OFFSET, id->src-
>get_port(id->src));
            writeReg(ptr, SAD_OUTPUT_UDP_DST_PORT_OFFSET, id->dst-
>get_port(id->dst));
        }
        else
        {
            writeReg(ptr, SAD_OUTPUT_UDP_ENCAP_OFFSET, 0);
            writeReg(ptr, SAD_OUTPUT_UDP_SRC_PORT_OFFSET, 0);
            writeReg(ptr, SAD_OUTPUT_UDP_DST_PORT_OFFSET, 0);
        }
        writeReg(ptr, SAD_OUTPUT_CIPHER_SUITE_OFFSET, 2);
        writeReg(ptr, SAD_OUTPUT_SAD_WR_ADDR_OFFSET, data->reqid - 1);
    }
    munmap(ptr, MAP_SIZE);
    close(fd);
}

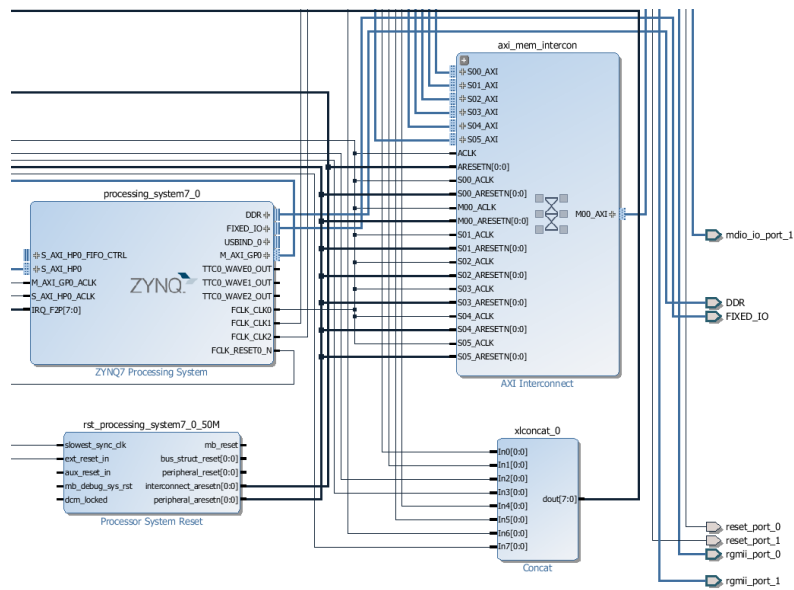
this->mutex->unlock(this->mutex);

....
....

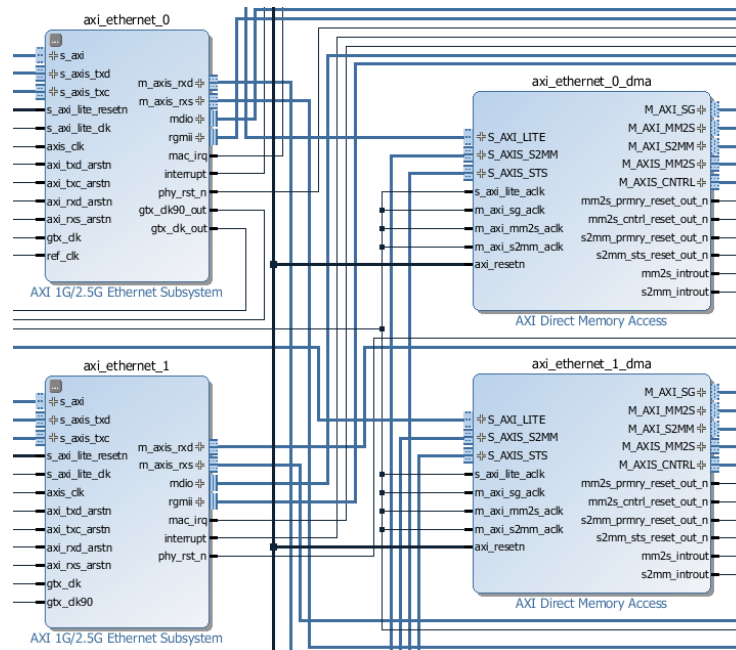
```

Tiếp theo tổng hợp thiết kế HMS-IPSec để lấy bitstream. Thiết kế bao gồm đầy đủ các IPCores theo kiến trúc đã đề xuất:

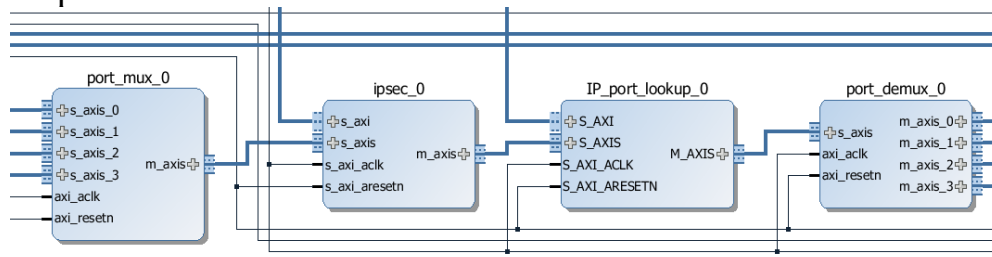
Thêm khối ZYNQ7 Processing System



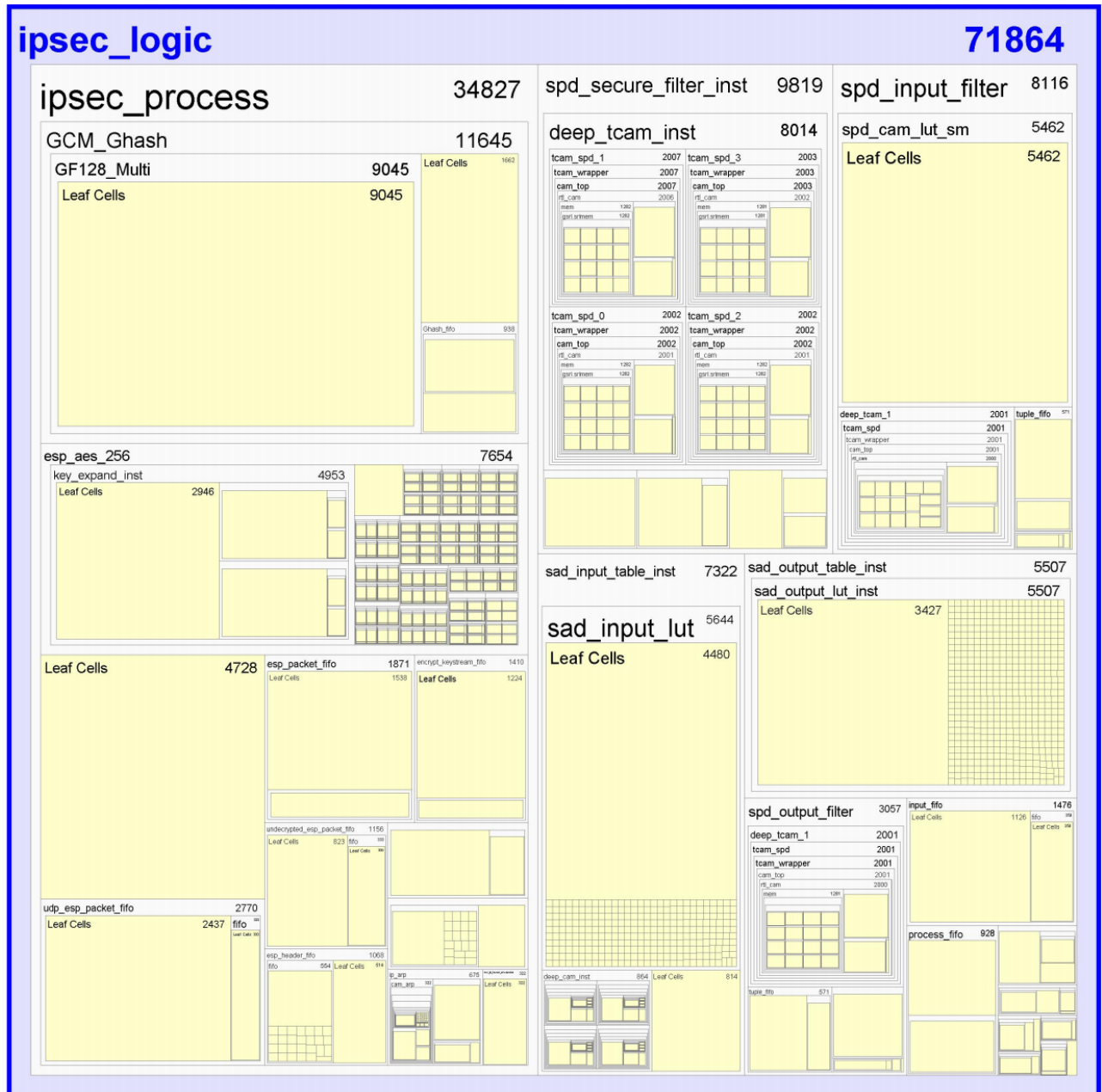
Thêm khối MAC và MAC DMA



Ghép các khối vào thiết kế



Phân lớp các modules của khối IPSec trên Vivado:



Cuối cùng tạo file BOOT.BIN cùng với FPGA bằng lệnh:

```
petalinux-package --boot --fsbl ./images/linux/zynq_fsbl.elf --fpga
../hsm ipsec/hms ipsec.runs/impl 1/design 1 wrapper.bit --uboot -force
```

Copy file BOOT.BIN và rootfs vào hai phân vùng khác nhau trên SD card.

Truy cập console vào Kit ZC706 và cấu hình IPSec trong tệp ipsec.conf

```
COM3 - PuTTY
# ipsec.conf - strongSwan IPsec configuration file

# basic configuration

config setup
    # strictcrulpolicy=yes
    # uniqueids = no

conn %default
    ikelifetime=500m
    keylife=450m
    rekeymargin=30s
    keyingtries=3
    keyexchange=ikev2
    #esp=aes256ctr-sha256-esn
    esp=aes256gcm16-esn

conn test0
    left=192.168.0.2
    leftid=@pc.com
    leftsubnet=20.0.0.0/24
    leftauth=psk
    right=192.168.0.1
    rightid=@kit.com
    rightsubnet=10.0.0.0/24
    rightauth=psk
    auto=add
```

Tạo kết nối giữa hai Kit ZC706:

```
COM3 - PuTTY
root@IPSEC_GW2:~# /usr/ipsec/sbin/ipsec up test0
initiating IKE_SA test0[1] to 192.168.0.1
generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP)
N(HASH_ALG) N(REDIR_SUP) ]
sending packet: from 192.168.0.2[500] to 192.168.0.1[500] (392 bytes)
received packet: from 192.168.0.1[500] to 192.168.0.2[500] (248 bytes)
parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N
(HASH_ALG) N(CHDLESS_SUP) N(MULT_AUTH) ]
selected proposal: IKE:AES_CBC_128/HMAC_SHA2_256_128/PRF_AES128_XCBC/CURVE_25519
authentication of 'pc.com' (myself) with pre-shared key
establishing CHILD_SA test0{1}
libcharon/plugins/kernel_netlink/kernel_netlink_ipsec.c get_spi_internal
generating IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr AUTH SA TSr N(MOBIKE
_SUP) N(ADD_4_ADDR) N(MULT_AUTH) N(EAP_ONLY) N(MSG_ID_SYN_SUP) ]
sending packet: from 192.168.0.2[4500] to 192.168.0.1[4500] (352 bytes)
received packet: from 192.168.0.1[4500] to 192.168.0.2[4500] (224 bytes)
parsed IKE_AUTH response 1 [ IDr AUTH SA TSr N(AUTH_LFT) N(MOBIKE_SUP) N(ADD
_4_ADDR) ]
authentication of 'kit.com' with pre-shared key successful
IKE_SA test0[1] established between 192.168.0.2[pc.com]...192.168.0.1[kit.com]
scheduling reauthentication in 29945s
maximum IKE_SA lifetime 29975s
selected proposal: ESP:AES_GCM_16_256/EXT_SEQ
CHILD_SA test0{1} established with SPIs cbee982b_i c5f24355_o and TS 20.0.0.0/24
=== 10.0.0.0/24
connection 'test0' established successfully
root@IPSEC_GW2:~#
```

Thực hiện đo thông lượng mạng sử dụng iperf3 giữa hai máy 10.0.0.10 và 20.0.0.10:

```
C:\Windows\System32\cmd.exe
D:\Working\Ipssec\iperf-3.1.3-win64>iperf3.exe -c 10.0.0.10 -w 128K -t 120
Connecting to host 10.0.0.10, port 5201
[ 4] local 20.0.0.10 port 29529 connected to 10.0.0.10 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.00-1.00  sec    109 MBytes  911 Mbits/sec
[ 4] 1.00-2.00  sec    108 MBytes  910 Mbits/sec
[ 4] 2.00-3.00  sec    109 MBytes  911 Mbits/sec
[ 4] 3.00-4.00  sec    109 MBytes  911 Mbits/sec
[ 4] 4.00-5.00  sec    108 MBytes  909 Mbits/sec
[ 4] 5.00-6.00  sec    109 MBytes  911 Mbits/sec
[ 4] 6.00-7.00  sec    109 MBytes  911 Mbits/sec
[ 4] 7.00-8.00  sec    108 MBytes  911 Mbits/sec
[ 4] 8.00-9.00  sec    109 MBytes  911 Mbits/sec
[ 4] 9.00-10.00 sec    108 MBytes  909 Mbits/sec
[ 4] 10.00-11.00 sec    108 MBytes  910 Mbits/sec
[ 4] 11.00-12.00 sec    109 MBytes  912 Mbits/sec
[ 4] 12.00-13.00 sec    109 MBytes  911 Mbits/sec
[ 4] 13.00-14.00 sec    109 MBytes  912 Mbits/sec
[ 4] 14.00-15.00 sec    108 MBytes  910 Mbits/sec
[ 4] 15.00-16.00 sec    109 MBytes  911 Mbits/sec
[ 4] 16.00-17.00 sec    108 MBytes  910 Mbits/sec
[ 4] 17.00-18.00 sec    107 MBytes  901 Mbits/sec
[ 4] 18.00-19.00 sec    109 MBytes  912 Mbits/sec
[ 4] 19.00-20.00 sec    109 MBytes  911 Mbits/sec
[ 4] 20.00-21.00 sec    108 MBytes  911 Mbits/sec
[ 4] 21.00-22.00 sec    109 MBytes  912 Mbits/sec
[ 4] 22.00-23.00 sec    109 MBytes  912 Mbits/sec
[ 4] 23.00-23.37 sec    40.0 MBytes  910 Mbits/sec
```