BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Phạm Văn Lực

NGHIÊN CỨU GIẢI PHÁP NÂNG CAO HIỆU QUẢ SỬ DỤNG MẬT MÃ ĐƯỜNG CONG ELLIPTIC TRÊN CÁC THIẾT BỊ TÍNH TOÁN NHÚNG

LUẬN ÁN TIẾN SĨ KỸ THUẬT

Hà Nội - 2022

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Phạm Văn Lực

NGHIÊN CỨU GIẢI PHÁP NÂNG CAO HIỆU QUẢ SỬ DỤNG MẬT MÃ ĐƯỜNG CONG ELLIPTIC TRÊN CÁC THIẾT BỊ TÍNH TOÁN NHÚNG

CHUYÊN NGÀNH: KỸ THUẬT ĐIỆN TỬ
MÃ SỐ: 9.52.02.03

LUẬN ÁN TIẾN SĨ KỸ THUẬT

NGƯỜI HƯỚNG DẪN KHOA HỌC
1. PGS. TSKH HOÀNG ĐĂNG HẢI
2. TS. LỀU ĐỨC TÂN

Hà Nội - 2022

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các nội dung, số liệu và kết quả nghiên cứu 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, các dữ liệu tham khảo được trích dẫn đầy đủ.

Người cam đoan

Phạm Văn Lực

LỜI CẢM ƠN

Luận án này được thực hiện tại Học viện Công nghệ Bưu chính Viễn thông - Bộ Thông tin và Truyền thông.

Nghiên cứu sinh xin được bày tỏ lòng biết ơn sâu sắc đến Thầy giáo PGS. TSKH. Hoàng Đăng Hải, TS. Lều Đức Tân đã tận tình hướng dẫn, giúp đỡ, trang bị phương pháp nghiên cứu, kiến thức khoa học để tôi hoàn thành các nội dung nghiên cứu của luận án.

Nghiên cứu sinh xin bày tỏ lòng biết ơn chân thành tới các thầy, cô của Học viện Công nghệ Bưu chính Viễn thông, các nhà khoa học thuộc Viện Khoa học - Công nghệ mật mã đã đóng góp nhiều ý kiến quý báu giúp tôi hoàn thành các nội dung nghiên cứu của luận án.

Nghiên cứu sinh xin trân trọng cảm ơn Học viện Công nghệ Bưu chính Viễn thông, Khoa Khoa Quốc tế và Đào tạo Sau đại học là cơ sở đào tạo và đơn vị quản lý, các đồng chí Lãnh đạo Viện Khoa học - Công nghệ mật mã, nơi tôi đang công tác đã tạo điều kiện thuận lợi, hỗ trợ và giúp đỡ tôi trong suốt quá trình học tập, nghiên cứu thực hiện luận án.

Tôi xin trân trọng cảm ơn các bạn bè người thân và gia đình đã cổ vũ, động viên giúp đỡ, tạo điều kiện cho tôi hoàn thành luận án.

Nghiên cứu sinh

Phạm Văn Lực

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CÁC TỪ VIẾT TẮT

F_q		Trường hữu hạn q
#(a)		Lực lượng của a
#(b)		Lực lượng của b
#(E)		Lực lượng của tập E, số điểm của đường cong E
$\mathbb{F}_{2^n}$		Trường nhị phân
ARM	Advanced RISC Machine	Kiến trúc vi xử lý sử dụng tập lệnh RISC
ASM	Assembly	Ngôn ngữ lập trình bậc thấp
$E(F_q)$		Nhóm các điểm của đường cong Elliptic trên trường hữu hạn q
ECDLP	Elliptic Curve Discrete Logarithm Problem	Bài toán Logarithm rời rạc trên nhóm các điểm của đường cong Elliptic
ECIES	Elliptic Curve Integrated Encryption System	Hệ mã hóa tích hợp dựa trên đường cong elliptic
FPGA	Field Programmable Gate Array	Công nghệ lập trình phần cứng
FPU	Floating Point Unit	Bộ thực hiện lệnh số học
GF(p)		Trường hữu hạn p nguyên tố
GPU	Graphical Processor Units	Đơn vị xử lý đồ họa
IKEv2	Internet Key Exchange version 2	Giao thức thỏa thuận khóa trên Internet phiên bản 2
IPSec	IP Security Protocol	Giao thức bảo mật dữ liệu IP
ISO/IEC	International Standard Organization/International Electrotechnical Commission	Tổ chức tiêu chuẩn quốc tế
k.P		Phép nhân một số nguyên k với điểm P trên đường cong Elliptic
MIPS	Million Instructions Per Second	Đơn vị đo số chỉ lệnh thực hiện trong một giây
NAF	Non-Adjacent Form	Dạng không liền kề
NEON		Một đồng xử lý tích hợp trong vi xử lý ARM
NIST	National Institute of Standards and Technology	Các chuẩn về công nghệ quốc gia của Mỹ
RISC	Reduced Instructions Set Computer	Tập chỉ lệnh tối giản cho vi xử lý

RSA	Rivest-Shamir-Adlemen	Hệ mật RSA
SIMD	Single Instruction Multiple Data	Cơ chế thực hiện song song (đơn chỉ thị đa dữ liệu)
SOC	System on Chip	Vi mạch tích hợp đa thành phần
TLS	Transport Layer Security	Giao thức bảo mật tầng vận chuyển
VPN	Virtual Private Network	Mạng riêng ảo
wNAF	window Non-Adjacent Form	Dạng không liền kề cửa sổ độ rộng *w*

DANH MỤC HÌNH VẼ

DANH MỤC CÁC BẢNG

MỞ ĐẦU

1. Lý do chọn đề tài

Các thiết bị tính toán nhúng (gọi tắt là các hệ thống nhúng) đang được ứng dụng rộng rãi trong nhiều lĩnh vực, điển hình như trong các thiết bị IoT (Internet of Things), thiết bị di động, thiết bị cầm tay, máy tính bảng, thẻ thông minh, các bộ điều khiển trong xe ô tô, các bộ cảm biến môi trường, các hộp kỹ thuật số,…Theo thống kê, các thiết bị sử dụng hệ thống nhúng chiếm tới 90% các thiết bị tính toán hiện nay. Do chủng loại và môi trường kết nối đa dạng, nhu cầu bảo đảm an toàn và bảo mật thông tin cho các thiết bị di động nói chung và các hệ thống nhúng nói riêng đang trở nên cấp thiết. Một lý do nữa là những thiết bị này thường là không dây, khả năng truy cập vật lý mọi lúc - mọi nơi, vấn đề an toàn bảo mật dữ liệu và bảo vệ truy cập để ngăn chặn tấn công nghe lén và rò rỉ thông tin cá nhân là rất quan trọng.

Đặc điểm chung của các hệ thống nhúng là có hạn chế về tài nguyên và năng lực xử lý. Mặc dù cấu hình của nhiều thiết bị đã được cải thiện đáng kể, song so với các máy tính có năng lực xử lý mạnh, các hệ thống nhúng thường có kích thước nhỏ hơn, khả năng xử lý yếu hơn, bộ nhớ nhỏ hơn và yêu cầu tiêu thụ ít năng lượng do sử dụng nguồn pin. Việc triển khai các giải pháp an toàn bảo mật thông tin cho các hệ thống nhúng có thêm nhiều thách thức so với các hệ thống máy tính truyền thống. Các giải pháp mật mã sử dụng các nguyên thủy mật mã truyền thống như RSA trong các giao thức trao đổi khóa như IKE, TLS, OpenVPN không còn phù hợp. Trên các hệ thống nhúng có hạn chế tài nguyên và năng lực xử lý, không thể áp dụng các hệ mật mã với yêu cầu tính toán cao, tiêu thụ nhiều tài nguyên cũng như nguồn năng lượng lớn. Do năng lực xử lý của CPU và khả năng lưu trữ hạn chế của bộ nhớ, cần có các giải pháp an toàn bảo mật thông tin hạng nhẹ cho các thiết bị nhúng. Nghiên cứu đề xuất các lược đồ mật mã hạng nhẹ đạt hiệu quả cao trên các thiết bị nhúng đang là chủ đề nghiên cứu rất cấp thiết, được quan tâm nhiều và có ý nghĩa thực tiễn cao do các thiết bị nhúng đang được ứng dụng rất rộng rãi.

Các giải pháp bảo mật thường dựa trên nền tảng mật mã khóa công khai và mật mã khóa đối xứng. Do độ phức tạp tính toán cao, các giải pháp này không thể áp dụng

cho các hệ thống nhúng với tài nguyên hạn chế. Hệ mật đường cong Elliptic (ECC – Elliptic Curve Cryptography) đã được đề xuất cho các hệ thống nhúng do kích thước khóa nhỏ hơn nhiều so với các hệ mật khóa công khai khác. Các lược đồ mã hóa ECC đã được đề xuất điển hình là lược đồ thỏa thuận khóa Diffie Hellman trên đường cong elliptic (ECDH) và thuật toán chữ ký số trên đường cong elliptic (ECDSA). Các lược đồ này chủ yếu xây dựng trên nền ECC với phép tính mật mã cơ bản là phép nhân vô hướng của một điểm trên đường cong với một số nguyên.

Nhiều nghiên cứu về hệ mật đường cong Elliptic trên thiết bị nhúng đã được thực hiện trong những năm qua. Các nghiên cứu chủ yếu tập trung vào vấn đề lựa chọn tham số, số modulo an toàn và phù hợp với môi trường có tài nguyên hạn chế. Đã có một số cải tiến thuật toán cùng với việc sử dụng nền tảng hỗ trợ phần cứng (FPGA, GPU, NEON...) cũng như một số cải tiến các phép toán cơ bản. Với các cải tiến dựa trên lựa chọn tham số, hạn chế lớn nhất là việc lựa chọn các tham số đặc biệt sẽ không phù hợp cho các ứng dụng mật mã có yêu cầu độ mật cao. Việc sử dụng các thành phần hỗ trợ như FPGA, GPU hoặc các chip hỗ trợ tính toán cũng còn hạn chế do những khó khăn trong triển khai cài đặt thuật toán và cũng khó thay đổi tham số thường được đặt cố định. Do vậy, nhiều nghiên cứu tập trung vào cải thiện các phép toán cơ bản, ví dụ các phép toán số học dựa trên đặc điểm thiết bị cụ thể và các tham số đặc biệt. Tuy nhiên, các hệ mật này có hạn chế khi áp dụng cho các thiết bị khác, hoặc khi cần sử dụng các tham số khác.

ECC có nhiều ưu việt và phù hợp cho bảo mật dữ liệu trên các thiết bị nhúng. Tuy nhiên, nhiều nghiên cứu đã chỉ ra ECC vẫn còn một số hạn chế khi sử dụng cho các hệ thống nhúng. Tương tự như các hệ mật khóa công khai, việc tính toán trên ECC cũng khá phức tạp và tốn kém liên quan đến một số thuật toán như: cộng điểm, nhân đôi điểm và thời gian tính toán hoàn toàn cũng bị chi phối bởi các phép nhân modulo trong trường hữu hạn. Thực hiện mật mã đường cong Elliptic về bản chất là thực hiện phép nhân điểm và đây cũng là phép tính tiêu tốn thời gian và tài nguyên nhất trong ECC. An toàn của hệ mật đường cong Elliptic dựa trên bài toán khó đó là tìm logarit rời rạc trên đó (ECDLP – Elliptic Curve Discrete Logarithm Problem). Các phép toán

trên ECC được thực hiện từ sự kết hợp của các phép toán cơ bản trên trường hữu hạn, trong đó phép nhân số học là phép tính quan trọng nhất. Hiệu quả của hệ mật ECC phụ thuộc vào lựa chọn các tham số miền đường cong elliptic, trường hữu hạn cơ bản và các thuật toán thực hiện tính toán. Rất khó có được sự lựa chọn có tính tối ưu, đặc biệt trong môi trường hạn chế của các hệ thống nhúng. Tích hợp các phép toán một cách hiệu quả vào các lược đồ ECDH và ECDSA cũng chưa được nghiên cứu đầy đủ. Nghiên cứu đưa ra các thuật toán an toàn và hiệu quả cho ECC trong các hệ thống nhúng có tài nguyên hạn chế vẫn đang là vấn đề có nhiều thách thức.

Vì những lý do nêu trên, luận án chọn chủ đề nghiên cứu là nghiên cứu và đề xuất "Giải pháp nâng cao hiệu quả sử dụng mật mã đường cong Elliptic trên các thiết bị tính toán nhúng".

2. Mục tiêu nghiên cứu

Nghiên cứu, đề xuất các giải pháp để nâng cao hiệu quả thực thi một số thuật toán mật mã đường cong elliptic trên nền hệ thống nhúng sử dụng vi xử lý ARM, đảm bảo hiệu quả về tốc độ tính toán, tài nguyên sử dụng và an toàn trong sử dụng.

3. Đối tượng nghiên cứu

Đối tượng nghiên cứu của luận án là các thuật toán mật mã đường cong Elliptic an toàn và hiệu quả trên môi trường tính toán nhúng.

4. Phạm vi nghiên cứu

Phạm vi nghiên cứu của luận án là các kỹ thuật, thuật toán, an toàn và hiệu quả của mật mã đường cong Elliptic trên môi trường tính toán nhúng sử dụng vi xử lý ARM.

5. Phương pháp nghiên cứu

– Phương pháp nghiên cứu lý luận: Nghiên cứu, phân tích, tổng hợp các thông tin liên quan. Lựa chọn các cách tiếp cận đã được áp dụng thành công. Tiến hành nghiên cứu sâu hơn về giải pháp cải tiến có thể có để xây dựng và đề xuất được các thuật toán mật mã đường cong elliptic mà cân bằng giữa yếu tố an toàn và hiệu quả, phù hợp cho môi trường tính toán nhúng.
– Phương pháp nghiên cứu thực nghiệm: Nghiên cứu, tích hợp các giải pháp đề xuất vào nền tảng phần cứng để minh chứng cho kết quả nghiên cứu lý thuyết.

6. Nội dung

- Nghiên cứu cơ bản về hệ thống nhúng, đặc điểm của hệ thống nhúng và vi xử lý ARM sử dụng trên hệ thống nhúng

- Nghiên cứu cơ sở lý thuyết và các thuật toán của hệ mật đường cong Elliptic, ứng dụng các giao thức của hệ mật đường cong elliptic để bảo đảm an toàn dữ liệu trên thiết bị nhúng. Nghiên cứu, đề xuất phương pháp đánh giá chi phí của thuật toán nhân số lớn mà chỉ phụ thuộc vào phép cộng cơ bản.

- Nghiên cứu, đề xuất cải tiến thuật toán số học trên trường hữu hạn và các thuật toán của hệ mật đường cong Elliptic nhằm nâng cao về hiệu quả thực thi, tài nguyên sử dụng. Tích hợp các giải pháp đề xuất vào thuật toán ECDH và ECDSA trên nền vi xử lý ARM.

7. Ý nghĩa khoa học và thực tiễn

- ***Ý nghĩa khoa học:*** Quá trình nghiên cứu luận án hướng tới nghiên cứu, đề xuất các kỹ thuật mới nhằm cải tiến thuật toán và ứng dụng một số nguyên thủy mật mã vào bài toán bảo mật thông tin trên hệ thống nhúng, cụ thể:

+ Nghiên cứu, đề xuất phương pháp đánh giá chi phí của thuật toán nhân số lớn mà chỉ phụ thuộc vào phép cộng cơ bản

+ Nghiên cứu, đề xuất kỹ thuật nâng cao hiệu quả cho các lược đồ trên đường cong elliptic khóa công khai như thuật toán chữ ký số trên đường cong elliptic (ECDSA) và lược đồ thỏa thuận khóa Diffie Hellman trên đường cong elliptic (ECDH).

- ***Ý nghĩa thực tiễn:*** Các thiết bị IoT, di động (thoại thông minh, máy tính bảng) có sử dụng thiết bị nhúng đang trở nên rất phổ biến và ứng dụng rộng rãi. Các giải pháp được đề xuất trong luận án sẽ góp phần tăng cường khả năng bảo mật thông tin trên các thiết bị nhúng, đáp ứng nhu cầu cấp thiết trong thực tiễn hiện nay về bảo mật dữ liệu, ngăn chặn tấn công nghe lén và rò rỉ thông tin cá nhân. Giải pháp được đề xuất trong luận án được áp dụng cho các thiết bị phổ biến trong thực tiễn.

8. Bố cục của luận án

Ngoài phần mở đầu, phần kết luận và phần phụ lục, luận án gồm ba chương với bố cục như sau.

Chương 1. Tổng quan các vấn đề nghiên cứu

Chương 1 đi sâu phân tích các yêu cầu cần bảo mật dữ liệu trên nền thiết bị nhúng, các giải pháp nâng cao hiệu quả hệ mật khóa công khai Elliptic trên nền các thiết bị nhúng (cải tiến thuật toán, sử dụng các tham số đặc biệt; sử dụng các thành phần tính toán nhanh GPU, DSP, FPGA; sử dụng các đồng xử lý ...). Trong các giải pháp này, luận án cũng đã phân tích những ưu điểm và hạn chế của từng phương pháp. Trên cơ sở đó, luận án sẽ tập trung vào đề xuất hướng nghiên cứu của luận án.

Chương 2. Nâng cao hiệu quả của phép nhân số học trong trường nhị phân trên vi xử lý ARM

Nội dung thứ nhất của chương 2 là sẽ nghiên cứu, phân tích và đánh giá về mặt hiệu năng của một số phương pháp để thực hiện phép tính nhân trong trường hữu hạn (cả trường nhị phân và trường nguyên tố).

Nội dung chính của chương này sẽ nghiên cứu và đề xuất: a)Nghiên cứu, đề xuất phương pháp đánh giá chi phí của thuật toán nhân số lớn mà chỉ phụ thuộc vào phép cộng cơ bản, b) Phương pháp nhân hai số hạng trên trường hữu hạn, gọi là phương pháp phân tầng. Với phương pháp phân tầng, ta có thể xây dựng được thuật toán nhân có chi phí tốt nhất trong các trường hợp cụ thể. Dựa trên thuật toán nhân phân tầng đề xuất, các đề xuất áp dụng cụ thể để nhân hai đa thức trong trường nhị phân dựa trên kết hợp giữa thuật toán Karatsuba, bộ nhân 128-bit và bộ nhân 192-bit trên vi xử lý ARMv7 và ARMv8.

Chương 3. Nâng cao hiệu quả phép nhân vô hướng của hệ mật ECC trong trường nguyên tố trên vi vử lý ARM

Chương 3 của luận án tập trung các giải pháp cải tiến hiệu năng cho các thuật toán mật mã dựa trên đường cong Elliptic trên trường số nguyên tố dựa trên hai cải tiến là: Đề xuất cải tiến thuật toán NAF và giải pháp nâng cao hiệu quả của các phép toán số học (cộng, nhân đôi và nhân vô hướng) trên đường cong Elliptic trong trường số nguyên tố bằng phương pháp song song hai phép nhân đồng thời.

CHƯƠNG 1. TỔNG QUAN CÁC VẤN ĐỀ NGHIÊN CỨU

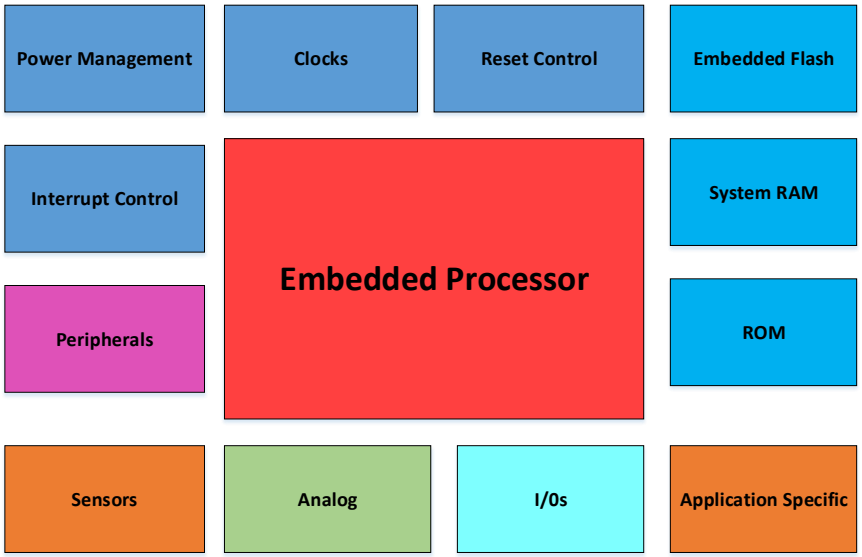
1.1. Hệ thống nhúng

Hệ thống nhúng (Embedded System) là một thuật ngữ thường dùng để chỉ một thiết bị tính toán đặc biệt, có tích hợp cả phần cứng và phần mềm phục vụ các ứng dụng chuyên dụng trong nhiều lĩnh vực của đời sống hiện nay. Khác với các hệ thống phổ biến khác, ví dụ như các máy tính đa năng hay nhiều hệ thống máy tính trong điều khiển công nghiệp, các hệ thống nhúng thường có kích thước nhỏ hơn, khả năng xử lý yếu hơn, bộ nhớ nhỏ hơn và yêu cầu tiêu thụ ít năng lượng [10, 11].

Một hệ thống nhúng gồm ba thành phần chính:

- Phần cứng
- Phần mềm ứng dụng
- Hệ điều hành hoặc một hệ thống thời gian thực (RTOS)

Phần cứng hệ thống nhúng:



Hình 1.1: Các thành phần chính trong một phần cứng nhúng

Phần cứng của một hệ thống nhúng thông thường gồm các thành phần chính sau:

- Quản lý nguồn (Power Management): Chức năng của thành phần này là cung cấp nguồn điện và điều khiển chế độ cấp nguồn nhằm mục đích tối ưu trong tiêu thụ năng lượng. Thành phần này có thể chứa một bộ đồng hồ thời gian thực RTC (Real Time Clock)

- Vi xử lý nhúng (Embedded Processor): Đây là trung tâm của bất kỳ một vi điều khiển nào (microcontroller) dựa trên hệ thông nhúng. Các vi xử lý này được tối ưu về kích thước cũng như chức năng riêng biệt cho từng sản phẩm. Đa phần trong lớp vi xử lý này sẽ tích hợp một vài chức năng DSP cơ bản gồm bộ nhân/chia bằng phần cứng.
- Bộ nhớ (System RAM, ROM, Embedded Flash): Thành phần nhớ RAM trong một hệ thống nhúng nên có thời gian truy cập thấp. Một vài vi điều khiển nhúng gồm thành phần ROM để lưu thành phần phần mềm khởi tạo (bootloader), và có bộ nhớ Flash cho phép lập trình chương trình.
- Ngoại vi (Peripherals) và I/O: Các ngoại vi là các thiết bị vào và ra được kết nối đến cổng nối tiếp hoặc song song của hệ thống nhúng. Một vi điều khiển truyền thông với các ngoại vi sử dụng một giao tiếp có thể lập trình.
- Đồng hồ thời gian (Timers) và Watchdog: Một bộ vi điều khiển sẽ gồm bộ đếm thời gian khác nhau để xác định sự kiện theo thời gian, ví dụ cho phép hệ thống vào chế độ tiết kiệm năng lượng và thoát khỏi chế độ này. Một bộ đếm thời gian đặc biệt gọi là "watchdog timer" khác, là một phần thiết yếu của bất kỳ hệ thống nhúng được sử dụng để phát hiện và khôi phục chương trình khi chạy một đoạn mã gặp sự cố.
- Thành phần cảm biến (Sensors) và tương tự (Analog): Một hệ thống nhúng thường gồm nhiều cảm biến như cảm biến nhiệt độ và các thành phần tương tự như bộ chuyển đổi tương tự - số (ADC), bộ chuyển đổi số - tương tự (DAC)…
- Ngoài ra, trong thiết kế phần cứng hệ thống nhúng còn có các thành phần khác như: điều khiển khởi động hệ thống (Reset control), điều khiển ngắt (Interrupt control), thành phần liên quan đến ứng dụng (Application Specific) …

Phần mềm ứng dụng và hệ điều hành:

Do không có thiết bị ổ cứng trong hệ thống nhúng, nên mã chương trình được lưu ở trong bộ nhớ Flash hoặc bộ nhớ ROM. Trong quá trình thực thi một chương trình, không gian nhớ cho các biến được cấp pháp trong bộ nhớ RAM.

Các ứng dụng hoạt động trên hệ điều hành cơ sở hoặc trên một hệ điều hành thời gian thực RTOS (Real-Time Operating System).

Các hệ thống nhúng rất đa dạng, song có đặc điểm chung là được cấu thành từ một bộ xử lý nhúng chuyên dụng, phần mềm sụn (Firmware) lưu trong thành phần nhớ ROM, hoặc Flash và các thành phần đầu vào, đầu ra [1, 10].

Kiến trúc của một hệ thống nhúng được tập trung xung quanh một vi điều khiển (microcontroller), còn được gọi là MCU (MicroController Unit) được tích hợp trên một mạch chứa bộ xử lý, RAM bộ nhớ Flash, và các thành phần lõi khác. Trên thị trường cung cấp nhiều sự lựa chọn khác nhau liên quan đến kiến trúc, hãng, khoảng giá, đặc điểm và các nguồn tài nguyên được tích hợp. Những thiết kế thông thường có đặc điểm là không đắt, tiêu thụ năng lượng thấp, nguồn tài nguyên thấp và hệ thống trên một mạch tích hợp đơn, vì những lý do này mà chúng thường được gọi là Hệ thống trên Chip (SoC).

Có nhiều họ vi xử lý được dùng cho các hệ thống nhúng tới nay, song các bộ vi xử lý ARM là phổ biến nhất cho tới nay [1][2]. Theo thống kê của Arm Holdings, đã có trên 184 tỷ vi xử lý dựa trên ARM đã được sử dụng trên toàn thế giới tính đến năm 2020 [2]. Các vi xử lý này được dùng cho các thiết bị di động, thiết bị IoT, thiết bị nhúng, hệ thống điều khiển thời gian thực… Mặt khác, như đã trình bày ở trên, phần mềm nhúng được thiết kế cần phù hợp với kiến trúc phần cứng cụ thể. Do vậy, hệ mật mã trong luận án này được phát triển dựa trên bộ xử lý ARM, là nền tảng phần cứng cho các hệ thống nhúng đang được sử dụng phổ biến cho các thiết bị nhúng hiện nay. Trong phần tiếp theo, luận án trình bày về đặc trưng của kiến trúc ARM làm cơ sở cho việc xây dựng và triển khai các thuật toán của luận án.

1.2. Bộ vi xử lý ARM trong hệ thống nhúng

1.2.1. Kiến trúc ARM (Advanced RISC Machine)

ARM là viết tắt của Advanced RISC machines, thực chất là một kiến trúc bộ xử lý sử dụng tập lệnh rút gọn (RISC = Reduced Instructions Set Computer). Kiến trúc RISC tối ưu hóa các đường dẫn trên mạch, giúp giảm mức tiêu thụ điện năng, tiết kiệm diện

tích, yêu cầu tản nhiệt thấp, cung cấp mức hiệu suất vượt trội, lý tưởng cho các thiết bị nhỏ gọn có tài nguyên hạn chế. ARM được phát triển bởi Arm Holdings, Ltd.

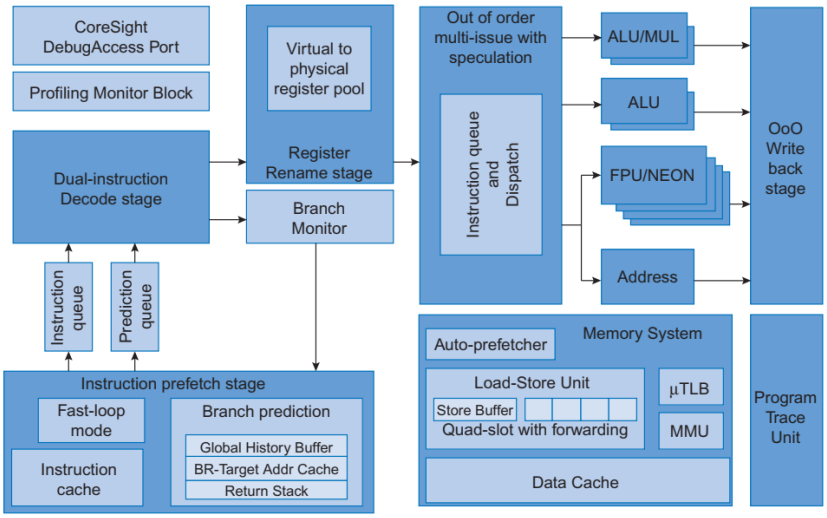
Kiến trúc ARM có một số đặc điểm chính sau [3]:

- Vi xử lý ARM có mười sáu thanh ghi 32 bit (r0-r15).
- Hỗ trợ các chỉ lệnh có điều kiện. Các cờ mã điều kiện N, Z, C, V tương ứng N = Kết quả âm từ ALU; Z = kết quả zero từ ALU; C = nhớ ra từ thao tác ALU; V = tràn từ phép toán ALU.
- Hỗ trợ hai tập chỉ lệnh là ARM và Thumb2.
- Được tích hợp đơn vị xử lý SIMD và VFP (Vector Floating Point) hỗ trợ tập chỉ lệnh NEON và VFPv3 tương ứng.

1.2.2. Các vi xử lý ARM trong thực tế

Các phiên bản ARM đã được phát triển trải rộng từ ARMv1 (năm 1985) đến ARMv8 (năm 2012) [5], và hiện nay mới nhất ARMv9 (năm 2021).

Các bộ vi xử lý cao cấp ARMv7 đã chiếm lĩnh thị trường điện thoại di động và máy tính bảng. Cortex-A9 được sử dụng rộng rãi trong SoC của các điện thoại di động, gồm 2 lõi vi xử lý Cortex-A9, bộ tăng tốc đồ họa, modem 3G/4G, các thiết bị ngoại vi khác. Hình 1.2 thể hiện sơ đồ khối của dòng vi xử lý Cortex-A9. Các giai đoạn pipeline của vi xử lý loại này gồm: giải mã chỉ lệnh, thực hiện đổi tên thanh ghi, thực thi chỉ lệnh ở dạng không theo trật tự.



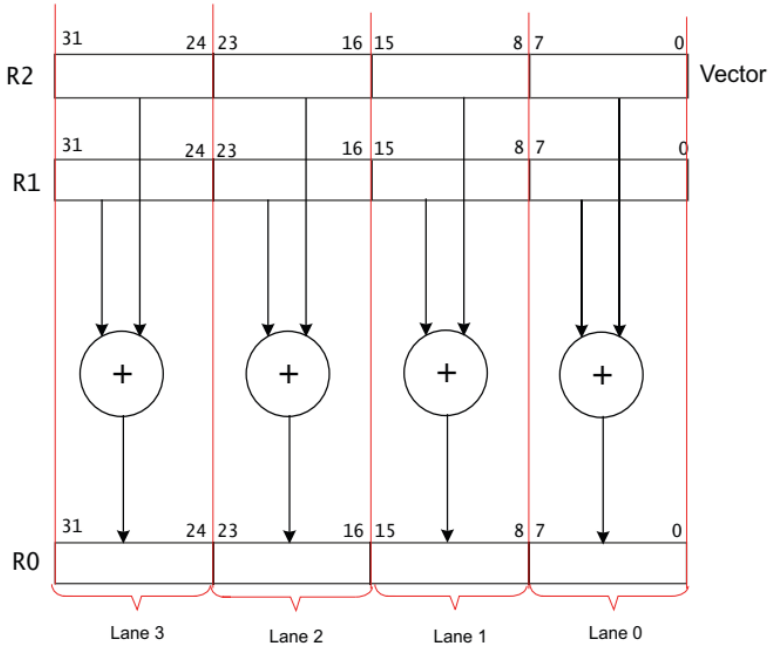
Hình 1.2: Sơ đồ khối của vi xử lý Cortex-A9

Kiến trúc ARMv8 là kiến trúc 64-bit. Các dòng vi xử lý Cortex-A53 và Cortex-A57 có cơ chế pipeline tương tự như Cortex-A7 và -A15 tương ứng.

1.2.3. Kiến trúc mở rộng NEON cho ARM

Kiến trúc tính toán song song đơn lệnh-đa dữ liệu (SIMD = Single Instruction Multiple Data) được đưa vào các bộ vi xử lý ARM từ phiên bản 6 (ARMv6). Từ phiên bản 7 (ARMv7), bộ vi xử lý ARM hỗ trợ engine tích hợp một số thuật toán đã được cứng hóa sẵn được gọi là thành phần NEON. ARM NEON là thành phần mở rộng kiến trúc SIMD cho các bộ vi xử lý ARM họ Cortex-A nhằm tăng cường hỗ trợ cho các ứng dụng đa phương tiện, đặc biệt trong các thiết bị di động. NEON thực chất là thành phần đồng xử lý được tích hợp vào trong các dòng vi xử lý ARM [4], điển hình như Cortex-A8, Cortex-A9, Cortex-A15, Cortex-A53…

NEON hỗ trợ các chỉ lệnh SIMD đặc biệt trên các phần tử vector có cùng kiểu dữ liệu và mỗi chỉ lệnh sẽ thực hiện cùng một phép toán trên tất các các làn (lane) của vector đó [19].

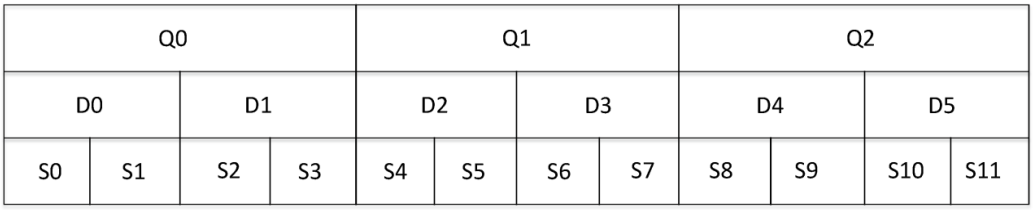


Hình 1.3: Phép toán cộng 8 bit trên 4 làn sử dụng 1 chỉ lệnh SIMD trên ARMv6

Hình 1.3 biểu thị ví dụ một phép cộng 8 bit trên 4 làn sử dụng 1 chỉ lệnh SIM trên ARMv6.

1.2.3.1. NEON trong ARMv7

ARMv7 có 13 thanh ghi 32 bit cho mục đích chung được ký hiệu là từ R0 – R12. Thành phần NEON trong kiến trúc ARMv7 sử dụng các thanh ghi đặc biệt, những thanh ghi này tách biệt với các thanh ghi mục đích chung và được biểu diễn ở dạng thanh ghi D hai từ (64-bit) hoặc thanh ghi Q 128-bit (bốn từ). Hình dưới đây minh họa cấu trúc thanh ghi trong ARMv7.

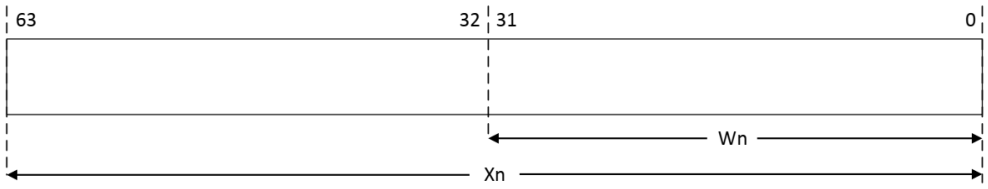


Hình 1.4: Cấu trúc của các thanh ghi NEON trên ARMv7

Đối với ARMv7 hoặc các bộ vi xử lý ARM có phiên thấp hơn v7, có 16 thanh ghi 128-bit được ký hiệu là từ Q0 đến Q15; hoặc ta có thể coi là 32 thanh ghi 64-bit được ký hiệu D0-D31. Thanh ghi Q0 twong ứng với D0-D1, thanh ghi Q1 tương ứng với D2-D3, ... NEON sử dụng các chỉ lệnh để tải/lưu dữ liệu và xử lý dữ liệu trong những thanh ghi này. Các chỉ lệnh NEON có khả năng thực hiện truy cập bộ nhớ, sao chép dữ liệu giữi thanh ghi NEON và thanh ghi mục đích chung. NEON còn có thể thực hiện chuyển đổi kiểu dữ liệu và xử lý dữ liệu trong các thanh ghi D hoặc Q.

1.2.3.2. NEON trong ARMv8

ARMv8 có kiến trúc 64-bit, kiến trúc này có tập chỉ lệnh mới (AARCH64). Kiến trúc ARMv8 có 31 thanh ghi mục đích chung (X0-X30), đây là những thanh ghi 64-bit và nửa thấp của mỗi thanh ghi này được ký hiện là W0-W30, được biểu diễn như hình sau:



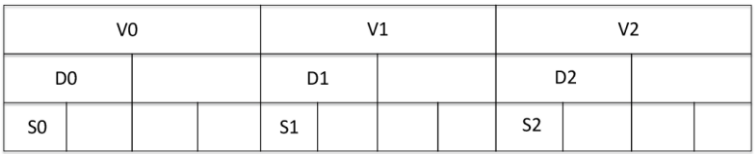
Hình 1.5: Các thanh ghi trong kiến trúc ARMv8

Sự khác nhau chính trong các chỉ lệnh giữa ARMv7 và ARMv8 được thể hiện trong bảng sau:

Bảng 1.1: Sự khác nhau giữa chỉ lệnh trong ARMv7 và ARMv8

Instructions Type	Architecture	
	A32	A64
Arithmetic instructions	*ADD Rd, Rn, #9*	*ADD Wd, Wn, #9*
	ADDS Rd, Rm, LSL#2	*ADDS Wd, Wn, LSL #2*
	MUL Rd, Rn, Rm	*MUL Wd, Wn, Wm*
	MOV Rd, #imm	*MOV Wd, #imm*
Load/Store	*PUSH r0 – r1*	*STP x0, x1, [sp, #16]*
	POP r0 – r1	*LDP x0, x1, [sp], #16*
	LDMIA r0!, r1, r2	*LDP x0, x1, [x0], #16*
	STMIA r0!, r1, r2	*LDP x0, x1, [x0], #16*
Subroutine return	*MOV PC, LR*	*RET*
	POP PC	
	BX LR	
Exception return	*SUBS PC, LR, #4* *MOVS PC, LR*	*ERET*

Kiến trúc ARMv8 có tập các thanh ghi NEON riêng biệt khác với tập các thanh ghi ARMv7. NEON mở rộng thanh ghi trong ARMv8 khác với tập thanh ghi chính trong ARM và chứa tập các thanh ghi vector có độ rộng 32 x 128 bit hoặc các thanh ghi vector có độ rộng 32 x 64 bit chính là 64-bit thấp của mỗi thanhghi 128-bit. Kiến trúc ARMv8 hỗ trợ tập chỉ lệnh giống như AArch32 sử dụng cách lập trình gọi hàm nội tại (intrinsic), điều này tương đương với ARMv7 và AArch64 mà hỗ trợ trên chính thanh ghi NEON này. Hình 1.6 minh họa cấu trúc của thanh ghi vector trên ARMv8. *Vn* biểu diễn thanh ghi vector 128-bit, *Dn* có thể được sử dụng để truy nhập tới 64-bit thấp của thanh ghi *Vn* và *Sn* biểu diễn 32-bit thấp nhất của *Dn*.

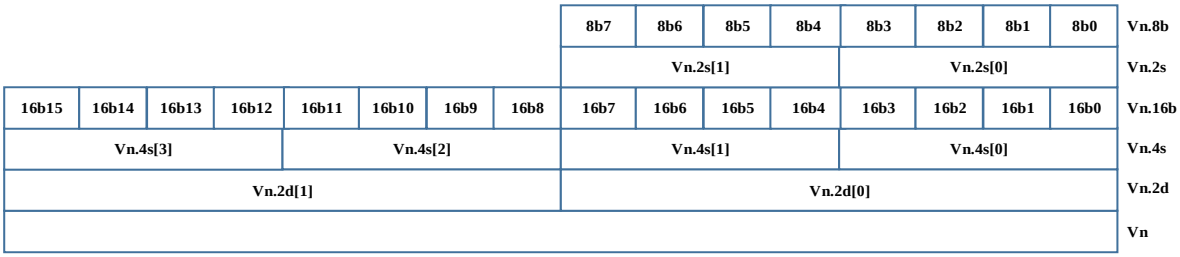


Hình 1.6: Cấu trúc thanh ghi NEON trong ARMv8

Bảng 1.2: Quy ước tên và độ rộng vector trong ARMv8

Shape (bits x lanes)	8b x 8	8b x 16	16b x 4	16b x 8	32b x 2	32b x 4	64b x 1	64b x 2
Tên	Vn.8b	Vn.16b	Vn.4H	Vn.8H	Vn.2S	Vn.4S	Vn.1d	Vn.2d

Bảng trên giải thích về các thanh ghi vector được sử dụng trong ARMv8. Những vector này có thể có độ rộng 128-bit với hai hoặc nhiều phần tử; hoặc là thanh ghi có độ rộng 64-bit với một hoặc nhiều phần tử.



Hình 1.7: Truy nhập trên thanh ghi ARMv8

Tập chỉ lệnh NEON của kiến trúc ARMv8 Aarch64 khác so với kiến trúc ARMv7, dạng chỉ lệnh như sau:

$\{< prefix >\} < op > \{< suffix >\} Vd. < T >, V n. < T >, V m. < T >$

Trong đó các tham số được thể hiện trong bảng sau

Bảng 1.3: Chỉ lệnh NEON trong ARMv7 và ARMv8

Parameters	meaning	Definition
<prefix>	prefix	S/U/F/P represents signed/unsigned/bool data type
<op>	Operation	Such as *ADD*, *SUB*.
<suffix>	Suffix	P is "pairwise" operations, such as *ADDP*.
		V is the new reduction operations, such as *FMAXV*.
		2 is new widening/narrowing. Such as *ADDHN2*, *SADDL2*.
<T>	Data Type	Such as 8B/16B/4H/8H/2S/4S/2D.
		B represents byte (8-bit).
		H represents half-word (16-bit).
		S represents word (32-bit).
		D represents a double-word (64-bit).

1.2.4. Lập trình NEON trên kiến trúc ARM

GNU GCC cung cấp các tùy chọn vector cho mã C để sinh ra mã NEON. Trình biên dịch GCC phải tối ưu để sinh ra mã thực thi. Để cải thiện hiệu suất thực hiện của hệ thống dựa trên NEON, có hai phương pháp để viết mã lập trình là sử dụng chỉ lệnh nội tại NEON (intrinsic) hoặc lập trình trực tiếp bằng ngôn ngữ Assembly [19]. Các bước cơ bản để viết đoạn mã sử dụng tính toán trong NEON là: tải dữ liệu, thực hiện tính toán và lưu dữ liệu.

1.2.4.1. Lập trình sử dụng phương pháp NEON intrinsic

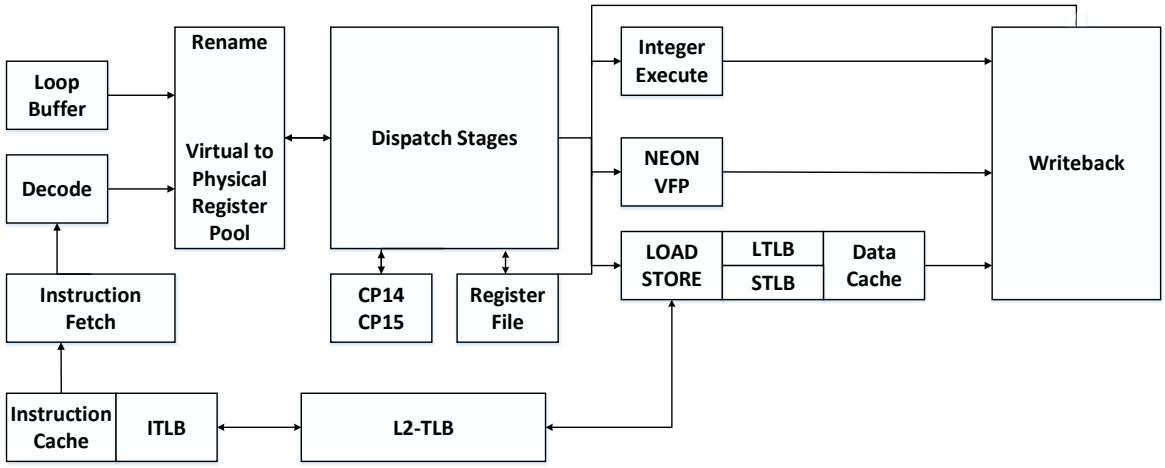
Ngôn ngữ NEON nội tại cung cấp các phương pháp để các hàm tựa C tương tác với các phép toán trong NEON, các phép tính toán này được sử dụng để gọi các chỉ lệnh NEON trong mã nguồn của ngôn ngữ C. Trình biên dịch có thể sinh ra các chỉ lệnh NEON thích hợp dựa trên các tệp đối tượng và rồi mã thực thi chạy trên cả kiến trúc

ARMv7-A hoặc ARMv8-A. Tất cả các hàm nội tại được định nghĩa trong "arm_neon.h".

1.2.4.2. Lập trình NEON trực tiếp bằng ngôn ngữ Assembly

Lập trình ngôn ngữ bậc thấp (assembly) trên NEON có thể được viết theo kiểu "inline assembly" sử dụng khai báo "asm volatile();" hoặc viết trực tiếp bằng ngôn ngữ bậc thấp trong tệp có đuôi .s. Các chỉ lệnh NEON là khác nhau như đã trình bày ở mục trên.

Thành phần NEON có khả năng chuyển dữ liệu bộ nhớ có độ rộng 128-bit. Thao tác đọc và lưu dữ liệu sử dụng cùng một đơn vị Load/Store trong mô hình pipeline của kiến trúc ARM Cortex-A như hình 1.8. Tập chỉ lệnh NEON cung cấp chỉ lệnh VLD để tải dữ liệu và VST để lưu dữ liệu.



Hình 1.8: Kiến trúc Pipeline trên vi xử lý ARM

1.3. An toàn và bảo mật thông tin trên hệ thống nhúng

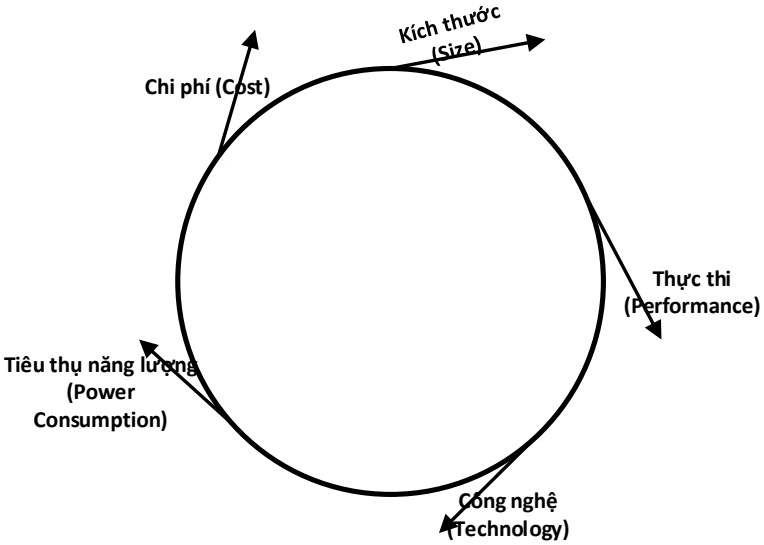
1.3.1 Các thách thức khi xây dựng hệ thống nhúng

Các thách thức trong xây dựng hệ thống nhúng:

Người thiết kế hệ thống nhúng phải xây dựng một triển khai để đáp ứng các chức năng mong muốn, nhưng một thách thức khó khăn là xây dựng một triển khai đảm bảo tối ưu hóa đồng thời nhiều thuộc tính trong thiết kế hệ thống nhúng.

Tất nhiên, việc thiết kế hệ thống nhúng phải xây dựng một triển khai đáp ứng các chức năng mong muốn, nhưng một thách thức khó khăn là xây dựng một triển khai

thỏa mãn tối ưu hóa đồng thời nhiều thuộc tính trong thiết kế. Hình dưới đây thể hiện một vài tham số cần tính toán khi thiết kế hệ thống nhúng.



Hình 1.9: Các thách thức trong thiết kế hệ thống nhúng.

Những tính chất của thiết kế trên bị ràng buộc và đối nghịch nhau: Cải thiện một thuộc tính này lại thường dẫn đến trả giá một thuộc tính khác.

- Thực thi (Performance): Thực thi của hệ thống nhúng không chỉ là tốc độ của vi xử lý, vấn đề quan trong thực sự là hiệu suất thời gian thực. Chẳng hạn như hệ thống sẽ phản ứng nhanh như thế nào với các sự kiện cụ thể. Một hệ thống nhúng thường chạy hệ điều hành xử lý thời gian thực (RTOS) đảm bảo một phản ứng xảy ra trong một cửa sổ thời gian xác định. Đây là một đặc tính khác so với yêu cầu PC thông thường.
- Tiêu thụ năng lượng: Tiêu thụ năng lượng thấp là một tham số quan trọng cho các hệ thống nhúng. So sánh với máy tính để bàn PC hay máy tính xách tay, thì nhiều hệ thống nhúng hoạt động dựa trên nguồn PIN. Một hệ thống nhúng thường có yêu cầu đối nghịch nhau: Yêu cầu năng lượng tiêu thụ thấp nhưng hiệu năng cao. Để giảm tiêu thụ năng lượng, nhiều hệ thống nhúng được cấp nguồn theo chế độ khác nhau: Chế độ hoạt động, chế độ chờ, chế độ ngủ…
- Ngoài ra còn nhiều thách thức khác trong xây dựng hệ thống nhúng: kích thước, chi phí thiết kế, công nghệ sử dụng, bộ nhớ (ROM, RAM, Flash)..

Nhận xét :

Tài nguyên của hệ thống nhúng bao gồm: bộ vi xử lý, bộ nhớ trong (RAM, ROM), bộ nhớ Flash, nguồn pin… Trên nền thiết bị nhúng, việc triển khai các giải pháp bảo đảm an toàn, bảo mật còn gặp nhiều hạn chế do phải đối mặt với nhiều thách thức: thực thi của vi xử lý, hạn chế bộ nhớ, nguồn PIN, đáp ứng xử lý thời gian thực… Do vậy, việc thể triển khai các mô hình mật mã đã phát triển cho các máy tính PC cần phải thay đổi cho phù hợp với hệ thống nhúng. Ngay cả hệ mật mã đường cong Elliptic đã được cho là hiệu quả thực thi tốt cũng khó để triển khai khả thi trong môi trường thiếu bộ nhớ, hiệu năng xử lý CPU thấp và yêu cầu phản ứng thời gian thực. Hoạt động của hàm mật mã tiêu tốn năng lượng cũng có thể làm cạn pin của thiết bị trước khi đạt kết quả mã hóa/giải mã thành công.

1.3.2 Mật mã trên hệ thống nhúng

Để bảo đảm an toàn và bảo mật dữ liệu cho hệ thống nhúng, các thuật toán mã hóa sử dụng mã khối và hệ mã hóa công khai cũng được sử dụng, song cần được phát triển phù hợp trong môi trường thiết bị nhúng.

- Mã khối: Nhiều nghiên cứu trên thế giới đã tập chung nghiên cứu về mã khối hạng nhẹ (Lightweight Block Ciphers), và tối ưu những thuật toán này cho những thiết bị nhúng.
- Mã khoá công khai là một công cụ không thể thiếu để thực hiện các tính năng an toàn, bảo mật khác nhau cần thiết trên các nền tảng nhúng như trao đổi khóa, ký số, mã hóa dữ liệu…Tuy nhiên, việc tính toán trên hệ mật khóa công khai là rất tốn kém, bởi vậy việc cài đặt các thuật toán thực hiện các hệ mật RSA, ECC hiệu quả là một thách thức đối với bất kỳ một hệ thống, thiết bị mà có nguồn tài nguyên hạn chế.

ECC là một hệ mật phổ biến được cài đặt trên các thiết bị tính toán nhúng do kích thước khóa nhỏ hơn nhiều so với các hệ mật khóa công khai khác (ví dụ độ an toàn tương đương 128-bit, đối với hệ mật RSA cần khóa có độ dài 3072-bit, trong khi đó ECC chỉ cần khóa có độ dài cỡ 256-bit). Bảng dưới đây đưa ra một số so sánh giữa hệ mật RSA và hệ mật đường cong elliptic theo khuyến cáo của NIST [53].

Bảng 1.4: Độ an toàn theo kích cỡ khóa của ECC và RSA.

Năm	Độ an toàn (tính theo bit)	Kích cỡ khóa ECC (tính theo bit)	Kích cỡ khóa RSA (tính theo bit)
2016–2030	112	224	2,048
> 2030	128	256	3,072
>> 2030	192	384	7,680
>>> 2030	256	512	15,360

Tương tự như các hệ mật khóa công khai, thì việc tính toán trên ECC cũng khá phức tạp và tốn kém liên qua đến một số thuật toán như: cộng điểm, nhân đôi điểm và thời gian tính toán hoàn toàn cũng bị chi phối bởi các phép nhân modulo trong trường hữu hạn.

Mật mã ECC đã có những tính chất ưu việt hơn về yêu cầu bộ nhớ, năng lực tính toán so với các hệ thống mật mã khác. Tuy nhiên, đối với các thiết bị nhúng có nền tảng phần cứng và tài nguyên hạn chế, việc nghiên cứu các giải pháp nhằm triển khai hiệu quả hệ thống mật mã ECC trên môi trường thực tế nói chung và trên nền các thiết bị tính toán nhúng nói riêng vẫn còn có những hạn chế và vẫn đang là chủ đề được quan tâm trong cộng đồng nghiên cứu cũng như trong thực tế triển khai trên thế giới. Nghiên cứu đưa ra các thuật toán an toàn và hiệu quả cho ECC trong các hệ thống nhúng, cụ thể là các thiết bị nhúng có tài nguyên hạn chế vẫn đang là vấn đề có nhiều thách thức.

Trong phần tiếp theo, luận án trình bày khái quát về hệ mật đường cong Elliptic làm cơ sở cho nghiên cứu đề xuất các giải thuật mới trong luận án.

1.4. Hệ mật đường cong Elliptic và ứng dụng

Hệ mật đường cong Elliptic (ECC - Ellicpic Curve Cryptography) là một hệ mật phổ biến được cài đặt trên các thiết bị tính toán nhúng do kích thước khóa nhỏ hơn nhiều so với các hệ mật khác [6]. Trong phần này, luận án tóm lược những đặc điểm cơ bản của hệ mật ECC.

1.4.1. Cách biểu diễn điểm trên trường hữu hạn

Có ba dạng tạo độ cơ bản thường được sử dụng. Đó là tọa độ quan hệ (Affine coordinate), tọa độ chiếu (Projective coordinate) và tọa độ nén [7].

1.4.1.1. Tọa độ quan hệ (tọa độ Affine)

Một điểm hữu hạn P trên (E) được xác định bởi hai phần tử x, y trong GF(p) thoả mãn phương trình của đường cong:

(E): $Y^2 = X^3 + aX + \mathrm{b};\ \mathrm{a}, \mathrm{b} \in F_q;\ 4\mathrm{a}^3 + 27b^2 \neq 0 \pmod{\mathrm{p}}$

Chúng được gọi là các tọa độ quan hệ của điểm *P*. Điểm ở vô cực ∞ không có tọa độ Affine. Với mục đích phục vụ cho việc tính toán người ta thường biểu diễn ∞ bởi một cặp các hệ số (x, y) không nằm trên (E).

LUẬT NHÓM

Gọi E là một đường cong elliptic được định nghĩa trên trường Φ_p *bởi phương trình* $y^2 = x^3 + Ax + B$ *với* $A, B \in \Phi_p$ *và* $4A^3 + 27B^2 \neq 0 \bmod p$. *Gọi* $P_1 = (x_1, y_1)$ *và* $P_2 = (x_2, y_2)$ *là các điểm trên E với* $P_1, P_2 \neq \infty$. *Định nghĩa* $P_1 + P_2 = P_3 = (x_3, y_3)$ *như sau:*

1. *Nếu* $x_1 \neq x_2$ *thì*

   $x_3 = m^2 - x_1 - x_2,\ y_3 = m(x_1 - x_3) - y_1,$ *ở đây* $m = \frac{y_2 - y_1}{x_2 - x_1}$. *(1.1)*

2. *Nếu* $x_1 = x_2$ *nhưng* $y_1 \neq y_2$ *thì* $P_1 + P_2 = \infty$.

3. *Nếu* $P_1 = P_2$ *và* $y_1 \neq 0$ *thì*

   $x_3 = m^2 - 2x_1,\ \ y_3 = m(x_1 - x_3) - y_1,$ *ở đây* $m = \frac{3x_1^2 + A}{2y_1}$. *(1.2)*

4. *Nếu* $P_1 = P_2$ *và* $y_1 = 0$ *thì* $P_1 + P_2 = \infty$.

1.4.1.2. Tọa độ chiếu (tọa độ Projective)

Để tránh phép tính chia trong trường, người ta đưa ra việc biểu diễn điểm theo tọa độ chiếu (tọa độ ở dạng phân số). Có hai loại hệ tọa độ chiếu cơ bản là, tọa độ chiếu chuẩn và tọa độ chiếu Jacobian.

Trong hệ tọa độ chiếu chuẩn, một điểm được biểu diễn dưới dạng (X, Y, Z) với Z ≠ 0, tương đương với tọa độ Affine là (X/Z, Y/Z). Phương trình đường cong Elliptic tọa độ chiếu chuẩn sẽ có dạng:

$$Y^2Z = X^3 + a\mathrm{X}\mathrm{Z}^2 + bZ^3$$

Trong hệ tọa độ chiếu Jacobian, một điểm (X, Y, Z) trên tọa độ chiếu Jacobian sẽ tương đương với điểm (X/Z^2, Y/Z^3) trong hệ tọa độ Affine. Phương trình đường cong (E) có dạng:

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

Chúng ta có thể biến đổi một điểm có tọa độ chiếu Jacobian sang tọa độ Affine. Công thức biến đổi như sau:

$$(X, Y, Z) \Rightarrow (x = \frac{X}{Z^2}, y = \frac{Y}{Z^3})$$

Ngược lại, chúng ta có thể đổi một điểm ở tọa độ Affine sang tọa độ chiếu Jacobian. Công thức biến đổi như sau:

$$(x, y) \Rightarrow (X = x, Y = y, Z = 1)$$

Trong các phép tính cộng và nhân đôi điểm ở hệ tọa độ chiếu, không cần sử dụng phép tính nghịch đảo (phép chia). Công thức xác định phép tính cộng và nhân đôi hai điểm đối với tọa độ chiếu có thể nhận được bằng cách chuyển các điểm sang tọa độ Affine, sau đó sử dụng công thức cộng và nhân đôi các điểm ở hệ tọa độ Affine, cuối cùng bỏ đi mẫu số chúng ta sẽ nhận được công thức tính cho hệ tọa độ chiếu.

Phép cộng

Cho 2 điểm $P(x_1 : y_1 : z_1)$, $Q(x_2 : y_2 : z_2)$ thuộc đường cong E với $P, Q \neq \infty$, $P \neq \pm Q$, ta định nghĩa điểm $P + Q = (x_3 : y_3 : z_3)$ như sau:

Đặt

$$a = x_1 z_2^2, \quad b = x_2 z_1^2, \quad c = y_1 z_2^3$$
$$d = y_2 z_1^3, \quad e = b - a, \quad f = d - c$$

Thì

$$x_3 = -e^3 - 2a \cdot e + f^2$$
$$y_3 = c \cdot e^3 + f(a \cdot e^2 - x_3)$$
$$z_3 = z_1 \cdot z_2 \cdot e$$

Phép nhân đôi

Cho 2 điểm $P(x_1 : y_1 : z_1)$ thuộc đường cong E với $P \neq \infty$, ta định nghĩa điểm $[2]P = (x_3 : y_3 : z_3)$ như sau:

Đặt

$$w1 = 4x_1y_1^2, w2 = 3x_1^2 + A \cdot z_1^4$$

Thì

$$x_3 = -2w1 + w2^2$$

$$y_3 = -8y_1^4 + w3(w1 - x_3)$$

$$z_3 = 2y_1z_1$$

Bảng 1.5: Số lượng các phép toán được thực hiện để cộng và nhân đôi điểm

Hệ tọa độ	Phép cộng thông thường	Phép nhân đôi
Afin	$1I, 2M$	$1I, 2M$
Hình chiếu chuẩn $(X/Z, Y/Z)$	$13M$	$7M$
Hình chiếu Jacobi $(X/Z^2, Y/Z^3)$	$14M$	$5M$

Nhận xét: Trong bảng trên, thực hiện mật mã đường cong Elliptic chính là thực hiện phép nhân điểm và đây cũng là phép tính tiêu tốn thời gian và tài nguyên nhất trong ECC, bản chất của việc thực hiện phép nhân điểm chính là thực hiện các phép toán của biểu thức (1.1) và (1.2), trong đó phải thực hiện các phép toán trên trường cơ sở như: nhân, chia, nghịch đảo, cộng, trừ các số nguyên

1.4.2 Ứng dụng của hệ mật dựa trên đường cong Elliptic

Có nhiều ứng dụng được phát triển trên nền mật mã đường cong elliptic, trong số đó phải kể đến là thuật toán chữ ký số ECDSA, thuật toán trao đổi khóa ECDH, ECMQV, ECHMQV, thuật toán mã hóa ECIES... Các ứng dụng này đều xây dựng trên nền ECC với phép tính mật mã cơ bản là phép nhân vô hướng của một điểm trên đường cong với một số nguyên.

1.4.2.1. Thuật toán trao đổi khóa ECDH

Thuật toán trao đổi khóa Diffie-Hellman trên đường cong Elliptic (ECDH) [7] tương tự thuật toán trao đổi khóa Diffie-Hellman thông thường trên trường hữu hạn. ECDH cũng dựa vào nguyên lý của bài toán logarit rời rạc nhưng áp dụng trên nhóm các điểm của đường cong elliptic [8]. Thuật toán này dùng để thiết lập một hoặc nhiều khóa chung giữa hai đối tác A (Alice) và B (Bob). Quá trình trao đổi khóa dùng ECDH được thực hiện như sau:

1. Alice và Bob thoả thuận cùng sử dụng một đường cong Elliptic (E) trên trường F_q sao bài toán logarithm rời rạc là khó trong $E(F_q)$. Họ cũng thống nhất một điểm công khai P sao cho nhóm con sinh bởi P có cấp là một số nguyên tố lớn.
2. Alice chọn một số bí mật a và tính khoá công khai của mình là $P_a = a.P$ và gửi P_a cho Bob.
3. Bob cũng chọn cho mình một khoá bí mật b và tính $P_b = b.P$ và gửi P_b cho Alice
4. Alice tính được khoá chung là $a.P_b = ab.P$
5. Bob tính được khoá chung là $b.P_a = ab.P$

1.4.2.2 Thuật toán ký số ECDSA

Ta tóm tắt thuật toán như sau:

Alice muốn kí một văn bản *m,* với *m* được xem như là một số nguyên (thực tế, thông thường cô ta kí giá trị băm của văn bản). Alice chọn một đường cong elliptic trên một trường hữu hạn $\mathbf{F}_q$ sao cho $\#E(\mathbf{F}_q) = fr$, ở đây *r* là một số nguyên tố lớn và *f* là một số nguyên nhỏ, thông thường là 1, 2, hoặc 4 (*f* nên là một số nhỏ để thuật toán là hiệu quả). Cô ta chọn một điểm cơ sở *G* trong $E(\mathbf{F}_q)$ cấp *r*. Cuối cùng, Alice chọn một số nguyên bí mật *a* và tính $Q = aG$. Alice tạo công khai các thông tin sau:

$\mathbf{F}_q$, *E*, *r*, *G*, *Q*.

(Không cần phải giữ bí mật *f*; nó có thể suy ra được từ *q* và *r* sử dụng định lý Hasse theo kĩ thuật trong các Ví dụ 4.6 và 4.7). Để kí văn bản *m* Alice thực hiện như sau :

1. Chọn một số nguyên ngẫu nhiên *k* với $1 \le k < r$ và tính $R = kG = (x, y)$.
2. Tính $s = k-1(m + ax) \pmod r$.

Văn bản được kí là

(*m*, *R*, *s*).

Để kiểm tra chữ kí, Bob thực hiện như sau.

1. Tính $u_1 = s^{-1}m \pmod r$ và $u_2 = s^{-1}x \pmod r$.
2. Tính $V = u_1G + u_2Q$.
3. Khẳng định chữ kí là có hiệu lực nếu $V = R$.

Nếu văn bản được kí một cách chính xác, phương trình kiểm tra sẽ đúng:

$V = u_1G + u_2Q = s^{-1}mG + s^{-1}xQ = s^{-1}(mG + xaG) = kG = R.$

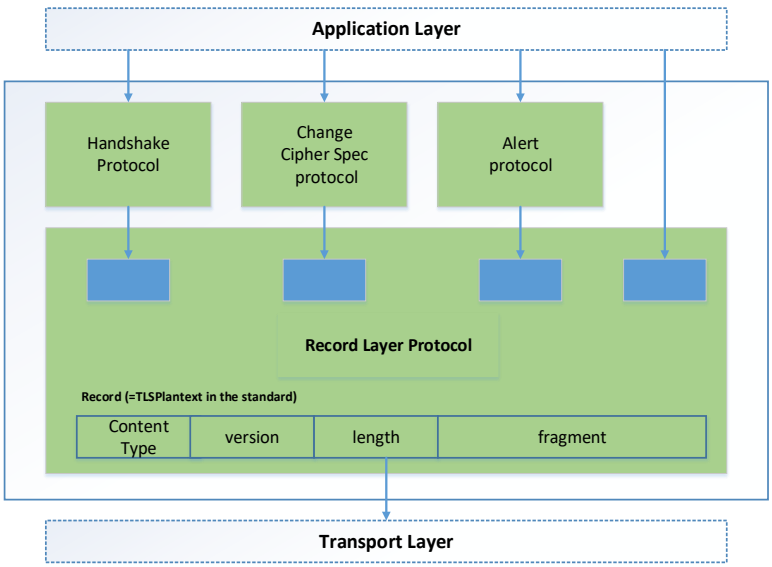
1.4.3 Ứng dụng ECDH và ECDSA trong bảo mật truyền dữ liệu trên thiết bị nhúng.

IKEv2 [57] và TLS [58] là hai giao thức trao đổi khóa phổ biến hiện nay mà sử dụng giao thức trao đổi khóa ECDH và giao thức ký số ECDSA. IKEv2 là một giao thức trao đổi khóa cho IPSec VPN, trong khi đó TLS được sử dụng làm giao thức trao đổi khóa cho OpenVPN.

1.4.3.1 Giao thức TLS

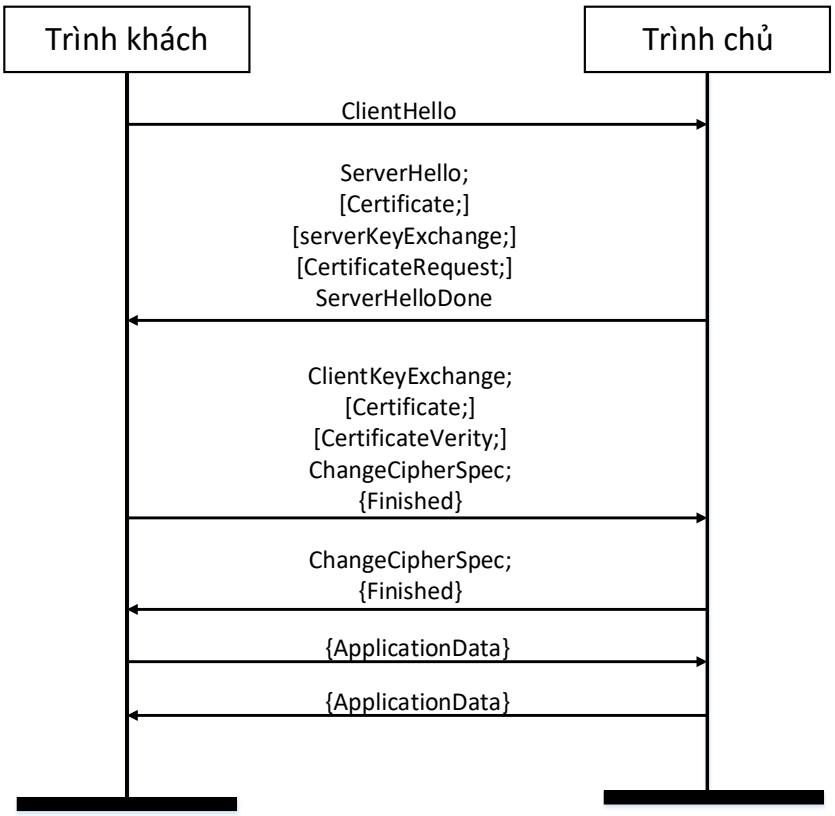
TLS là giao thức cung cấp kết nối an toàn trên mạng Internet, nó gắn liền với giao thức HTTPS. Để thiết lập một kết nối an toàn, có ba giao thức con được sử dụng trong TLS:

- Giao thức bắt tay (Handshake): Giao thức này được sử dụng để thiết lập khóa phiên, các tham số, xác thực máy chủ và trình khách.
- Giao thức ChangeCipherSpec: Gồm chỉ một thông báo, nó được sử dụng để chỉ thời điểm bắt đầu sử dụng khóa phiên
- Giao thức cảnh báo (Alert): Được sử dụng để gửi các cảnh báo.



Hình 1.10: Các thành phần trong giao thức TLS

Các bước bắt tay của giao thức handshake như sau:

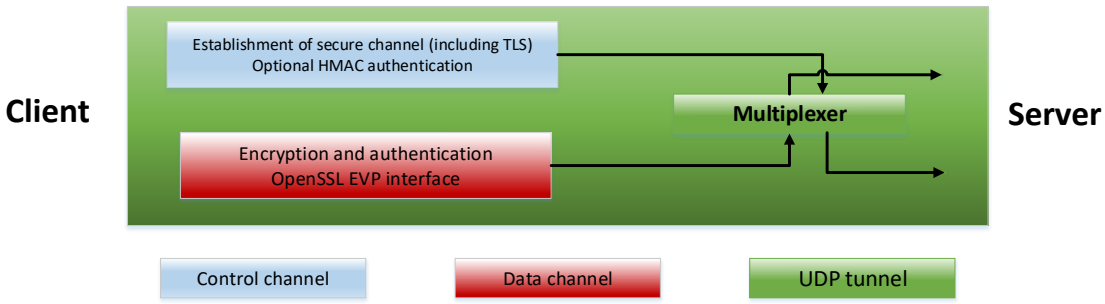


Hình 1.11: Các bước bắt tay trong giao thức TLS

Trong các bước bắt tay ở trên, thì thành phần ServerKeyExchange và ClientKeyExchage được sử dụng để trao đổi khóa, trong bước này có thể sử dụng đến giao thức ECDH. Các bước ServerKeyExchange và CertifcateVerify có thể chứa chữ ký số để xác thực máy chủ và trình khách. Các bước này có thể sử dụng đến giao thức ECDSA.

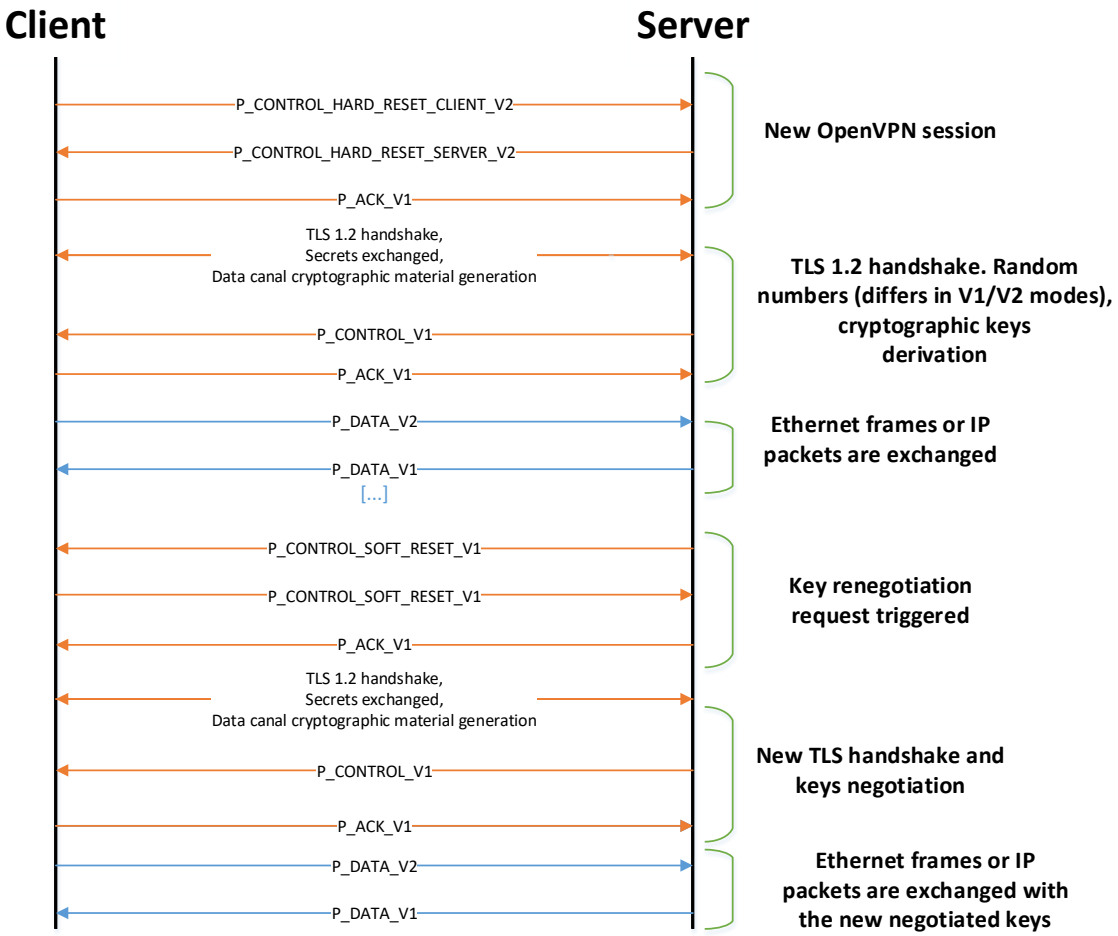
1.4.3.2 Giao thức OpenVPN

OpenVPN [59] là kiểu SSL VPN, nó cho phép thiết lập các đường hầm trên nền giao thức TCP hoặc UDP. OpenVPN cung cấp xác thực, mã hóa, bảo đảm tính toàn vẹn của gói tin trong nó. OpenVPN sử dụng giao thức TLS để thực giao đoạn trao đổi khóa.



Hình 1.12: Kiến trúc của giao thức OpenVPN

Các bước trao đổi khóa trong OpenVPN xem hình 1.13.

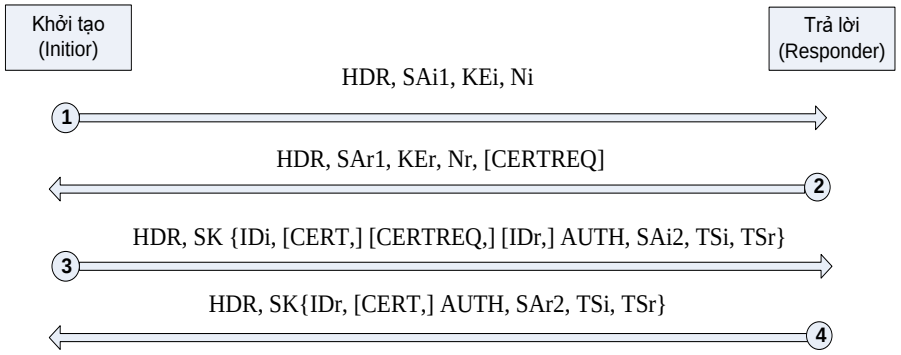


Hình 1.13: Các bước trao đổi khóa trong giao thức OpenVPN

1.4.3.3 Giao thức IKEv2

Giao thức trao đổi khóa IKEv2 [57] được sử dụng để thỏa thuận khóa trong bộ giao thức IPSec. Giao thức này gồm hai giai đoạn:

- Giai đoạn I: Thực hiện đổi khóa và xác thực thực thể (IKE_SA_INIT và IKE_AUTH)
- Giai đoạn II: Thực hiện tạo các khóa con con tầng giao thức IPSec (CREATE_CHILD_SA)



Hình 1.14: Các bước trong giai đoạn 1 của giao thức IKEv2.

Việc trao đổi khóa được thực hiện ở trong bước 1 và 2 của sơ đồ trên, trong đó:

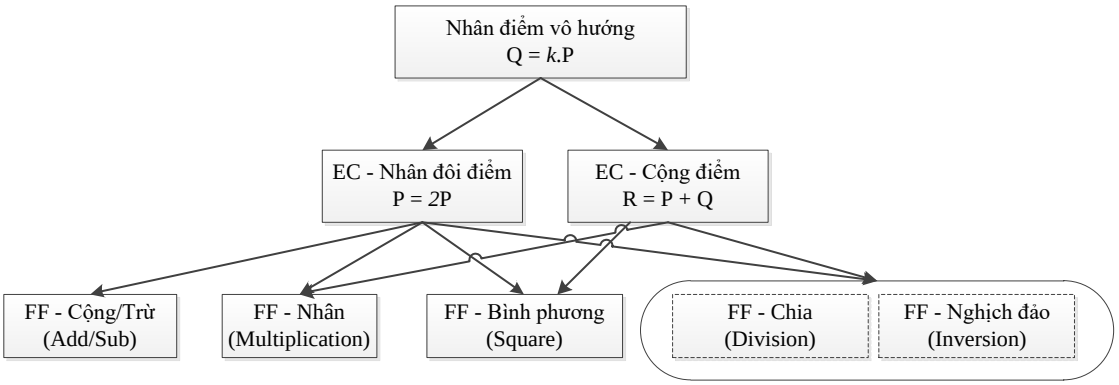
- *KEi*: Giá trị DH của bên khởi tạo
- *KEr*: Giá trị DH của bên trả lời

Xác thực đối được thực hiện trong bước 3, 4 (thành phần AUTH). Có ba cơ chế xác thực: xác thực bằng khoá chia sẻ trước, xác thực dùng chữ ký số, xác thực mở rộng. Khi xác thực bằng chữ ký số thì có thể sử dụng giao thức ECDSA.

1.5. Hiệu quả sử dụng mật mã đường cong Elliptic trên thiết bị nhúng và các nghiên cứu liên quan

1.5.1. Sử dụng mật mã đường cong Elliptic trên thiết bị nhúng

An toàn của hệ mật đường cong Elliptic dựa trên bài toán khó đó là tìm logarit rời rạc trên đó (ECDLP – Elliptic Curve Discrete Logarithm Problem), tức là cho một đường cong E trên trường hữu hạn F_q, một điểm $P \in E\left(F_q\right)$ có cấp là n và một điểm $Q \in E\left(F_q\right)$, tìm một số nguyên $k \in \left[0, n-1\right]$ sao cho Q = k.P . Thuật toán nhân điểm được thực hiện bởi phép cộng và phép nhân đôi, ví dụ 7P=2((2P)+P)+P, đối với ECC ta xem xét những điểm này thuộc trong vài trường hữu hạn. Phép tính Q=kP được gọi là phép nhân điểm vô hướng, đây phép tính này tiêu thụ thời gian thực hiện trong cài đặt ECC. Phân cấp của các phép toán số học cơ bản trên EC như hình dưới đây:



Hình 1.15: Cấu trúc của phép toán trên EC

Như hình trên các phép toán trên EC được thực hiện từ sự kết hợp của các phép toán cơ bản trên trường hữu hạn (finite field – FF).

Trước khi cài đặt một hệ thống ECC, một số lựa chọn cần phải được thực hiện, bao gồm lựa chọn các tham số miền đường cong elliptic (trường hữu hạn cơ bản, biểu diễn trường, đường cong elliptic) và các thuật toán về số học trường, số học trên đường cong elliptic và số học giao thức. Các lựa chọn này có thể được thực hiện dựa trên các tính toán cân nhắc về vấn đề an toàn, nền tảng ứng dụng (phần mềm, phần sụn hoặc phần cứng), các ràng buộc của môi trường máy tính cụ thể (ví dụ: tốc độ xử lý, kích thước mã (ROM), kích thước bộ nhớ (RAM), số cổng, mức tiêu thụ điện năng và các ràng buộc của môi trường truyền thông cụ thể (ví dụ: băng thông, thời gian phản hồi). Việc quyết định một tập hợp các sự lựa chọn "tốt nhất" duy nhất là rất khó, nếu không muốn nói là không thể thực hiện được, ví dụ: các lựa chọn tối ưu đối với một ứng dụng trên PC có thể hoàn toàn khác với lựa chọn tối ưu đối với ứng dụng trên thẻ thông minh hay trên thiết bị nhúng.

1.5.2. Các nghiên cứu liên quan

Các nghiên cứu liên quan có thể được phân chia thành 3 nhóm chính, được trình bày cụ thể trong phần tiếp theo.

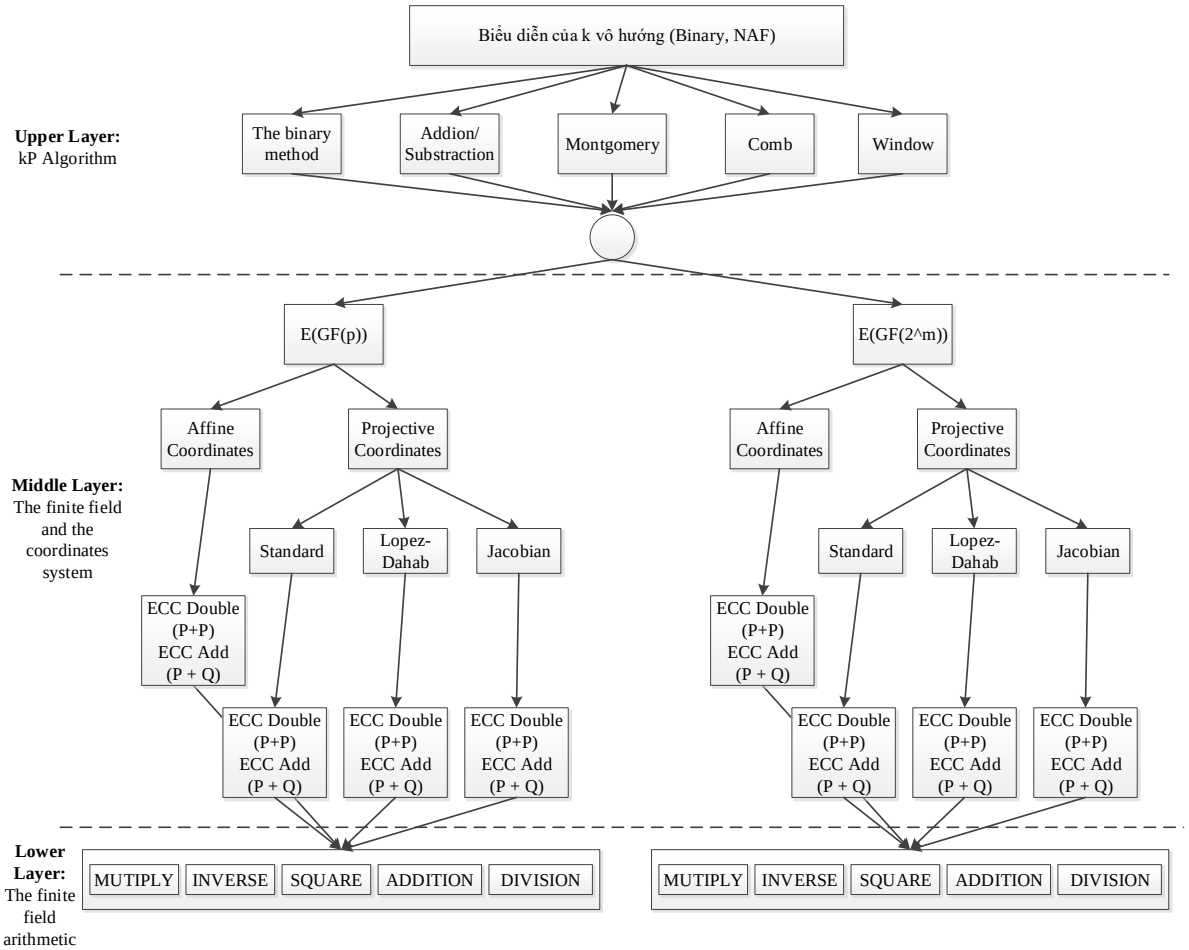
1.5.2.1. Cải tiến thuật toán và lựa chọn tham số phù hợp

a) Một số cải tiến thuật toán trên ECC

Trên thế giới có nhiều nghiên cứu về lý thuyết để tối ưu và đánh giá hiệu quả thực hiện các thuật toán nhân điểm như công trình của các tác giả Darrel Hankerson, Alfred

Menezes, Scott Vanstone (2004) [7]; Praful Kumar Singh, Mrityunjay Kumar Choudhary (2013) và [47] đã nghiên cứu và so sánh đánh giá hiệu quả thực hiện một số thuật toán nhân điểm như: nhân điểm theo phương pháp nhị phân, nhân điểm theo phương pháp triển khai một số nguyên dương theo thuật toán NAF… Các cải tiến và đánh giá thuật toán nhân trên môi trường nhúng được nghiên cứu trong [20].

Phép nhân điểm Q = *k*P có thể được thực hiện theo các phương pháp khác nhau. Hình dưới đây tóm tắt các phương pháp khác nhau khi thực hiện phép nhân điểm vô hướng, các thuật toán khác nhau được sử dụng để thực hiện phép nhân. Ngoài ra có các sự kết hợp đa dạng giữa biểu diễn trong trường hữu hạn và hệ tọa độ cho các điểm đường cong.



Hình 1.16: Các phương pháp khác nhau khi thực hiện phép nhân điểm vô hướng

Các phép toán trên đường cong elliptic được cài đặt thông qua ba tầng như hình trên:

- Tầng trên cùng (Upper Layer): Là các thuộc toán thực hiện phép toán nhân vô hướng k.P, trong đó số nguyên dương k có thể được biểu diễn dưới các dạng khác nhau như dạng nhị phân (Binary) hoặc dạng không liền kề (NAF). Có rất nhiều thuật toán thực hiện phép nhân, một số thuật toán nhân vô hướng điển hình gồm: Thuật toán nhị phân (Binary), thuật toán Montgomery, thuật toán Comb…
- Ở tầng trung gian (middle layer): Tầng này liên quan tới các phép toán trên đường cong elliptic gồm cộng điểm và nhân đôi điểm. Trong tầng này, các thuật toán được cài đặt dựa trên nhiều sự kết hợp khác nhau của biểu diễn trường hữu hạn và hệ tọa độ cho đường cong. Sự kết hợp thích hợp của các thuật toán khác nhau có thể ảnh hưởng đáng kể đến hiệu suất của phép cộng điểm, nhân đôi điểm. Trong tầng này có thể chia thành 2 nhánh chính là nhánh các phép toán đường cong elliptic trên trường nhị phân ($E(GF(2^m))$) và trên trường nguyên tố ($E(GF(p))$). Trong mỗi nhánh lại chia thành 2 loại là: sử dụng hệ tọa độ affine và hệ tọa độ xạ ảnh (projective).
- Tầng thấp nhất (lower layer): Liên quan tới các thuật toán số học trên trường hữu hạn (cộng, trừ, nhân, bình phương, nghịch đảo, modulo …). Một số thuật toán nhân nhanh modulo phải kể đến là: Phương pháp Barrett [7], Phương pháp Montgomery [7], [44, 45].

b) Lựa chọn tham số cho ECC

Ngoài chủ đề tối ưu và hiệu quả ở mức thuật toán trên ECC trong môi trường nguồn tài nguyên hạn chế thì nhiều nghiên cứu có tính hệ thống tập trung vào vấn đề lựa chọn tham số và số modulo an toàn và phù hợp với môi trường có tài nguyên hạn chế, trong đó có kết quả tốt nhất phải kể đến thuật toán của Barrett (xem Algorithm 2.14 trong [7]), trong các nghiên cứu này có thể kể đến việc tìm ra các loại modulo đặc biệt hỗ trợ cho việc tính toán nhanh điển hình là các số nguyên tố NIST (National Institute of Standards and Tecnology) được đưa vào chuẩn FPIS-186-2 [9] để dùng cho các ứng dụng trên hệ mật đường cong elliptic.

1.5.2.2. Áp dụng các kỹ thuật đặc thù cho thiết bị nhúng

a) Sử dụng ngôn ngữ lập trình bậc thấp (ASM)

Trong một số trường hợp thì viết mã bằng ngôn ngữ bậc thấp (assembly) có thể mang lại hiệu quả hơn so với mã assembly được sinh ra từ trình biên dịch. Nhưng cũng không nên kỳ vọng rằng lập trình thuật toán bằng ngôn ngữ assembly thì luôn luôn thực hiện hiệu quả hơn mã mà sinh ra bởi trình biên dịch ngôn ngữ bậc cao. Trong thực tế, các bộ tối ưu được tích hợp trong các trình biên dịch hiện đại thường cũng hiệu quả khi biên dịch ngôn ngữ bậc cao. Một số nghiên cứu cài đặt thuật toán bằng ngôn ngữ assembly tiêu biểu như [41].

b) Sử dụng kỹ thuật song song giữa các CPU: OpenMP

Các bộ vi xử lý có nhiều lõi (core) thường không tự động phân bố tải của một chương trình vào trong các thành phần lõi của vi xử lý đó. Người lập trình phải thêm các khai báo để hỗ trợ hoạt động đa lõi (multicore) vào trong mã nguồn của chương trình đó. Các vi xử lý ARM hiện nay thường có nhiều lõi. Ví dụ, ARM Cortex A9 có hai lõi, và ARM Cortex A15 có bốn lõi. Một chương trình yêu cầu hiệu năng cao nên sử dụng tất cả các lõi của vi xử lý. Để thực hiện điều này, người lập trình phải thêm các khai báo vào trong mã nguồn của chương trình để có thể phân bố các nhiệm vụ đến nhiều tiến trình cùng thực hiện đồng thời thông qua kỹ thuật chia sẻ bộ nhớ. Thông thường, số lượng các tiến trình sẽ bằng với số lõi của vi xử lý, nhằm mục đích mỗi tiến trình có thể chạy trên một lõi vi xử lý khác nhau. OpenMP [60] là một phương pháp để người lập trình có thể song song một chương trình theo cách này.

1.5.2.3. Giải pháp sử dụng phần cứng đặc thù trên thiết bị nhúng

Ngoài các giải pháp thông thường nêu trên thì nhiều hệ thống nhúng cũng như chipset SoC đã tích hợp các bộ đồng xử lý (co-processors) như GPU (Graphical Processor Units), DSP (Digital Signal Processor) hoặc FPGA (Field Programmable Gate Arrays) để thực hiện những kiểu tính toán chuyên dụng mà nhanh hơn so với tính toán trên vi xử lý thông thường. Do các đồng vi xử lý được coi là một thiết bị ngoại vi của bộ vi xử lý chính, nên người lập trình phải viết các đoạn mã riêng cho bộ đồng

xử lý đó, ngoài ra còn phải xử lý các vấn đề khác như điều khiển, giao tiếp, quản lý bộ nhớ và đồng bộ…

a) GPU

Thành phần xử lý đồ họa (GPU) được phổ biến vào cuối những năm 1990, cho phép người dùng trải nghiệm hình ảnh 3D độ nét cao. Ngày trước, các mạch xử lý đồ họa thường cố định, tuy nhiên sau này các GPU bắt đầu được cải tiến để có thể lập trình lại được, giúp cho khả năng linh hoạt cao. Do các tính toán trên GPU ngày càng tăng, nên nhiều hướng nghiên cứu về tính toán lớn đã lựa chọn nền tảng GPU.

Nhằm mục đích để truy nhập đến GPU, OpenCL được sử dụng. Đây là một hạ tầng dùng ngôn ngữ lập trình dựa trên C, để viết chương trình và thực thi nó trên các nền tảng mà có CPU và GPU.

b) FPGA

Các công trình thực hiện cứng hóa phép nhân điểm như các tác giả Moncef Amara, Amar Siad (2011) [12]; Sandeeps, Hameen Shanavas I (2012) [13] để tăng tốc cho việc thực hiện phép nhân điểm trên ECC nhằm tăng nhanh thời gian trao đổi khóa giữa các liên kết trong hệ thống. Một số luận án trong nước để tăng hiệu quả phép nhân điểm của hệ mật ECC được thực hiện trên nền tảng FPGA [42].

c) DSP

Các bộ xử lý tín hiệu số (DSP) được thiết kế để nâng cao hiệu quả các thuật toán xử lý tín hiệu như FFT (Fast Fourier Transform), các bộ lọc FIR/IIR (Finite/Infinite Impulse Response). Các ứng dụng thông thường hay sử dụng DSP là: bộ mã và giải mã tiếng nói và hình ảnh, điều khiển động cơ, nhân dạng tiếng nói. ARM cung cấp một số các chỉ lệnh DSP cho những mục đích này. Các chỉ lệnh DSP gồm nhân, cộng, nhân tích lũy…

d) NEON trên ARM

Ngày nay, thành phần NEON [19] được tích hợp trên các vi xử lý ARM nhằm song song hóa một số phép tính sử dụng kỹ thuật SIMD, NEON là một mở rộng ISA cho các bộ vi xử lý ARM. Các vector trên thành phần này có độ dài 128-bit, và có phép

thực hiện cùng một phép tính trên nhiều làn dữ liệu (lane). Công nghệ NEON đã được trình bày kỹ trong mục 1.2 ở trên. Một số nghiên cứu sử dụng NEON trong mật mã:

- Bernstein and Schwabe [14] đã mô tả cài đặt hiệu quả đối với các nguyên thủy mật mã phi chuẩn sử dụng NEON engine trên bộ xử lý Cortex-A8 ở múc an toàn 128-bit, bằng cách sử dụng đường cong Montgomery and Edwards trên trường nguyên tố $F_{(2^{256}-19)}$, những nguyên thủy mật mã này có thể kháng tấn công kênh kề. Kết quả họ đạt được 527 Kcycles để tính khóa bí mật chia sẻ.
- Hamburg [15] đã cài đặt hiệu quả cho các nguyên thủy mật mã phi chuẩn trên CortexA9 mà không sử dụng NEON với mức an toàn 128-bit, sử dụng đường cong Montgomery and Edwards Curves trên trường $F_{(2^{252}-2^{232}-1)}$ với khả năng kháng tấn công kênh kề. Kết quả đạt được 616 Kcycles để tính khóa bí mật chia sẻ.
- Polyakov [59] đã cài đặt GHASH với NEON, mã xác thực được sử dụng bởi GCM, cho dự án OpenSSL. Hiệu suất thu được là 15 cpb trên Cortex-A8.
- Gần đây nhiều bộ vi xử lý hiện đại đã tích hợp thành phần NEON để xử lý song song kiểu SIMD (đơn chỉ thị đa dữ liệu). ARMv7 hỗ trợ các phép toán VMULL.p8 để nhân đồng thời 8 đa thức 8 bit với chỉ một chỉ lệnh. Trong [17], các tác giả đã đề xuất kỹ thuật cài đặt hiệu quả để xây dựng bộ nhân 64-bit bằng phép toán VMULL.p8, sau đó sử dụng thuật toán nhân Karatsuba để áp dụng cho các đa thức có bậc lớn hơn 64 bit trên các trường $F_{2^{251}}$, $F_{2^{283}}$ và $F_{2^{571}}$.
- Các vi xử lý ARMv8 hỗ trợ phép toán PMULL, chỉ lệnh này sẽ nhân 2 đa thức 64-bit bằng một chỉ lệnh. Bài báo [18] đã khai thác bộ nhân đa thức 64-bit (PMULL) được hỗ trợ trên các vi xử lý ARMv8 và kết hợp với thuật toán Karatsuba để tạo thành các phép nhân có độ dài khác nhau 128-bit, 256-bit và 512-bit. Cuối cùng, các tác giả đã áp dụng phương pháp đề xuất cho đường cong B-251. Kết quả phương pháp đề xuất nhân đa thức trên ARMv8 nhanh hơn 5,5 lần so với các kỹ thuật trên ARMv7.

1.5.3. Đánh giá, nhận xét

1.5.3.1. Hạn chế của các giải pháp an toàn bảo mật trên thiết bị nhúng

Các giải pháp an toàn bảo mật trên thiết bị nhúng rất đa dạng, song có thể nhóm thành hai dạng chính như sau.

Dạng 1: An toàn bảo mật dữ liệu trực tuyến (online)

Có nhiều ứng dụng trên nền thiết bị nhúng, thiết bị di động hoặc các thiết bị IoT gồm: TLS, OpenVPN, IPSec bảo mật ứng dụng VoIP, họp trực tuyến (E2E) (MIKIEY, SDES, ZRTP). Vấn đề đặt ra đối với sản phẩm yêu cầu bảo mật dữ liệu trên đường truyền trong triển khai thực tế đó là việc đảm bảo độ an toàn và hiệu năng khi hoạt động trên nền thiết bị nhúng. Các ứng dụng yêu cầu bảo mật dữ liệu trực tuyến (giao thức TLS, IPSec, bảo mật dữ liệu VoIP...) có hai giai đoạn chính ảnh hưởng tới độ an toàn và tốc độ thực thi:

- ✓ Giai đoạn thiết lập, trao đổi khóa (TLS handshake, IKEv2): Thiết lập các tham số an toàn, các thuật toán mã hóa, giải mã, xác thực và trao đổi khóa.
- ✓ Giai đoạn mã hóa và giải mã gói tin (IPSec, OpenVPN): Sử dụng các thuật toán mã khối với chế độ mã an toàn để bảo mật dữ liệu

Các giao thức trao đổi khóa thường sử dụng các nguyên thủy mật mã khóa công khai như RSA hoặc Elliptic. Ví dụ, trong các giai đoạn trao đổi khóa của TLS và IKEv2, thì giao thức trao đổi khóa hay được sử dụng là: ký số (ECDSA, RSAPSS), trao đổi khóa (ECDH, DH) …

Dạng 2: An toàn bảo mật dữ liệu ngoại tuyến (offline)

Một số ứng dụng hoạt động offline như (mã tệp, mã dữ liệu lưu trữ…) cũng thường có 2 bước:

- Tạo và mã hóa/giải mã các mầm khóa: Trước tiên, các mầm khóa được sinh ngẫu nhiên, sau đó để bảo mật cho những mầm khóa này cần sử dụng các thuật toán mật mã khóa công khai để mã hóa và ký số trên những thông tin này.
- Mã dữ liệu: Các mầm khóa sau đó được giải mã, kiểm tra chữ ký số… và dẫn xuất thành các khóa cho thuật toán mã hóa khóa đối xứng (AES, DES…)

Nhật xét:

Thời gian thực thi của giai đoạn 1 của hai kiểu ứng dụng yêu cầu bảo mật là rất quan trọng trong đời sống thực tế ***cần đáp ứng thời gian thực***, ví dụ giao thức trao đổi khóa cần thời gian đủ nhanh để cho ứng dụng hoạt động mượt mà (trên hệ điều hành Android) hoặc đối với giao thức trao đổi khóa liên quan đến ứng dụng VoIP yêu cầu ngặt nghèo hơn: người dùng không thể chờ quá 10 giây để thiết lập cuộc gọi có bảo mật…

Việc sử dụng các nguyên thủy mật mã thông thường như (RSA) trong những giao thức trao đổi khóa như IKEv2, TLS, OpenVPN trên nền thiết bị nhúng (vi xử lý ARM) là không phù hợp. Do vậy hệ mật đường cong Elliptic là một chọn lựa phù hợp cho môi trường nhúng.

Các nghiên cứu để nâng cao hiệu quả của hệ mật đường cong Elliptic trên nền thiết bị nhúng ARM khá đa dạng (được trình bày ở trong mục 1.5.2) như cải tiến bản thân thuật toán, sử dụng ngôn ngữ bậc thấp, sử dụng các đồng xử lý (FPGA, GPU, NEON…) . Một số hạn chế của những phương pháp đó là:

a) Đánh giá hiệu quả thuật toán về mặt toán học:

Theo hiểu biết của nghiên cứu sinh thì cho đến nay việc đánh giá các thuật toán nhân số lớn chỉ dựa vào tiêu chí *"Thuật toán được coi là hiệu quả hơn nếu có số phép nhân cơ bản ít hơn"*. Do vậy, việc phân rã, đánh giá bài toán thường chỉ tính đến phép nhân mà bỏ qua phép cộng (nguyên nhân do các vi xử lý trước đây có thời gian thực hiện phép nhân lớn hơn nhiều lần so với thời gian thực hiện phép cộng).

b) Về mặt công nghệ:

- Với các nghiên cứu cải thiện hiệu quả dựa trên các tham số đặc biệt [14, 15, FOURQ], tuy nhiên trong một ứng dụng mật mã yêu cầu độ mật cao thì việc sử dụng các tham số đặc biệt này không phù hợp, nhất là trong lĩnh vực an ninh quốc phòng.
- Với những nghiên cứu sử dụng thành phần FPGA [12, 13, 46], GPU [42] hoặc chip hỗ trợ tính toán mật mã chuyên dùng, thì những giải pháp này khá khó khăn khi cài đặt thuật toán. Các chip hỗ trợ tính toán mật mã chuyên dùng thì khó để thay đổi tham số cũng như thuật toán mật mã cài đặc được cố định.

- Hầu hết các nghiên cứu về lĩnh vực này thường tập trung nghiên cứu cải thiện các phép tính toán cơ bản (các phép tính số học) dựa trên đặc điểm của một vài thiết bị cụ thể [41], hoặc dựa trên các tham số đặc biệt [43]. Do vậy nó khó có thể áp dụng cho những nền thiết bị khác hoặc khi sử dụng các tham số khác. Ngoài ra, việc nghiên cứu tích hợp đầy đủ nâng cao hiệu quả vào trong giao thức ECDH, ECDSA cũng còn thiếu.
- Các nghiên cứu cải tiến về thuật toán sẽ có ngưỡng tới hạn [44, 45] và không thể có những đột phá khi sử dụng kết hợp giữa phần cứng và phần mềm

Các hạn chế này có thể được cải thiện trong các nội dung nghiên cứu của luận án.

1.5.3.2. Định hướng nghiên cứu, phát triển của luận án

Để nâng cao hiệu quả cho các thuật toán ECDH, ECDSA trên thiết bị nhúng, luận án định hướng nghiên cứu, đề xuất một số giải pháp để nâng cao hiệu suất, tốc độ tính toán gồm:

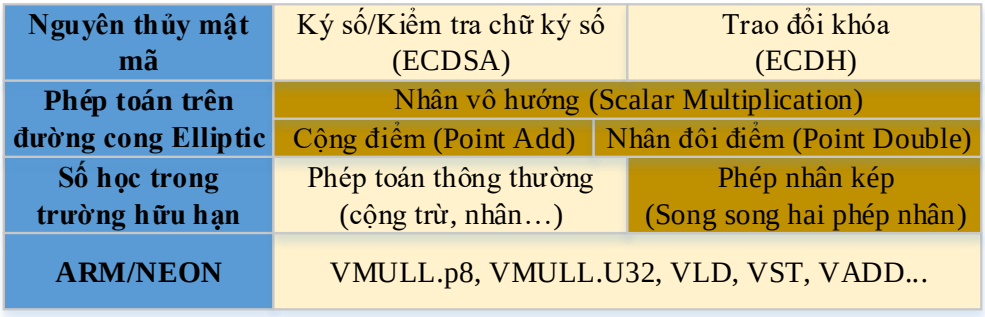
a) Đề xuất phương pháp đánh giá hiệu quả thuật toán:

+ Từ thực tế là trong thuật toán nhân số lớn không chỉ có mặt các phép nhân cơ bản mà còn có các phép toán khác, chẳng hạn như phép cộng cơ bản, vậy nên nếu đưa ra được công thức tính chi phí một thuật toán nhân số lớn chứa cả số phép nhân và số phép cộng thì việc đánh giá tính hiệu quả của một thuật toán sẽ chính xác hơn. Thêm vào đó, trong các thiết bị vi xử lý hiện đại thì thời gian thực hiện giữa phép nhân và phép cộng sẽ không còn chênh lệnh nhiều như trên các vi xử lý trước đây. Do vậy, nếu ta xác định được tỉ số giữa thời gian thực hiện phép nhân cơ bản trên thời gian thực hiện phép cộng cơ bản thì ta sẽ thu được một công thức chỉ phụ thuộc vào số phép cộng cơ bản.

+ Nghiên cứu, đề xuất mô hình toán học cho thuật toán nhân phân tầng để nhân hai số hạng trên trường hữu hạn với chi phí thấp nhất: Xây dựng mô hình để đánh giá chi phí tổng thể của thuật toán và đề xuất thuật toán nhân với chi phí thấp nhất.

b) Nghiên cứu, đề xuất các giải pháp nâng cao hiệu quả thuật toán ECDH và ECDSA sử dụng trong hệ mật Elliptic trên vi xử lý ARMv7 và ARMv8 trong trường nhị phân và trường nguyên tố, đảm bảo an toàn và dễ dàng khi thực thi trên nền thiết bị nhúng.

Mô hình thực hiện như sau:



Hình 1.17: Mô hình các tầng trong mật mã ECC

Các nghiên cứu trong luận án tập trung cải thiện tốc độ thuật toán dựa trên sự kết hợp giữa thuật toán tầng cao (phép toán trên đường cong Elliptic, phép toán số học) và thực thi phép toán song song của thành phần NEON trong tầng thấp (phụ thuộc vào vi xử lý). Trong tầng thấp, luận án chọn sử dụng công nghệ song song nhiều phép tính trong một chỉ lệnh (SIMD) dùng thành phần NEON vì lý do sau:

- Thứ nhất, thành phần NEON hiện nay được tích hợp vào trong các vi xử lý thông thường ARMv7, ARMv8. Hai dòng vi xử lý này hiện được ứng dụng phổ biến trong các thiết bị IoT, thiết bị điện thoại thông minh, các thiết bị nhúng chuyên dụng
- Thứ hai, các thuật toán mật mã cần được bản địa hóa, do vậy việc sử dụng phép toán song song có thể giúp cài đặt các thuật toán được tối ưu và không cần cứng nhắc vào một thuật toán đã cài đặt sẵn có (ví dụ như AES được cứng hóa trên các CHIP chuyên dụng)
- Thức ba, việc nâng cao hiệu quả dựa trên thành phần NEON là khá mềm dẻo hơn các công nghệ khác như FPGA, DSP, GPU…

c) Lựa chọn các nền tảng phần cứng vi xử lý nhúng phù hợp với xu hướng sử dụng trong thực tế. Theo thống kê, đa phần các vi xử lý ARM ứng dụng trong các thiết bị hiện nay dựa trên nền tảng ARMv7 và ARMv8.

Trong luận án sẽ chọn các nền vi xử lý SoC được dùng trong các thiết bị thông thường như được sử dụng trong điện thoại thông minh và máy tính bảng, IoT, sử dụng trong

các thiết bị nhúng. Các thiết bị phần cứng lựa chọn đều có vi xử lý dựa trên kiến trúc ARMv7 và ARMv8; cả hai kiến trúc đều hỗ trợ thành phần NEON.

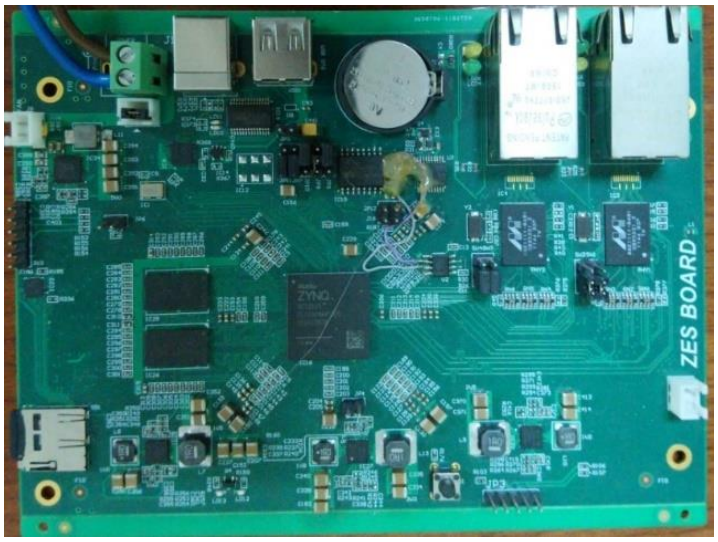
Với những nội dung nghiên cứu trên, luận án góp phần giải quyết một số hạn chế và nâng cao hiệu quả tính toán của một số thuật toán mật mã cho yêu cầu bảo mật dữ liệu truyền thông trên nền thiết bị nhúng.

1.5.4 Các nền tảng phần cứng sử dụng trong luận án

Hiện nay, các sản phẩm bảo mật công nghệ thông tin trên nền thiết bị nhúng được thiết kế và phát triển mạnh ở trong nước nói chung cũng như trong lĩnh vực an ninh quốc phòng nói riêng. Hầu hết các sản phẩm này đều sử dụng vi xử lý ARM phiên bản v7 (ARMv7) và v8 (ARMv8), do vậy để triển khai thực nghiệm các kết quả đề xuất trong luận án nghiên cứu sinh chọn hai Kit phát triển đại diện mà hỗ trợ hai dòng vi xử lý này. Cụ thể, luận án chọn ZesBoard (ZYNQ 7000) để mô phỏng kết quả trên dòng vi xử lý ARMv7 và Kit NXP IMX8M để mô phỏng kết quả trên dòng vi xử lý ARMv8.

1.5.4.1 Kit phát triển ZesBoard

Kit ZesBoard dựa trên nền tảng ZYNQ 7000 [49] là nền tảng lai ghép giữa một ARM dualcore Cortex-A9 (ARMv7) với một FPGA Artic7 của Xilinx tích hợp nhiều ngoại vi truyền thông mạnh như hai ngoại vi truyền thông mạng Gigabit MAC 10/100/1000.



Hình 1.18:Kít ZesBoard (ZYNQ 7000)

Mặc dù Zynq 7000 là một hệ thống mạnh cho lập trình FPGA, tuy nhiên bên trong CHIP zynq này có tích hợp sẵn một vi xử lý ARM Cortex A9 (ARMv7) với hiệu năng vừa phải nhằm mục đích thực thi các phần mềm điều khiển, trong đó các thuật toán mật mã khóa công khai (ECDSA, ECDH) thường được chạy trên vi xử lý ARMv7 này.

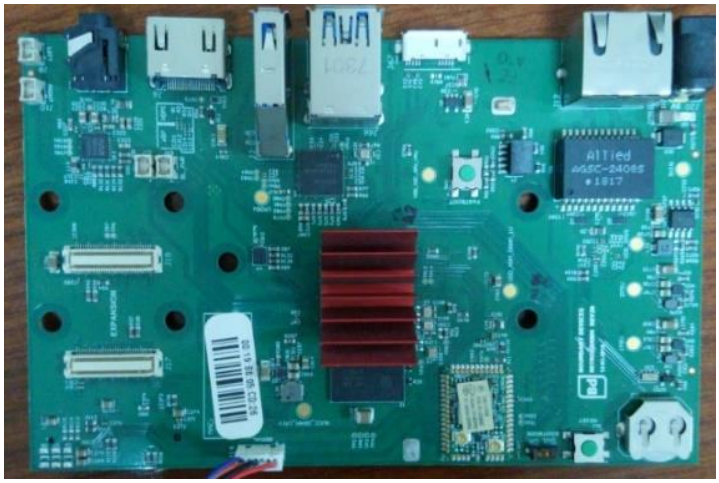
1.5.4.2 Thiết bị phát triển NXP IMX8M

NXP I.MX 8M [47] được thiết kế trên vi xử lý ARMv8 (Cortex A53) là một dòng sản phẩm tập trung vào việc mang lại trải nghiệm âm thanh và video, kết hợp các tính năng dành riêng cho phương tiện truyền thông với xử lý hiệu suất cao được tối ưu hóa cho mức tiêu thụ điện năng thấp.

Lõi phần cứng ARM Cortex-A53

Các bộ vi xử lý ứng dụng họ i.MX dựa trên nền tảng ARM Cortex-A53, có các đặc điểm sau:

- Vi xử lý Cortex-A53 4 lõi với đặc tính:
    - Bộ nhớ cache L1 cho chỉ lệnh: 32 KB
    - Bộ nhớ cache dữ liệu L1: 32 KB
    - Module chuyên dùng sử lý đa phương tiện (MPE)với hỗ trợ công nghệ NEON
    - Floating Point Unit (FPU) hỗ trợ kiến trúc VFPv4-D16
- Hỗ trợ 1 vi xử lý ARM Cortex-M4
- Hỗ trợ kiến trúc Armv8-A 64-bit
- Bộ nhớ cache L2: 512 KB
- Tần số 1.8 GHz



Hình 1.19: Kít phát triển IMX8M

1.6. Kết luận chương 1

Các thiết bị IoT, di động (thoại thông minh, máy tính bảng) đang trở nên phổ biến. Những thiết bị này thường bị hạn chế ở một số khía cạnh như điện năng tiêu thụ, tốc độ xử lý, tài nguyên sử dụng... Do đặc điểm của những thiết bị này thường là không dây và yêu cấp đáp ứng thời gian thực, vấn đề bảo mật dữ liệu đường truyền để ngăn chặn tấn công nghe lén và rò rỉ thông tin cá nhân là rất quan trọng. Vì vậy, việc nghiên cứu triển khai phần mềm bảo mật hiệu quả trên các thiết bị này trở nên rất quan trọng. Các giải pháp bảo mật thường dựa trên nền tảng mật mã khóa công khai và mật mã khóa đối xứng. Đặc biệt, các lược đồ, thuật toán dựa trên hệ mật đường cong elliptic thường được sử dụng do có hiệu quả cao (kích thước khóa nhỏ hơn nhiều so với các hệ mật khóa công khai khác) như thuật toán chữ ký số trên đường cong elliptic (ECDSA) và lược đồ thỏa thuận khóa Diffie-Hellman trên đường cong elliptic (ECDH).

Chương 1 đã đi sâu phân tích các yêu cầu cần bảo mật dữ liệu trên nền thiết bị nhúng, các giải pháp nâng cao hiệu quả hệ mật khóa công khai Elliptic trên nền các thiết bị nhúng (cải tiến thuật toán, sử dụng các tham số đặc biệt; sử dụng các thành phần tính toán nhanh GPU, DSP, FPGA; sử dụng các đồng xử lý ...). Trong các giải pháp này, luận án cũng đã phân tích những ưu điểm và hạn chế của từng phương pháp. Trên cơ sở đó luận án sẽ tập trung vào giải pháp cải tiến thuật toán kết hợp với sử dụng đồng

xử lý (NEON) mà được tích hợp sẵn trên các vi xử lý từ ARMv7 trở đi. Từ đó đặt ra các nội dung cần giải quyết:

- Từ thực tế là trong thuật toán nhân số lớn không chỉ có mặt các phép nhân cơ bản mà còn có các phép toán khác, chẳng hạn như phép cộng cơ bản, vậy nên nếu đưa ra được công thức tính chi phí một thuật toán nhân số lớn chứa cả số phép nhân và số phép cộng thì việc đánh giá tính hiệu quả của một thuật toán sẽ chính xác hơn. Thêm vào đó, trong các thiết bị vi xử lý hiện đại thì thời gian thực hiện giữa phép nhân và phép cộng sẽ không còn chênh lệnh nhiều như trên các vi xử lý trước đây. Do vậy, nếu ta xác định được tỉ số giữa thời gian thực hiện phép nhân cơ bản trên thời gian thực hiện phép cộng cơ bản thì ta sẽ thu được một công thức chỉ phụ thuộc vào số phép cộng cơ bản
- Nghiên cứu mô hình, thuật toán nhân hiệu quả mới với chi phí thấp nhất và đề xuất các thuật toán cải tiến nhằm tăng hiệu năng xử lý dựa trên hỗ trợ tính toán song song của thành phần NEON trên trường nhị phân. Các thuật toán đề xuất được minh chứng và được tích hợp vào giao thức ECDH, ECDSA (nội dung cụ thể được trình bày trong chương 2)
- Các ứng dụng bảo mật thực tế thường hay sử dụng các tham số miền của đường cong ellitpic trên trường nguyên tố, do vậy việc nghiên cứu nâng cao hiệu quả của thuật toán ECDSA và ECDH trên các trường nguyên tố là cần thiết. Các đề xuất cải tiến và nâng cao hiệu quả của cho thuật toán nhân vô hướng dựa trên kết hợp giữa cải tiến NAF và sử dụng phép nhân kép được trình bày trong chương 3.

CHƯƠNG 2. NÂNG CAO HIỆU QUẢ CỦA PHÉP NHÂN SỐ HỌC TRONG TRƯỜNG NHỊ PHÂN TRÊN VI XỬ LÝ ARM

Chương 2 của luận án trình bày về giải pháp để nâng cao hiệu quả của phép nhân số học trên vi xử lý ARM bao gồm: nghiên cứu mô hình, thuật toán nhân hiệu quả mới với chi phí thấp nhất và đề xuất các thuật toán cải tiến nhằm tăng hiệu năng xử lý dựa trên sự kết hợp của thuật toán đề xuất và sự hỗ trợ tính toán song song của thành phần NEON trên trường nhị phân. Các kết quả chính của chương 2 được trình bày trong ba công trình công bố [J1, C1, C2].

Trước đây, việc đánh giá các thuật toán nhân số lớn chỉ dựa vào tiêu chí *"Thuật toán được coi là hiệu quả hơn nếu có số phép nhân cơ bản ít hơn".* Do vậy, việc phân rã, đánh giá bài toán thường chỉ tính đến phép nhân mà bỏ qua phép cộng. Lý do này bởi vì thời gian thực hiện phép nhân lớn hơn nhiều lần so với thời gian thực hiện phép cộng (trên vi xử lý cũ). Cụ thể:

- Đối với phép nhân phổ thông nhân hai số lớn *t* ký tự thì độ phức tạp thuật toán là $O(t^2)$, tức là cần đến cỡ t^2 phép nhân và không quan tâm đến phép cộng
- Đối với phép nhân Karatsuba, để nhân hai số lớn t ký tự thì độ phức tạp của thuật toán là $O\left(t^{\log_2 3}\right) = O(t^{1.59})$.

Ý tưởng:

- Để giảm đi 1 phép nhân so với thuật toán nhân phổ thông thì thuật toán Karatsuba lại phải thêm một vài phép cộng trừ.
- Trong các thiết bị vi xử lý hiện đại thì thời gian thực hiện giữa phép nhân và phép cộng sẽ không còn chênh lệnh nhiều như trên các vi xử lý trước đây (thường trên vi xử lý mới thời gian thực hiện phép nhân gấp 2-3 lần thời gian thực hiện phép cộng).
- Từ thực tế là trong thuật toán nhân số lớn không chỉ có mặt các phép nhân cơ bản mà còn có các phép toán khác, chẳng hạn như phép cộng cơ bản, vậy nên nếu đưa ra được công thức tính chi phí một thuật toán nhân số lớn chứa cả số phép nhân và số phép cộng thì việc đánh giá tính hiệu quả của một thuật toán sẽ chính xác hơn. Ngoài ra, nếu ta xác định được tỉ số giữa thời gian thực hiện

phép nhân cơ bản trên thời gian thực hiện phép cộng cơ bản thì ta sẽ thu được một công thức chỉ phụ thuộc vào số phép cộng cơ bản. Như vậy, khá dễ dàng để so sánh đánh giá độ phức tạp của thuật toán áp dụng cho từng trường hợp.

Trên cơ sở ý tưởng này, trong chương 2 NCS thực hiện hai nội dung (giới hạn trong 2 phép nhân: phổ thông và Karatsuba):

- Đề xuất thuật toán nhân phân tầng có chi phí thấp nhất dựa trên 2 thuật toán nhân cơ sở là phép nhân phổ thông và nhân Karatsuba.
- Áp dụng thuật toán nhân phân tầng này trên hai nền tảng ARMv7 và ARMv8.

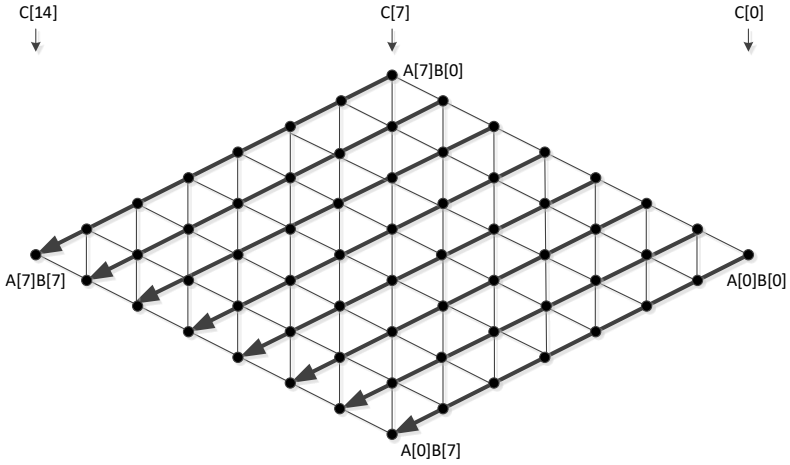
2.1. Phép nhân và phép cộng số học cơ bản trong trường hữu hạn

Trong mục này sẽ mô tả về các kỹ thuật nhân số lớn phù hợp cho các kiến trúc dùng vi xử lý hiện đại. Để mô tả các phương pháp nhân số lớn, ta sử dụng các ký hiệu sau. Gọi A và B là ký hiệu hai số hạng có độ dài m-bit và được lưu ở trong mảng gồm nhiều từ (word). Mỗi số hạng lưu trong mảng được ký hiệu như sau $A = (A[t-1], \dots, A[2], A[1], A[0])$ và $B = (B[t-1], \dots, B[2], B[1], B[0])$. Ta gọi w là số bit của từ, thường được chọn gắn với vi xử lý (8, 16, 32, 64bit) và $t = \lceil m/w \rceil$ là số từ để biểu diễn số nguyên A và B. Ta ký hiệu kết quả phép nhân $C = A * B$ được biểu diễn trong mảng $C = (C[2t-1], \dots, C[2], C[1], C[0])$.

2.1.1. Phép nhân phổ thông

2.1.1.1. Nhân bằng phương pháp quét số hạng (Operand-Scanning)

Đây là phương pháp đơn giản nhất (còn gọi là nhân phổ thông [25]) để tính phép nhân số lớn mà có hai số hạng A và B có t từ. Phương pháp này được thực hiện thông qua hai vòng lặp lồng nhau: ở vòng lặp bên ngoài (vòng lặp i), các giá trị $A[i]$ được nạp. Trong khi vòng lặp bên trong (vòng lặp j) sẽ nạp các giá trị $B[j]$ và được nhân với $A[i]$ với $j = 0, \dots, t-1$. Tích thành phần được tích lũy vào trong cột kết quả trung gian $C[i+j]$ cùng với phần dư của cột trước đó.



Hình 2.1: Minh họa cách tính các tích thành phần. Phương pháp này sử dụng quan điểm hướng hàng (row-wise), trong đó luồng tính toán theo hướng mũi tên.

Số chỉ lệnh để nạp và lưu dữ liệu của phương pháp này như sau:

+ Ở mỗi hàng: Chỉ lệnh nạp trong mỗi vòng lặp bên trong là 2*t* (để tải dữ liệu *B*[*j*], *C*[*i*+*j*]); Chỉ lệnh lưu trong mỗi vòng lặp bên trong là *t* (để lưu *C*[*i*+*j*]). Do vậy tổng chỉ lệnh ở mỗi hàng là 3t

+ Vòng lặp bên ngoài cần nạp *t* số hạng *A*[*i*], và chứa giá *t* giá trị *C*[*i*+*t*]. Tổng chỉ lệnh là 2t.

Do vậy, tổng chi phí của phương pháp này là tải dữ liệu là $2t^2$+t, chi phí lưu dữ liệu $t^2 + t$. Như vậy cần cỡ $3t^2 + 2t$ chỉ lệnh tải và lưu dữ liệu.

Với phương pháp này, thì khả năng thực hiện song song trong thuật toán là khó thực hiện do dữ liệu phụ thuộc vào nhau theo từng hàng.

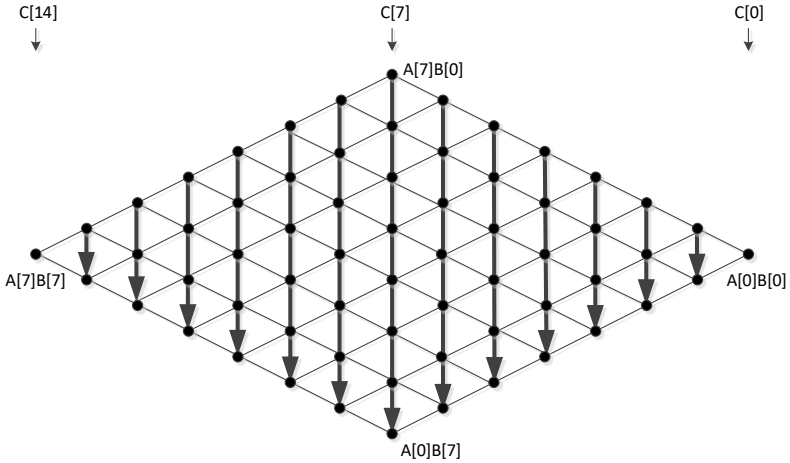
2.1.1.2. Nhân bằng phương pháp quét tích (Product-Scanning)

Ngược với phương pháp quét số hạng, phương pháp này tính nhân số nguyên lớn theo quan điểm dựa trên cột. Được đề xuất bởi Comba [26] là một cách để giảm số lượng truy nhập bộ nhớ. Bởi vì trong mật mã, số lượng cột cần nhân không vượt quá 2^w, do bộ tích lũy có độ lớn là 3 từ, nên nó có thể chứa tổng của tất cả các phép nhân thành phần của một cột mà không phải chứa kết quả trung gian, bởi vì $t < \frac{2^{3w}}{2^{2w}} = 2^w$, trong đó *w* ký hiệu là độ dài bit của một từ. Đầu tiên tất cả các số hạng của mỗi cột được nhân với nhau và được cộng tích lũy (trong một phương pháp nhân và tích lũy). Sau quá trình xử lý một cột, thì từ đầu tiên của bộ tích lũy được lưu trong bộ nhớ là một

phần của kết quả cuối cùng. Do vậy, không có kết quả trung gian cần được lưu hoặc được tải trong giải thuật. Hơn nữa, việc xử lý lan truyền phần dư là khá dễ bởi vì phần dư sẽ được cộng với kết quả của cột kế tiếp. Ngoài ra, chỉ cần năm thanh ghi để thực hiện phép nhân: hai thanh ghi để lưu số hạng đầu vào, ba thanh ghi làm bộ tích lũy. Đây là phương pháp rất phù hợp cho các thiết bị có nguồn tài nguyên hạn chế.

Công thức để tính các tích trong phương pháp quét tích như sau:

$C = A * B = \sum_{k=0}^{2t-2}(\sum_{\substack{i+j=k \\ 0\le i,j\le t-1}} A_i * B_j)W^k$. Trong đó $W = 2^w$ gọi là cơ số.



Hình 2.2. Nhân số lớn theo phương pháp quét tích.

Trong hình thoi thể hiện xử lý các tích thành phần theo hướng cột thay cho theo quan điểm hướng hàng, cần chỉ một phép lưu trữ để chứa từ của kết quả cuối cùng. Chi phí trong toàn bộ phép nhân như sau:

+ Do vòng lặp ngoài có kích thước 2t và vòng lặp trong thay đổi đến từ 0 đến t nên chỉ lệnh tải dữ liệu (cần nạp $A[i]$, $B[j]$ trong mỗi vòng lặp) là: $2t^2$

+ Chỉ lệnh lưu dữ liệu là: $2t$ (mỗi bước của vòng lặp ngoài chỉ cần lưu một giá trị)

+ Tổng chỉ lệnh tải dữ liệu và lưu dữ liệu là: $2t^2 + 2t$

2.1.2. Phép nhân số nguyên lớn trong trường nguyên tố

Trong cài đặt phần mềm, phép nhân trong trường nguyên tố được thực hiện trong các bộ nhân hỗ trợ sẵn trong vi xử lý (ALU) hoặc bộ bằng phần cứng được nhúng vào trong các vi xử lý nhúng.

Bảng 2.1: Tóm tắt chi phí chỉ lệnh nạp và lưu dữ liệu

Phương pháp	Chỉ lệnh tải dữ liệu	Chỉ lệnh lưu dữ liệu	Tổng số chỉ lệnh truy cập bộ nhớ
Quét số hạng	$2t^2 + t$	$t^2 + t$	$3t^2 + 2t$
Quét tích thành phần	$2t^2$	2t	$2t^2 + 2t$

2.1.3 Phép nhân trong trường nhị phân

Trên phần mềm, phép nhân trong trường nhị phân sử dụng kết hợp giữa phép toán EOR (exclusive-or) và phép toán dịch bit. Do các vi xử lý không hỗ trợ các bộ nhân nhị phân chuyên dụng, nên thực hiện phép toán nhân trên trường nhị phân sẽ chậm hơn thực hiện phép nhân trên trường nguyên tố. Tuy nhiên, trong một số thuật toán cài đặt bằng phần mềm có thể nâng cao hiệu quả của phép toán bằng cách giảm số lượng tích thành phần và thay thế phép toán dịch bit bằng phép tra bảng tính trước. Dưới đây sẽ trình bày một số thuật toán nhân nhị phân thường sử dụng trong phần mềm.

Cách lưu trữ đa thức trong trường nhị phân trên vi xử lý ARM

Số học trường nhị phân thường được cài đặt bằng phần mềm bằng cách sử dụng biểu diễn cơ sở đa thức, trong đó các phần tử của $\mathbb{F}_{2^m}$ được biểu diễn dưới dạng đa thức có bậc lớn nhất là $m - 1$ trên $\mathbb{F}_2$. Giả sử trên một nền tảng có kiến trúc W-bit (W = 32 đối với ARM), một phần tử trên trường nhị phân $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$ có thể được biểu diễn bằng một vectơ nhị phân $a = (a_{m-1}, \dots, a_2, a_1, a_0)$ có độ dài m. Đặt $t = \lceil m/W \rceil$ và $s = Wt - m$. Trong phần mềm, a được lưu trữ trong một mảng gồm các từ W-bit: $A = (A[t-1], \dots, A[2], A[1], A[0])$, trong đó bit ngoài cùng bên phải của $A[0]$ là a_0 và s bit ngoài cùng bên trái của $A[t-1]$ không được sử dụng (luôn được thiết lập là 0).

- Phép cộng cơ bản trong trường nhị phân: Trên nền kiến trúc ARM hỗ trợ chỉ lệnh EOR để thực hiện phép cộng hai phần tử 32 bit trong trường nhị phân.

- Phép nhân:

Bản thân tập lệnh của vi xử lý ARM không hỗ trợ chỉ lệnh để thực hiện phép nhân cơ bản trong trường nhị phân. Để thực hiện điều này trên phần mềm thường sử dụng kết hợp giữa chỉ lệnh dịch bit và chỉ lệnh XOR.

Phép nhân trên $\mathbb{F}_{2^m}$ (phép nhân trên trường) được thực hiện theo modulo đa thức rút gọn $f(z) = z^m + r(z)$ - một đa thức nhị phân cố định có bậc bằng m. Phép nhân này có thể được thực hiện theo hai bước: đầu tiên, phép nhân đa thức của các toán hạng; thứ hai, rút gọn modulo theo đa thức $f(z)$.

Phương pháp dịch và cộng (thuật toán 2.1) đối với phép nhân trên trường dựa trên quan sát rằng $a \cdot b = a_{m-1}x^{m-1}b + \cdots + a_2x^2b + a_1xb + a_0b$. Bước lặp i của thuật toán tính toán $x^ib \bmod f(x)$ và thêm kết quả vào bộ tích lũy c nếu $a_i = 1$. Lưu ý rằng $b \cdot x \bmod f(x)$ có thể được tính toán một cách dễ dàng bằng cách dịch trái biểu diễn vectơ của b, sau đó cộng thêm $r(x)$ vào b nếu $b_m = 1$.

Thuật toán 2.1. Thuật toán nhân dịch-và-cộng từ phải đến trái [7]

Đầu vào: Các đa thức nhị phân $a(x)$ và $b(x)$ có bậc lớn nhất là $m - 1$.

Đầu ra: $c(x) = a(x) \cdot b(x) \bmod f(x)$.

1. Nếu $a_0 = 1$ thì $c \leftarrow b$; nếu không thì $c \leftarrow 0$.

2. Với i từ 1 đến $m - 1$ thực hiện:

2.1 $b \leftarrow b \cdot x \bmod f(x)$.

2.2 Nếu $a_i = 1$ thì $c \leftarrow c + b$.

3. Trả về (c).

Mặc dù thuật toán 2.1 phù hợp với cài đặt trên phần cứng vì có thể thực hiện dịch vectơ trong một xung nhịp, nhưng số lượng lớn các phép dịch theo ký tự làm cho việc triển khai bằng phần mềm trở nên không đạt hiệu quả như mong muốn. Phần tiếp theo đây sẽ trình bày các phương pháp nhanh hơn để thực hiện phép nhân trên trường.

Phép nhân đa thức theo comb

Phương pháp comb cho phép nhân đa thức dựa trên quan sát rằng nếu $b(x) \cdot x^k$ đã được tính cho một giá trị $k \in [0, 31]$ nào đó, thì có thể dễ dàng tính được $b(x) \cdot x^{32j+k}$ bằng cách nối j từ 0 vào bên phải biểu diễn vectơ của $b(x) \cdot x^k$. Thuật toán 2.2 xem xét các bit của các ký tự của A từ phải sang trái, trong khi Thuật toán 3 xem

xét các bit từ trái sang phải. Ký hiệu sau được sử dụng: nếu $C = (C[n], \dots, C[2],\ C[1],\ C\ [0])$ là một vectơ, thì $C\{j\}$ biểu thị vectơ chặt cụt $(C[n], \dots, C[j+1],\ C[j])$.

Thuật toán 2.2. Thuật toán nhân đa thức comb từ phải sang trái [7]

Đầu vào: Các đa thức nhị phân $a(x)$ và $b(x)$ có bậc lớn nhất là $m-1$.

Đầu ra: $c(x) = a(x) \cdot b(x)$.

1. $C \leftarrow 0$.

2. Với k từ 0 đến 31 thực hiện

2.1 Với j từ 0 đến $t-1$ thực hiện

Nếu bit thứ k của $A[j]$ bằng 1 thì cộng thêm B vào $C\{j\}$.

2.2 Nếu $k = 31$ thì $B \leftarrow B \cdot x$.

3. Trả về (C).

Thuật toán 2.3. Thuật toán nhân đa thức comb từ trái sang phải [7]

Đầu vào: Các đa thức nhị phân $a(x)$ và $b(x)$ có bậc lớn nhất là $m-1$.

Đầu ra: $c(x) = a(x) \cdot b(x)$.

1. $C \leftarrow 0$.

2. Với k từ 31 đến 0 thực hiện

2.1 Với j từ 0 đến $t-1$ thực hiện

Nếu bit thứ k của $A[j]$ bằng 1 thì cộng thêm B vào $C\{j\}$.

2.2 Nếu $k \neq 0$ thì $C \leftarrow C \cdot x$.

3. Trả về (C).

Thuật toán 2.2 và 2.3 đều nhanh hơn thuật toán 2.1 vì có ít phép dịch vectơ (nhân với x). Thuật toán 2.2 nhanh hơn thuật toán 2.3 vì phép dịch vectơ trong thuật toán 2.2 liên quan đến vectơ B có độ dài t từ, trong khi phép dịch vectơ trong thuật toán 2.3 liên quan đến vectơ C có độ dài $2t$. Theo [7], thuật toán 2.3 có thể được tăng tốc đáng kể bằng cách tính trước $u(x) \cdot b(x)$ với tất cả các đa thức $u(x)$ bậc nhỏ hơn w, trong đó w là độ dài của từ và xem xét các bit w của $A[j]$ tại một thời điểm. Tuy nhiên, phương pháp này đòi hỏi phải đánh đổi chi phí lưu trữ. Phương pháp đã được chỉnh sửa này với $w = 4$ được trình bày dưới dạng Thuật toán 2.4.

Thuật toán 2.4. Thuật toán nhân đa thức comb từ trái sang phải với cửa sổ có độ dài $w = 4$ [7]

Đầu vào: Các đa thức nhị phân $a(x)$ và $b(x)$ có bậc lớn nhất là $m - 1$.

Đầu ra: $c(x) = a(x) \cdot b(x)$.

1. Tính toán $B_u = u(x) \cdot b(x)$ với tất cả các đa thức $u(x)$ có bậc lớn nhất bằng 3.

2. $C \leftarrow 0$.

3. Với k từ 7 đến 0 thực hiện

3.1 Với j từ 0 đến $t - 1$ thực hiện

Đặt $u = (u_3, u_2, u_1, u_0)$, trong đó u_i là bit $(4k + i)$ của $A[j]$. Cộng thêm B_u vào $C\{j\}$.

3.2 Nếu $k \neq 0$ thì $C \leftarrow C \cdot x^4$.

4. Trả về (C).

2.1.4. Xác định tỷ số giữa phép nhân và phép cộng trên vi xử lý ARMv7/v8

Mục này sẽ xác định tỷ số giữa chỉ lệnh nhân và chỉ lệnh cộng trên vi xử lý ARM. Việc xác định được tỷ số giữa phép nhân và phép cộng giúp cho việc xây dựng được thuật toán nhân phân tầng với chi phí thấp nhất (chi tiết trình bày ở mục 2.2).

2.1.4.1 Tỷ số giữa chỉ lệnh nhân và cộng trong trường nguyên tố

a) Đối với chỉ lệnh cơ bản của ARM

Tập lệnh trên vi xử lý ARMv7,v8 hỗ trợ chỉ có lệnh nhân cơ bản cho phép cộng và phép nhân 32 bit thông thường sử dụng chỉ lệnh: ADD, ADC, SUB, MUL, SMULL, UMULL. Theo phụ lục B (Instruction Cycle Timings) [27], đối với tập lệnh cơ bản trên ARM thì tỷ số giữa phép nhân và phép cộng $\frac{SMULL}{ADC} = 3$

b) Đối với chỉ lệnh trên thành phần NEON của ARM

Ngoài ra chỉ lệnh trên vi xử lý ARM thì thành phần đồng xử lý NEON cung cấp chỉ lệnh để thực hiện song song nhiều phép nhân thông thường:

- Chỉ lệnh VMULL.U32: Nhân song song hai cặp số nguyên không dấu (32bit) ở dạng vector

- Chỉ lệnh VMLAL.U32: Nhân và cộng tích lũy song song hai cặp số nguyên không dấu (32bit) ở dạng vector
- VADD: Cộng song song các cặp số nguyên không dấu ở dạng vector

Thời gian thực hiện chỉ lệnh trên NEON trên vi xử lý ARM Cortex A9, được nêu tại mục 3.3 [28], theo thông số trong tài liệu này thì tỷ số thời gian thực thi giữa VMULL.U32 và VADD: $\frac{VMULL.u32}{VADD} = 2$.

2.1.4.2 Tỷ số giữa chỉ lệnh nhân và chỉ lệnh cộng trong trường nhị phân

a) Đối với chỉ lệnh cơ bản của ARM

Tập lệnh cơ bản trong vi xử lý ARM không hỗ trợ phép nhân trong trường nhị phân và chỉ hỗ trợ phép cộng trong trường nhị phân là chỉ lệnh EOR

b) Đối với chỉ lệnh trên thành phần NEON của ARM

Đối với vi xử lý ARMv7/v8 có tích hợp thành phần NEON, hỗ trợ chỉ lệnh nhân và cộng song song trên trường nhị phân là:

- VMULL.p8: Nhân nhị phân 8 đa thức 8bit
- VEOR: Thực hiện phép toán EOR trên vector.

Trong tài liệu [28] chỉ ra tỷ số thời gian thực thi giữa VMULL.P8 và VEOR: $\frac{VMULL.P8}{VEOR} = 1$.

Ghi chú: Kiến trúc ARMv8 là kiến trúc 64-bit. Các dòng vi xử lý Cortex-A53 và Cortex-A57 có cơ chế pipeline tương tự như Cortex-A7 và Cortex-A15 tương ứng (xem mục 1.2.3 - chương 1).

2.2. Nhân phân tầng trong trường hữu hạn

2.2.1. Số nguyên lớn và việc xử lý trên các số nguyên lớn

Trong những thiết bị có hỗ trợ các phép tính số học cơ bản thì các phép tính này thường bị hạn chế bởi cấu hình của thiết bị. Cụ thể chúng chỉ thực hiện được với các số nguyên trong phạm vi $[0, 2^W)$ (hay còn được gọi là các số nguyên W-bit) với một giá trị W xác định nào đó. Việc thực hiện các phép toán với các số nguyên lớn hơn W-bit (còn được gọi là các *số nguyên lớn*, hay đơn giản là *số lớn*) sẽ được xử lý như sau:

– *Đầu tiên, khai báo các số lớn như là một mảng các số nguyên W-bit theo cách thức sau: Giả sử x là một số lớn có triển khai B-phân, ở đây $B = 2^W$, là*

$$x = x_{t-1}B^{t-1} + \cdots + x_0, 0 \le x_i \le B, i = 0, \dots, t-1, x_{t-1} > 0$$

được tương ứng với mảng $x_{t-1}, \dots, x_0$. Giá trị t ở trên được gọi là số ký tự của x. Lúc này ta viết

$$x = x_{t-1}, \dots, x_0$$

– *Tiếp đến thực hiện việc tính toán các số lớn thông qua các thuật toán tương ứng cụ thể.*

Thuật toán cộng và thuật toán trừ hai số *t* ký tự thực hiện theo Algorithm 2.5 và Algorithm 2.6 trang 30 trong [25]. Cả hai thuật toán trên đều cần đến *t* phép cộng có nhớ hoặc tương tự *t* phép trừ có mượn (hai phép toán trên có chi phí như nhau).

Ký hiệu *a là thời gian trung bình để thực hiện một phép cộng hai ký tự với nhau* và $A(t)$ là chi phí tính toán của phép cộng (trừ) hai số t ký tự theo hai thuật toán này thì

$$A(t) = ta. \qquad (2.1)$$

Chú ý 1. *Đại lượng a là một giá trị xác định phụ thuộc vào thiết bị tính toán được sử dụng.*

2.2.2. Thuật toán nhân số lớn

a) Thuật toán nhân theo phương pháp phổ thông

Cho hai số t ký tự có biểu diễn là $x = x_{t-1}, \dots, x_0$ và $y = y_{t-1}, \dots, y_0$ với $x_i, y_i, 0 \le i \le t-1$ là các số W-bit. Thuật toán nhân phổ thông thực hiện việc tính tích của x và y theo công thức

$$x * y = \sum_{s=0}^{2t-2} \Big(\sum_{\substack{i+j=s \\ 0 \le i,j \le t-1}} x_i y_i \Big) B^s. \qquad (2.2)$$

Theo công thức trên thì thuật toán nhân cần thực hiện t^2 phép nhân hai số *W*-bit. Do ta đang xét các số theo dạng khai triển *B*-phân của nó vậy nên tổng của t số hạng $x_0y_0 + x_1y_1 \cdot B^2 + \cdots + x_{t-1}y_{t-1} \cdot B^{2t-2}$ là không cần phải tính. Khi đấy trên thực tế ta phải cộng thêm $t^2 - t$ các số hạng x_iy_i còn lại vào tổng này.

Khi đó, ký hiệu $M_S[t]$ là thuật toán nhân phổ thông hai số t ký tự, $M_S(t)$ là chi phí của nó và m là *thời gian trung bình để thực hiện một phép nhân hai ký tự với nhau*, do $x_i, y_i, 0 \leq i \leq t-1$ là các số 2 ký tự nên luận án ước lượng được (một cách xấp xỉ)

$$M_S(t) = t^2 m + 2(t^2 - t)a. \qquad (2.3)$$

Rõ ràng chi phí này cũng được xác định phụ thuộc vào thiết bị sử dụng.

b) Thuật toán nhân theo phương pháp của Karatsuba

Xuất phát từ nhận xét là trong tổng $\sum_{i+j=s} x_i y_i$ nếu $i \neq j$ thì tổng này luôn chứa các cặp số hạng $x_i y_j$ và $x_j y_i$. Hơn nữa, ta có

$$x_i y_j + x_j y_i = (x_i + x_j)(y_i + y_j) - x_i y_i - x_j y_j. \qquad (2.4)$$

Thay tổng $x_i y_j + x_j y_i$ trong (2.2) bằng vế phải của (2.4) ta nhận được công thức tính tích Karatsuba (xem mục 2.2.2 [7]). Thuật toán thực hiện việc tính tích theo công thức Karatsuba được gọi là thuật toán nhân theo phương pháp Karatsuba. Bây giờ, bỏ qua việc trình bày chi tiết thuật toán nhân theo phương pháp Karatsuba, luận án xác định chi phí để thực hiện thuật toán thông qua việc phân tích sự thay đổi dựa theo công thức (2.4) ở trên.

Để ý rằng mỗi một thay đổi theo (2.1) sẽ làm cho số lượng số hạng của công thức Karatsuba tăng thêm 1 và do đó $x_i y_i, x_j y_j$ cũng cần được tính trong công thức (2.2) nên để có vế phải của (2.4) chỉ phát sinh thêm việc tính $(x_i + x_j)(y_i + y_j)$, tức là thêm 2 phép cộng các ký tự và 1 phép nhân các ký tự. Điều này có nghĩa là vế phải cần đến 2 phép cộng số 1 ký tự, 2 phép trừ các số 2 ký tự cùng với 1 phép nhân các ký tự trong khi vế trái cần 2 phép nhân các ký tự và 1 phép cộng các số 2 ký tự. Như vậy, mỗi thay đổi theo công thức (2.4) sẽ làm cho việc thực hiện công thức Karatsuba tăng thêm chi phí là

$$-m + 4a$$

Để thực hiện toàn bộ thuật toán nhân theo phương pháp Karatsuba, ta có tổng cộng $\frac{t(t-1)}{2}$ sự thay đổi theo công thức (2.4). Từ đó nhân được tổng chi phí tăng thêm so với thuật toán nhân theo phương pháp phổ thông là

$$\frac{t(t-1)}{2}(-m+4a)$$

Ký hiệu $M_K[t]$ là thuật toán nhân theo phương pháp Karatsuba hai số t ký tự và $M_K(t)$ là chi phí của nó. Từ chi phí tăng thêm ở trên và công thức (2.3) ta tính được

$$\begin{aligned} M_K(t) &= t^2m + 2(t^2-t)a + \frac{t(t-1)}{2}(-m+4a) \\ &= \frac{t^2-t}{2}m + 4(t^2-t)a \end{aligned} \tag{2.5}$$

Từ các công thức (2.3) và (2.5) ta dễ dàng nhận được kết quả sau.

Bổ đề 1. *Ta có:*

1. *Với mọi $t > 1$ ta có $M_S(t) > 4A(t), M_K(t) > 4A(t)$.*
2. *Thuật toán nhân theo phương pháp Karatstuba chỉ hiệu quả hơn thuật toán nhân phổ thông khi và chỉ khi $\frac{m}{a} > 4$.*

Chứng minh. Sử dụng các công thức (2.1), (2.2) và (2.5) để so sánh, tính toán trực tiếp cho ta khẳng định đã nêu.

2.2.3. Mô tả thuật toán phân tầng

Cho $t = u \cdot v$ với $u, v > 1$. Khi đó các số t ký tự B-phân có thể được hiểu như là các số u ký tự B^v-phân. Thật vậy, xét x là một số nguyên lớn biểu diễn dưới dạng B-phân

$$x = x_{t-1}B^{t-1} + x_{t-2}B^{t-2} + \cdots + x_1B^1 + x_0,$$

với $t = u \cdot v$. Ta viết lại biểu diễn của x thành

$$\begin{aligned} x = {} & x_{t-1}B^{v-1} \cdot (B^v)^{u-1} + x_{t-2}B^{v-2} \cdot (B^v)^{u-1} + \cdots + x_{t-(v-1)}B^{v-(v-1)} \cdot (B^v)^{u-1} \\ & + x_{t-v} \cdot B^{v-v} \cdot (B^v)^{u-1} + x_{t-v-1} \cdot B^{t-v-1} + \cdots + x_1 \cdot B + x_0 \\ = {} & (x_{t-1} \cdot B^{v-1} + x_{t-2} \cdot B^{v-2} + \cdots + x_{t-v}) \cdot (B^v)^{u-1} + x_{t-v-1} \cdot B^{t-v-1} + \cdots + x_1 \\ & \cdot B + x_0 \end{aligned}$$

Ta lại biến đổi tương tự

$$\begin{aligned} & x_{t-v-1} \cdot B^{t-v-1} + \cdots + x_1 \cdot B + x_0 \\ & = (x_{t-v-1} \cdot B^{v-1} + x_{t-v-2} \cdot B^{v-2} + \cdots + x_{t-2v}) \cdot (B^v)^{u-2} + x_{t-2v-1} \\ & \cdot B^{t-2v-1} + \cdots + x_1 \cdot B + x_0 \end{aligned}$$

Cứ tiếp tục thực hiện như trên, cuối cùng ta nhận được biểu diễn của x là

$$x = (x_{t-1}B^{v-1} + \cdots + x_{t-v+1} \cdot B + x_{t-v}) \cdot (B^v)^{u-1}.$$
$$+(x_{t-v-1}B^{v-1} + \cdots + x_{t-2v+1} \cdot B + x_{t-2v}) \cdot (B^v)^{u-2}.$$
$$+ \cdots + +(x_{t-(u-2)v-1}B^{v-1} + \cdots + x_{v+1} \cdot B + x_v) \cdot (B^v)^1.$$
$$+(x_{v-1}B^{v-1} + \cdots + x_1 \cdot B + x_0).$$

Như vậy ta thấy x được biểu diễn thành một mảng gồm u ký tự B^v-phân, và mỗi ký tự trong mảng lại chính là một số gồm v ký tự *B*-phân.

Với cách biểu diễn một số $t = u \cdot v$ ký tự *B*-phân như một số gồm u ký tự B^v-phân, trong đó mỗi ký tự này lai là một số gồm v ký tự *B*-phân, ta có thể xây dựng phép nhân hai số nguyên t ký tự *B*-phân thông qua thuật toán nhân hai số có u ký tự B^v-phân và thuật toán nhân hai số có v ký tự *B*-phân. Luận án gọi phương pháp này là phương pháp nhân *"phân 2 tầng* u*:* v*",* trong đó phần tính toán với các số có u ký tự B^v-phân được gọi là *tầng cao*, và phần tính toán còn lại được gọi là *tầng thấp*.

Một cách tổng quát, nếu $t = t_{s-1} \cdot t_{s-2} \dots t_0$ với $t_i > 1, 0 \le i \le s-1$ thì ta có thể xây dựng thuật toán nhân hai số t ký tự *B*-phân từ *s* thuật toán nhân hai số t_i ký tự B^{r_i}-phân, trong đó r_i được xác định như sau:

$$r_0 = 1.$$
$$r_i = t_{i-1} \dots t_0, 0 < i < s. \qquad (2.6)$$
$$r_s = t.$$

Thuật toán tổng quát này được gọi là "*Thuật toán phân s tầng* t_{s-1}: … : t_i: … : t_o", các chỉ số i được gọi là chỉ số tầng của thuật toán. Ta ký hiệu $A_i[t_i, r_i]$ là thuật toán nhân hai số t_i ký tự B^{r_i}-phân. Do r_i luôn được xác định theo công thức (2.6) nên để đơn giản, ta ký hiệu $A_i[t_i]$ là thuật toán sử dụng tại tầng thứ i và lúc này, Thuật toán phân s tầng t_{s-1}: … : t_i: … : t_o mô tả ở trên được ký hiệu là

$$A_{s-1}[t_{s-1}]: \dots : A_i[t_i]: \dots : A_0[t_0]$$

Ký hiệu $(A[t])^k = A[t]: \dots : A[t]$.

Nếu ký hiệu chi phí cho thuật toán $A_i[t_i]$ là $A_i(t_i)$ thì chi phí cho thuật toán $A_{s-1}[t_{s-1}]: \dots : A_i[t_i]: \dots : A_0[t_0]$ được ký hiệu là

$$A_{s-1}(t_{s-1}): \dots : A_i(t_i) :\dots: A_0(t_0)$$

Trong luận án này chỉ xét hai thuật toán nhân số lớn, hoặc là thuật toán theo phương pháp nhân phổ thông, hoặc là nhân theo phương pháp Karatstuba vậy nên các thuật toán dùng tại tầng thứ i của Thuật toán phân s tầng chỉ là một trong hai loại $A_i[t_i] = M_S[t_i]$ và $A_i[t_i] = M_K[t_i]$. Do vậy số lượng các thuật toán có thể phân thành s tầng đúng bằng 2^s và để xây dựng được Thuật toán phân s tầng cụ thể có chi phí tính toán thấp chúng ta chỉ cần xem xét và lựa chọn thuật toán tương ứng với giá trị chi phí nhỏ nhất trong 2^s thuật toán có thể.

2.2.4. Một số tính chất về chi phí của thuật toán phân tầng

Thuật toán phân tầng luận án trình bày ở mục trước có một số tính chất thể hiện qua mệnh đề sau.

Mệnh đề 1. *Cho $t = u \cdot v$ với $u, v > 1$. Khi đó ta có*

1. $M_S(u) : M_S(v) = M_S(s) : M_S(u) = M_S(t)$.
2. $M_K(u) : M_K(v) = M_K(v) : M_K(u) \leq M_K(t)$.
3. $M_K(u) : M_S(v) - M_K(v) : M_S(u) = \frac{t(u-v)}{2}(-m + 4a)$.
4. $M_S(u) : M_K(v) - M_S(v) : M_K(u) = \frac{t(u-v)}{2}(-m - 4a)$.
5. $M_S(u) : M_S(v) - M_K(u) : M_S(v) = \frac{t(u-v)}{2}m + t(u-1)(v-3)a \geq 0$.

Chứng minh. **1.** Theo các công thức (2.3) và (2.1) ta có

$$\begin{aligned} M_S(u) : M_S(v) &= u^2 M_S(v) + 2(u^2 - u)A(v). \\ &= u^2(v^2 m + 2(v^2 - v)a) + 2(u^2 - u)va. \\ &= (uv)^2 m + 2(u^2(v^2 - v) + (u^2 - u)v)a. \\ &= (uv)^2 m + 2((uv)^2 - uv)a. \\ &= t^2 m + 2(t^2 - t)a. \\ &= M_S(t). \end{aligned}$$

Đổi vai trò của u và v trong biến đổi ở trên, ta thu được

$$M_S(v) : M_S(u) = M_S(t).$$

Điều này có nghĩa là

$$M_S(u) : M_S(v) = M_S(s) : M_S(u) = M_S(t).$$

2. Theo (2.5) ta có

$$M_K(u): M_K(v) = \frac{u^2+u}{2} M_K(v) + 4(u^2-u)A(v).$$
$$= \frac{u^2+u}{2}\left(\frac{v^2+v}{2} m + 4(v^2-v)a\right) + 4(u^2-u)va.$$
$$= \frac{(u^2+u)(v^2+v)}{4} m + 2((u^2+u)(v^2-v) + 2\big((u^2-u)v\big)a.$$
$$= \frac{t^2+t(u+v)}{4} m + 2(t^2+t(u+v)-2t)a. \quad (2.7)$$

Biểu thức cuối cùng là một biểu thức đối xứng giữa u và v, cho nên nếu đổi vai trò của u và v ta cũng sẽ nhận kết quả như vậy. Điều này có nghĩa là

$$M_K(u): M_K(v) = M_K(v): M_K(u).$$

Bây giờ, từ các công thức (2.5) và (2.7), ta có

$$M_K(t) - M_K(u): M_K(v) = \left(\frac{t^2+t}{2} - \frac{t^2+t(u+v)}{4}\right) m.$$
$$+2(2(t^2-t) - \big((t^2+t(u+v)-2t)\big)a$$
$$= \frac{t^2+t-t(u+v)}{4} m + 2\big(t^2-t(u+v)\big)a.$$

Theo giả thiết các số nguyên $u, v > 1$ nên

$$t^2 - t(u+v) = (uv)^2 - (uv)(u+v)$$
$$= uv\big(uv-(u+v)\big).$$
$$= uv(u(v-1)-(v-1)-1).$$
$$= uv\big((u-1)(v-1)-1\big).$$
$$\geq 0.$$

Từ đây ta nhận được khẳng định

$$M_K(u): M_K(v) = M_K(v): M_K(u) \leq M_K(t).$$

3. Từ các công thức (2.3) và (2.5) ta có

$$M_K(u): M_S(v) = \frac{u^2+u}{2} M_S(v) + 4(u^2-u)A(v).$$
$$= \frac{u^2+v^2}{2}(v^2 m + 2(v^2-v)a) + 4(u^2-u)va.$$
$$= \frac{(u^2+u)v^2}{2} m + ((u^2+u)(v^2-v) + 4(u^2-u)v)a.$$
$$= \frac{t^2+tv}{2} m + (t^2+t(3u+v)-5t)a. \quad (2.8)$$

Đổi vai trò của u và v trong công thức (2.8) ta nhận được

$$M_K(v): M_S(u) = \frac{t^2+tu}{2}m + (t^2 + t(3v+u) - 5t)a. \quad (2.9)$$

Lấy (2.8) trừ đi (2.9) ta được

$$M_K(u): M_S(v) - M_K(v): M_S(u) = \frac{t(v-u)}{2}m + 2t(u-v)a.$$
$$= \frac{t(u-v)}{2}(-m + 4a).$$

4. Tương tự như trên, từ các công thức (2.3) và (2.5) ta có

$$M_S(u): M_K(v) = u^2 M_K(v) + 2(u^2 - u)A(v).$$
$$= u^2\left(\frac{v^2+v}{2}m + 4(v^2 - v)a\right) + 2(u^2 - u)va.$$
$$= \frac{u^2(v^2+v)}{2}m + 2(2u^2(v^2 - v) + (u^2 - u)v)a.$$
$$= \frac{t^2+tv}{2}m + 2t(2t - u - 1)a. \quad (2.10)$$

Đổi vai trò của u và v trong công thức (2.10) ta nhận được

$$M_S(v): M_K(u) = \frac{t^2+tu}{2}m + 2t(2t - v - 1)a. \quad (2.11)$$

Lấy (2.10) trừ đi (2.11) ta được

$$M_S(u): M_K(v) - M_S(v): M_K(u) = \frac{t(v-u)}{2}m + 2t(v-u)a.$$
$$= \frac{t(u-v)}{2}(-m - 4a).$$

5. Từ công thức (2.8) và tính chất 1 ta có

$$M_S(u): M_S(v) - M_K(u): M_S(v) = t^2 m + 2(t^2 - t)a.$$
$$-(\frac{t^2+tv}{2}m + (t^2 + t(3u+v) - 5t)a).$$
$$= \frac{t(t-v)}{2}m + t(u-1)(v-3)a. \quad (2.12)$$

Nếu $v \geq 3$ thì do $u > 1$, $t = uv$ nên từ (2.12) ta có ngay

$$M_S(u): M_S(v) - M_K(u): M_S(v) \geq 0.$$

Ngược lại, nếu $v < 3$, thì từ giả thiết $v > 1$ suy ra $v = 2$. Khi đó công thức (2.12) trở thành

$$M_S(u): M_S(v) - M_K(u): M_S(v) = 2u(u-1)(m-a) > 0$$

do ta luôn có $u > 1$ và $m > a$. Từ đây ta nhận được khẳng định cần chứng minh. Như vậy ta đã chứng minh được các tính chất của thuật toán phân tầng.

2.2.5. Thuật toán nhân số nguyên lớn có chi phí thấp nhất

Với thuật toán phân tầng và tính chất của nó trình bày trong hai mục trước luận án chứng minh một kết quả quan trọng sau.

Định lý 1. *Cho* $t = t_{s-1} \dots t_i \dots t_0$ với $t_i, 0 \le i \le s-1$ *là các số nguyên tố và* $t_{i-1} \ge t_i$. *Khi đó thuật toán nhân hai số nguyên* t *ký tự có chi phí thấp nhất là*

(i) $M_K[t_{s-1}]: \dots : M_K[t_0]$ *nếu* $m > 4a$.

(ii) $M_K[t_{s-1}]: \dots : M_K[t_1]: M_S[t_0]$ *trong trường hợp ngược lại.*

Chứng minh. Với mọi thuật toán nhân hai số nguyên có $k = t_{r-1} \dots k_j \dots k_0$ ký tự được phân thành r tầng $k_{r-1} \dots k_j \dots k_0$ là $A_{r-1}[k_{r-1}]: \dots : A_j[k_j]: \dots : A_0[k_0]$, nếu k_j là hợp số thì bằng cách phân tầng thuật toán $A_j[u_j]$ theo cùng một phương pháp như thuật toán này thì từ tính chất 1 và tính chất 2 trong Mệnh đề 1 ta sẽ thu được thuật toán có chi phí không lớn hơn thuật toán ban đầu. Do đó ở đây ta chỉ cần xét trường hợp thuật toán phân tầng với số tầng tối đa. Điều này có nghĩa là với t như trong giả thiết định lý, ta xét thuật toán được phân thành s tầng.

Từ bổ đề 1 ta thấy rằng trong thuật toán phân tầng với số tầng tối đa là s thì:

Trường hợp $m > 4a$: Nếu tồn tại thuật toán tại tầng thứ j, $0 \le j \le s-1$ là thuật toán theo phương pháp phổ thông thì bằng cách sử dụng thuật toán theo phương pháp Karatsuba để thay thế ta sẽ nhận được một thuật toán mới hiệu quả hơn. Điều này có nghĩa là thuật toán phân tầng với số tầng tối đa dùng phương pháp Karatsuba cho mọi tầng sẽ có chi phí thấp nhất và theo tính chất 2 trong mệnh đề 1 thì các thuật toán trong tầng có thể hoán đổi vị trí cho nhau để được thuật toán với chi phí không đổi. Như vậy ta đã chứng minh được khẳng định (i) của định lý.

Với trường hợp $m \le 4a$: Theo bổ đề 1 thì thuật toán dùng cho tầng thấp nhất của thuật toán phân tầng với số tầng tối đa sẽ là thuật toán $M_S[t_i]$ với $i \in \{0,1, \dots s-1\}$ nào đó (do việc phân tầng không nhất thiết phải theo đúng thứ tự như phân tích của t) còn thuật toán sử dụng cho các tầng còn lại theo Karatsuba sẽ có chi phí thấp nhất trong các thuật toán phân tầng có tầng thấp nhất là nhân hai số t_i ký tự theo như các tính chất 5 và 2 trong mệnh đề 1.

Nếu $t_i = t_0$ thì ta nhận được khẳng định (ii).

Ngược lại thì $t_i < t_0$, do $s-1$ thuật toán ở các tầng trên đều là thuật toán theo phương pháp Karatsuba nên theo tính chất 2 trong mệnh đề 1 ta có thể hoán đổi thuật toán $M_K[t_0]$ về tầng ngay sát tầng thấp nhất. Khi đó hai tầng thấp nhất của thuật toán sẽ là $M_K[t_0]: M_S[t_i]$. Theo tính chất 3 trong mệnh đề 1, ta có

$$M_K(t_0): M_S(t_i) - M_K(t_i): M_S(t_0) = \frac{(t_0 t_i)(t_0 - t_i)}{2}(-m + 4a) \geq 0.$$

Điều này có nghĩa là thuật toán phân tầng có chi phí thấp nhất trong trường hợp này là thuật toán phân tầng mà sử dụng thuật toán $M_S[t_0]$ tại tầng thấp nhất và các tầng còn lại sẽ sử dụng thuật toán nhân theo phương pháp Karatsuba. Nói cách khác, thuật toán sử dụng để nhân hai số nguyên lớn có t ký tự trong trường hợp này là

$$M_K[t_{s-1}]: \ldots : M_K[t_1]: M_S[t_0].$$

Tổng hợp tất cả các lập luận ở trên, ta nhận được khẳng định đã phát biểu trong định lý.

Tiếp sau đây luận án đưa ra một kết quả bổ trợ để xác định chi phí một thuật toán nhân các số t ký tự với t có dạng $t = 2^r \times d$. Ký hiệu $(M_K[2])^r$ là thuật toán phân r tầng $M_K[2]: \ldots : M_K[2]$.

Mệnh đề 2. *Cho $d > 1$ là một số nguyên. Khi đó thuật toán $(M_K[2])^r: M[d]$ có chi phí tính toán là*

$$(M_K[2]))^r: M[d]) = 3^r M(d) + 8(3^r - 2^r)A(d), \qquad (2.13)$$

trong đó M(d) là chi phí của thuật toán nhân hai số d ký tự M[d] và A(d) là chi phí cộng hai số d ký tự.

Chứng minh. Ta sẽ chứng minh khẳng định này bằng phương pháp quy nạp.

Từ công thức (2.5) ta có

$$M_K(2): M(d) = 3M(d) + 8A(d)$$
$$= 3^1 M(d) + 8(3^1 - 2^1)A(d).$$

Như vậy công thức (2.13) đúng với $r = 1$.

Giả sử công thức (2.13) đúng với $r > 1$, ta sẽ chỉ ra công thức này cũng đúng với $r+1$. Thật vậy, vì $(M_K[2])^{r+1}: M[d] = M_K[2]: (M_K[2])^r: M[d])$ nên ta có chi phí

$$(M_K(2))^{r+1}: M(d) = M_K(2): \left(M_K(2)\right)^r: M(d)).$$
$$= 3(3^r M(d) + 8(3^r - 2^r)A(d) + 8 \cdot 2^r A(d).$$
$$= 3^{r+1} M(d) + 8(3 \cdot 3^r - 3 \cdot 2^r + 2^r)A(d).$$
$$= 3^{r+1} M(d) + 8(3^{r+1} - 2^{r+1})A(d).$$

Ta có điều cần chứng minh.

2.2.6. Tìm thuật toán nhân tối ưu cho một số giá trị t với giả thiết m = 2a.

Trong mục này trước hết luận án đưa ra thuật toán nhân tối ưu trong phạm vi các thuật toán được trình bày trong các mục trước và trên giả thiết **m** = 2**a** cho các số t ký tự 32-bít cho các giá trị t ≤ 16 và một số t riêng rẽ khác thường được sử dụng trong các ứng dụng mật mã.

Với mỗi giá trị t được xem xét, luận án duyệt toàn bộ các thuật toán đã được giới thiệu trong các mục trước, tính chi phí của chúng rồi chọn thuật toán tối ưu, ký hiệu M[t], là thuật toán có chi phí thấp nhất, ký hiệu M(t).

2.2.6.1 Ví dụ đánh giá chi phí của thuật toán nhân hiệu quả đối với một số kích thước thường gặp

Trong mục này luận án giới hạn việc xây dựng thuật toán nhân số lớn trên những thiết bị với những số nguyên 32-bit và với giả thiết tỷ số giữa phép nhân và phép cộng ρ_{32} = 2 (hay tương đương $m \approx 2a$). Khi đó theo định lý 1 thì thuật toán nhân hai số t ký tự hiệu quả nhất là thuật toán được đưa ra trong phần (ii).

Trong các ứng dụng mật mã ta thường gặp việc thực hiện phép nhân giữa các số có kích thước bội t của 32 như:

- $192 = 2 \times 3 \times 32, 224 = 7 \times 32, 256 = 2^3 \times 32, 384 = 2^2 \times 3 \times 32$ và $512 = 2^4 \times 32$ cho mật mã đường cong elliptic.
- $1536 = 2^4 \times 3 \times 32, 2048 = 2^6 \times 32, 4096 = 2^7 \times 32, 8192 = 2^8 \times 32$ và $15360 = 2^5 \times 3 \times 5 \times 32$ cho các hệ mật RSA hoặc các hệ mật dựa trên bài toán logarit rời rạc trên trường hữu hạn.

Ví dụ 1. Thuật toán hiệu quả nhất để nhân hai số 192 bit sẽ là thuật toán nhân hai số 2×3 ký tự 32 bit. Khi đó theo phần (ii) của định lý 1 sẽ là $M_K[2]: M_S[3]$. Từ các công thức (2.3), (2.5) và tỷ số ρ_{32} = 2 ta có

$$M_K(2): M_S(3) = \frac{2^2+2}{2} M_S(3) + 4(2^2-2)A(3)$$
$$= 3(3^2 m + 2(3^2-3)a) + 24a.$$
$$= 27m + 60a.$$
$$\approx 114a.$$

2.2.6.2. Thuật toán nhân tối ưu cho các số t ký tự

a) Với t = 1, …, 16 với giả thiết m = 2a.

Trong mục này ta ký hiệu quan hệ giữa các thuật toán phân tầng là dấu "**•**" thay cho ký tự "**:**". Dễ dàng ta có được hệ quả 1 và các số liệu tính toán được tại cột 4 bảng 2.2 sau.

Hệ quả 1. Nếu m = 2a, từ (2.3) và (2.13) ta có

$$M_S(t) = 2(2t^2 - t)\mathbf{a}. \quad (2.14)$$

$$\left(M_K(2)\right)^r : M(2) = 4(7 \times 3^r - 4 \times 2^r)\mathbf{a}. \quad (2.15)$$

*Bảng 2.2: Thuật toán nhân tối ưu cho các số t ký tự với t = 1, …, 16 với giả thiết **m** = 2**a**.*

M[t]	Thuật toán	Công thức	M(t) (**a**)
M[1]		$2(2\times1^2 - 1)$.	**2**
M[2]	$\mathbf{M_S[2]}$.	$2(2\times2^2 - 2)$.	**12**
M[3]	$\mathbf{M_S[3]}$. $M_K[1+2]$.	$2(2\times3^2 - 3)$. 2M(2) + M(1) + 4×3.	**30** 38
M[4]	$\mathbf{M_K[2]}\bullet\mathbf{M[2]}$. $M_K[2+1+1]$.	$4(7 \times 3^1 - 4 \times 2^1)$. 3M(2) + 3M(1) + 8×4.	**52** 58
M[5]	$\mathbf{M_S[5]}$. $M_K[2+3]$. $M_K[1+2+2]$.	$2(2\times5^2 - 5)$. 2M(3) + M(2) + 4×3. 5M(2) + 1M(1) + 8×5.	**90** 92 102
M[6]	$\mathbf{M_K[2]}\bullet\mathbf{M[3]}$.	3M(3) + 8A(3).	**114**
M[7]	$M_S[7]$. $\mathbf{M_K[3+4]}$. $M_K[3+2+2]$.	$2(2\times7^2 - 7)$. 2M(4) + M(3) + 4×7. 3M(3) + 3M(2) + 8×7.	182 **162** 172
M[8]	$(\mathbf{M_K[2]})^{\mathbf{2}}\bullet\mathbf{M[2]}$. $M_K[2+3+3]$.	$4(7 \times 3^2 - 4 \times 2^2)$. 5M(3) + M(2) + 8×8	**188** 226
M[9]	$\mathbf{M_K[3]}\bullet\mathbf{M[3]}$. $M_K[4+5]$.	6M(3) + 24A(3). 2M(5) + M(4) + 4×9.	**252** 268
	$M_K[2]\bullet M[5]$.	3M(5) + 8A(5).	380

M[10]	$\mathbf{M_K[4+3+3].}$	3M(4) + 3M(3) + 8×10.	**326**
M[11]	$M_S[11]$. $\mathbf{M_K[5+6]}$. $M_K[3+4+4]$.	$2(2\times11^2-11)$. 2M(6) + M(5) + 4×11. 5M(4) + M(3) + 8×11.	462 **368** 378
M[12]	$\mathbf{(M_K[2])^2 \bullet M[3].}$	(21) với r = 2, d = 3.	**390**
M[13]	$M_S[13]$. $\mathbf{M_K[6+7]}$. $M_K[5+4+4]$**.**	$2(2\times13^2-13)$. 2M(7) + M(6) + 4×13. 3M(5) + 3M(4) + 8×13.	650 **490** 530
M[14]	$\mathbf{M_K[2] \bullet M[7]}$. $M_K[4+5+5]$**.**	**3M(7) + 8A(7).** 5M(5) + M(4) +8×14.	**542** 614
M[15]	$M_K[3] \bullet M[5]$. $\mathbf{M_K[7+8]}$.	$6M_S(5) + 24A(5)$. 2M(8) + M(7) + 4×15	660 **598**
M[16]	$\mathbf{(M_K[2])^3 \bullet M[2]}$. $M_K[6+5+5]$.	$4(7\times3^3-4\times2^3)$. 3M(6) + 3M(5) + 8×16.	**628** 676

Theo kết quả thu được trong bảng trên thì thuật toán tối ưu M[t] là thuật toán được tô đậm tại cột thứ hai trong bảng này.

b) Thuật toán nhân tối ưu cho một số giá trị t

Các giá trị t mà luận án đề cập đến ở đây bao gồm t = 2^5, 3×2^4, 2^6, 2^7, 2^8 và 15×2^5 (tương ứng với các modulo 1024, 1536, 2048, 4096, 8192 và 15360 bít). Để xác định thuật toán tối ưu cho các giá trị t nêu trên, trước hết luận án thực hiện xác định thuật toán tối ưu cho một số giá trị t bổ trợ bao gồm t = 17, 21, 22, 28, 29, 42, 43, 85 và 86. Kết quả thu được cho bởi bảng 2.3 dưới đây.

Bảng 2.3: Thuật toán tối ưu cho t = 17, 21, 22, 28, 29, 42, 43, 85 và 86.

M[t]	Thuật toán	Công thức	M(t) (**a**)
M[17]	$M_S[17]$. $\mathbf{M_K[13+14]}$. $M_K[5+6+6]$**.**	$2(2\times17^2-17)$. 2M(9) + M(8) + 4×17 **5M(6) + M(5) + 8×17**.	1088 **760** 796
M[21]	$\mathbf{M_K[3] \bullet M[7]}$. $M_K[10+11]$.	6M(7) + 24A(7) 2M(11) + M(10) + 21×4	**1140** 1146
M[22]	$\mathbf{M_K[2] \bullet M[11]}$. $M_K[8+7+7]$.	3M(11) + 8A(11) 3M(8) + 3M(7) + 22×8	**1192** 1226
M[28]	$\mathbf{(M_K[2])^2 \bullet M[7]}$. $M_K[10+9+9]$.	$3^2M(7) + 8(3^2-2^2)A(7)$. 3M(10) + 3M(9) + 8×28	**1738** 1958
M[29]	$M_S[29]$. $\mathbf{M_K[14+15]}$. $M_K[9+10+10]$**.**	$2(2\times29^2-29)$. 2M(15) + M(14) + 4×29 **5M(10) + M(9) + 8×29**.	3306 **1854** 2114

M[42]	$\mathbf{M_K[2]{\bullet}M[21]}$.	3M(21) + 8A(21)	**3588**
M[43]	$M_S[43]$. $\mathbf{M_K[22+21]}$. $M_K[15+14+14]$.	$2(2\times43^2 - 43)$. 2M(22) + M(21) + 43×4 3M(15) + 3M(14) + 8×43.	7310 **3696** 3764
M[85]	$M_K[5]{\bullet}M[17]$. $M_K[42+43]$. $\mathbf{M_K[29+28+28]}$.	15M(17) + 80A(17). 2M(43) + M(42) + 4×85. 3M(29) + 3M(28) + 8×85.	12760 11320 **10980**
M[86]	$\mathbf{M_K[2]{\bullet}M[43]}$. $M_K[28+29+29]$.	3M(43) + 8A(43). **5M(29) + M(28) + 8×86**.	**11432** 11540

Cuối cùng các thuật toán tối ưu cho các giá trị t cần xét được cho trong bảng 2.4 sau đây.

Bảng 2.4: Thuật toán tối ưu cho t = 32, 48, 64, 128, 256 và 480.

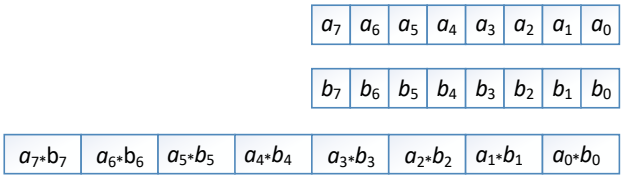
M[t]	Thuật toán	Công thức	M(t) (**a**)
M[32]	$\mathbf{(M_K[2])^4{\bullet}M[2]}$. $M_K[10+11+11]$.	$4(7\times3^4 - 4\times2^4)$. 5M(11) + M(10) + 8×32	**2012** 2422
M[48]	$\mathbf{(M_K[2])^4{\bullet}M[3]}$.	$3^4M(3) + 8(3^4 - 2^4)A(3)$.	**3809**
M[64]	$\mathbf{(M_K[2])^5{\bullet}M[2]}$. $M_K[22+21+21]$.	$4(7\times3^5 - 4\times2^5)$. 3M(22) + 3M(21) + 8×64	**6292** 7508
M[128]	$\mathbf{(M_K[2])^6{\bullet}M[2]}$. $M_K[42+43+43]$.	$4(7\times3^6 - 4\times2^6)$. 5M(43) + M(42) + 8×128	**19388** 23092
M[256]	$\mathbf{(M_K[2])^7{\bullet}M[2]}$. $M_K[86+85+85]$.	$4(7\times3^7 - 4\times2^7)$. 3M(86) + 3M(85) + 8×256	**59188** 69284
M[480]	$\mathbf{(M_K[2])^5{\bullet}M[15]}$.	$3^5M(15) + 8(3^5 - 2^5)A(15)$.	**147874**

2.3. Nhân phân tầng trong trường nhị phân trên vi xử lý ARMv7

2.3.1 Chỉ lệnh nhân nhị phân trong thành phần NEON trên vi xử lý ARMv7

Một sự tiện lợi rõ ràng của NEON SIMD cho mật mã đó là nó cung cấp nhiều không gian trong các thanh ghi, giảm số lần tải (load) và lưu (store) từ bộ nhớ. Thay vì 4 phép toán số học 32-bit thì với NEON SIMD có thể thực hiện chỉ với 1 phép toán số học 128-bit.

Phần này tập trung vào các lệnh NEON quan trọng được minh họa trong Hình 2.10. Lệnh VMULL có thể thực hiện một số phép nhân song song; phiên bản VMULL.P8 lấy đầu vào là hai vec-tơ 64-bit *A* và *B* của 8 đa thức nhị phân 8-bit và trả về vectơ đầu ra *C* 128-bit của tám đa thức nhị phân 16-bit, trong đó phần tử thứ *i* của *C* là tích của hai phần tử thứ *i* đầu vào.



Hình 2.3: Hoạt động của lệnh VMULL.P8.

Lệnh VEXT, đối với hai thanh ghi 64-bit và một số nguyên *i* tức thời, tạo ra một giá trị 64-bit, là sự kết hợp của 8*i* bit thấp của thanh ghi A và 64-8*i* bit cao của thanh ghi B. Lưu ý rằng nếu các đầu vào là cùng một thanh ghi thì lệnh VEXT sẽ chính là phép dịch vòng phải 8.*i* bit.

2.3.2. Tham số của đường cong NIST trên các trường nhị phân

Các đường cong elliptic của NIST trên $\mathbb{F}_{2^{283}}$ được liệt kê trong bảng 2.5. Ta sử dụng ký hiệu sau được sử dụng: Các phần tử của $\mathbb{F}_{2^{283}}$ được biểu diễn bằng một biểu diễn cơ sở đa thức với đa thức rút gọn $f(x)$, trong đó $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$. Ta ký hiệu $f(x) = x^{283} + r(x)$ với $r(x) = x^{12} + x^7 + x^5 + 1$. Đường cong elliptic E trên $\mathbb{F}_{2^{283}}$ được xác định bởi các hệ số $a, b \in \mathbb{F}_{2^{283}}$ của phương trình định nghĩa $y^2 + xy = x^3 + ax^2 + b$. Số điểm trên E được xác định trên $\mathbb{F}_{2^{283}}$ là nh, trong đó n là số nguyên tố và h được gọi là một đồng thừa số. Đường cong ngẫu nhiên trên $\mathbb{F}_{2^{283}}$ được ký hiệu là B-283, trong khi đường cong Koblitz trên $\mathbb{F}_{2^{283}}$ được ký hiệu là K-283.

Bảng 2.5: Tham số đường cong elliptic theo khuyến nghị của NIST trên $F_{2^{283}}$.

B-283: $a = 1, h = 2,$ b = 0x 027B680A C8B8596D A5A4AF8A 19A0303F CA97FD76 45309FA2 A581485A F6263E31 3B79A2F5 n = 0x 03FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFEF90 399660FC 938A9016 5B042A7C EFADB307
K-283: $a = 0, b = 1, h = 4,$ n = 0x 01FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFE9AE 2ED07577 265DFF7F 94451E06 1E163C61

2.3.3. Một phương pháp mới để nhân nhanh đa thức trên vi xử lý ARMv7

Trong mục này, luận án đề ứng dụng thuật toán nhân phân tầng đề xuất ở **mục 2.2** để nhân nhanh hai đa thức trên trường nhị phân GF(2^{283}) trên vi xử lý ARMv7. Phương pháp đề xuất mới ở đây (MNKv7) là kết hợp giữa thuật toán Karatsuba với bộ nhân cơ bản 64bit và 32bit trên nền vi xử lý ARMv7 để tạo thành thuật toán nhân đầy đủ

trên hai trường GF(2^{283}). Trong đề xuất này luận án không thực hiện cải tiến thuật toán giảm module mà sử dụng thuật toán giảm modulo theo thuật toán 2 trong [29].

2.3.3.1. Thuật toán Karatsuba với 5 khối (số hạng)

Gọi $t = \left\lfloor \frac{m+63}{64} \right\rfloor$, sử dụng $t-1$ khối 64-bit và 1 khối 32 bit để biểu diễn các phần tử của đa thức $a(x)$ và $b(x)$. Trong phần mềm, a được lưu trữ trong một mảng gồm $t-1$ các từ 64-bit và một từ 32bit (A_{t-1}): $A = (A_{t-1} \dots, A_2, A_1, A_0)$, trong đó bit ngoài cùng bên phải của A_0 là a_0 và s bit ngoài cùng bên trái của A_{t-1} không được sử dụng (luôn được thiết lập là 0)

Thuật toán 2.5: Nhân Karatsuba hai đa thức với 5 khối trên GF(2^{283})

Đầu vào: Hai đa thức $a(x)$ và $b(x)$ với $T = x^{64}$; A_i, B_i $(0 \le i < 4)$ là các khối đa thức con 64-bit; A_4, B_4 là các khối 32-bit.

Đầu ra: $c(x) = a(x) * b(x)$

1. $a(x) = A(T) = A_4T^4 + A_3T^3 + A_2T^2 + A_1T^1 + A_0T^0$
2. $b(x) = B(T) = B_4T^4 + B_3T^3 + B_2T^2 + B_1T^1 + B_0T^0$
3. $\text{v0} = (A_0 + A_1 + A_2 + A_3 + A_4) * (B_0 + B_1 + B_2 + B_3 + B_4)$
4. $\text{v1} = (A_0 + A_1 + A_3 + A_4) * (B_0 + B_1 + B_3 + B_4)$
5. $\text{v2} = (A_3 + A_4) * (B_3 + B_4)$
6. $\text{v3} = (A_1 + A_2 + A_4) * (B_1 + B_2 + B_4)$
7. $\text{v4} = (A_2 + A_4) * (B_2 + B_4)$
8. $\text{v5} = (A_4) * (B_4)$
9. $\text{v6} = (A_0 + A_2 + A_3) * (B_0 + B_2 + B_3)$
10. $\text{v7} = (A_3) * (B_3)$
11. $\text{v8} = (A_0 + A_2) * (B_0 + B_2)$
12. $\text{v9} = (A_2) * (B_2)$
13. $\text{v10} = (A_0 + A_1) * (B_0 + B_1)$
14. $\text{v11} = (A_1) * (B_1)$
15. $\text{v12} = (A_0) * (B_0)$
16. $c_0 = \text{v12}$

17. $c_1 = \text{v10} + \text{v11} + \text{v12}$
18. $c_2 = \text{v8} + \text{v9} + \text{v11} + \text{v12}$
19. $c_3 = \text{v0} + \text{v1} + \text{v4} + \text{v5} + \text{v6} + \text{v7} + \text{v9} + \text{v12}$
20. $c_4 = \text{v0} + \text{v2} + \text{v3} + \text{v5} + \text{v6} + \text{v7} + \text{v10} + \text{v11} + \text{v12}$
21. $c_5 = \text{v0} + \text{v1} + \text{v3} + \text{v5} + \text{v8} + \text{v9} + \text{v11} + \text{v12}$
22. $c_6 = \text{v4} + \text{v5} + \text{v7} + \text{v9}$
23. $c_7 = \text{v2} + \text{v5} + \text{v7}$
24. $c_8 = \text{v5}$

Khi đó, thực hiện phép nhân $a(x) * b(x)$, ta nhận được kết quả đầu ra là $c(x) = C(T) = c_8T^8 + c_7T^7 + c_6T^6 + c_5T^5 + c_4T^4 + c_3T^3 + c_2T^2 + c_1T^1 + c_0T^0$, trong đó mỗi c_i là khối 128-bit.

Nhật xét:

- Với phép nhân hai đa thức bậc không quá 283 trong thuật toán 2.5 thì số hạng A_4 và B_4 là hai đa thức bậc không quá 32 nên thuật toán 2.5 được xây dựng dựa trên 2 phép nhân cơ bản 64 bit (vmull64) và 32bit (vmull32).
- Chi phí phép nhân $a(x) * b(x)$ ở trên sử dụng 12 phép nhân 64-bit và 1 phép nhân 32bit. Việc xây dựng bộ nhân 64bit sẽ được trình bày ở mục 2.2.3.2
- Ở dòng thứ 8 của thuật toán 2.5, tính $\text{v5} = (A_4) * (B_4)$, do A_4 và B_4 là số hạng 32bit nên để tối ưu chỉ cần sử dụng phép nhân 32 bit là đủ (mặc dù ta vẫn có thể sử dụng phép nhân 64bit đối với phép toán này). Việc xây dựng bộ nhân 32-bit được trình bày ở trong mục 2.3.3.2.

2.3.3.2. Xây dựng phép nhân các đa thức bậc không quá 64 trên GF(2)

Trong phần này luận án xây dựng một thuật toán nhân hai đa thức bậc không quá 64 trên GF(2), ký hiệu là vmull64, trên các thiết bị ARMv7 hỗ trợ hai lệnh "vmull.p8" và "vext". Thuật toán thực hiện vmull64 được thực hiện theo phương pháp nhân phổ thông, tức là nếu:

$A = \sum_{i=0}^{7} a_i x^{8.i}$ hay $(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$ và

$B = \sum_{i=0}^{7} b x^{8.i}$ hay $(b_7, b, b_5, b, b_3, b, b_1, b_0)$

với a_i và b_i là các đa thức bậc nhỏ hơn 8 thì tích của *A* và *B* là *C* được xác định theo công thức dưới đây:

$$C = \sum_{i=0}^{i=14}\left(\sum_{j+k=i,0\le j,k<8} a_j.\, b_k\right) \quad (2.16)$$

Thuật toán được thực hiện qua các bước sau:

Bước 1. Lập bảng chứa tất cả các tích $c_{j,k} = a_j.\, b_k$ với $0 \le j,\ k < 8$.

Ký hiệu A_i, B_i là véc tơ thu được từ *A*, *B* bằng phép xoay vòng bên phải *i* vị trí và

$C_0 = \text{vmull.}\,\text{p8}(A, B)$, $C_1 = \text{vmull.}\,\text{p8}(A, B_1)$, $\text{C}_2 = \text{vmull.}\,\text{p8}(A_1, B)$,

$C_3 = \text{vmull.}\,\text{p8}(A, B_2)$, $C_4 = \text{vmull.}\,\text{p8}(A_2, B)$, $C_5 = \text{vmull.}\,\text{p8}(A, B_3)$,

$C_6 = \text{vmull.}\,\text{p8}(A_3, B)$ và $C_7 = \text{vmull.}\,\text{p8}(A, B_4)$

Ký hiệu $c_{j,k} = a_j b_k$ với $0 \le j,\ k < 8$ thì giá trị các tọa độ của các véc tơ C_i được cho trong bảng sau.

Bảng 2.6: Bảng chứa tất cả các tích $c_{j,k} = a_j.\, b_k$ với $0 \le j,\ k < 8$.

C_0	$c_{7,7}$	$c_{6,6}$	$\text{c}_{5,5}$	$c_{4,4}$	$c_{3,3}$	$c_{2,2}$	$c_{1,1}$	$c_{0,0}$
C_1	$c_{7,0}$	$c_{6,7}$	$c_{5,6}$	$c_{4,5}$	$c_{3,4}$	$c_{2,3}$	$c_{1,2}$	$c_{0,1}$
C_2	$c_{0,7}$	$c_{7,6}$	$c_{6,5}$	$c_{5,4}$	$c_{4,3}$	$\text{c}_{3,2}$	$\text{c}_{2,1}$	$c_{1,0}$
C_3	$c_{7,1}$	$c_{6,0}$	$c_{5,7}$	$c_{4,6}$	$c_{3,5}$	$c_{2,4}$	$c_{1,3}$	$c_{0,2}$
C_4	$c_{1,7}$	$c_{0,6}$	$c_{7,5}$	$c_{6,4}$	$c_{5,3}$	$c_{4,2}$	$c_{3,1}$	$c_{2,0}$
C_5	$\text{c}_{7,2}$	$\text{c}_{6,1}$	$\text{c}_{5,0}$	$\text{c}_{4,7}$	$\text{c}_{3,6}$	$\text{c}_{2,5}$	$\text{c}_{1,4}$	$\text{c}_{0,3}$
C_6	$\text{c}_{2,7}$	$\text{c}_{1,6}$	$\text{c}_{0,5}$	$\text{c}_{7,4}$	$\text{c}_{6,3}$	$\text{c}_{5,2}$	$\text{c}_{4,1}$	$\text{c}_{3,0}$
C_7	$\text{c}_{7,3}$	$\text{c}_{6,2}$	$\text{c}_{5,1}$	$\text{c}_{4,0}$	$\text{c}_{3,7}$	$\text{c}_{2,6}$	$\text{c}_{1,5}$	$\text{c}_{0,4}$

Dễ dàng nhận ra rằng các phần tử trong bảng 1 chứa toàn bộ các tích $a_j.\, b_k$ với $0 \le j,\ k < 8$.

Bước 2. Thực hiện tính *C* theo công thức (2.16).

Biến đổi các véc tơ C_i thành D_i sao cho chỉ số $j + k$ đúng bằng chỉ số mô tả trong bảng sau

Bảng 2.7: Biến đổi các véc tơ C_i thành D_i

	j+k	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E_0	D_0	$\text{c}_{7,7}$		$\text{c}_{6,6}$		$\text{c}_{5,5}$		$\text{c}_{4,4}$		$\text{c}_{3,3}$		$\text{c}_{2,2}$		$\text{c}_{1,1}$		$\text{c}_{0,0}$	

E_1	D_1		$c_{6,7}$	$c_{5,6}$	$c_{4,5}$	$c_{3,4}+c_{7,0}$	$c_{2,3}$	$c_{1,2}$	$c_{0,1}$	
	D_2		$c_{7,6}$	$c_{6,5}$	$c_{5,4}$	$c_{4,3}+c_{0,7}$	$c_{3,2}$	$c_{2,1}$	$c_{1,0}$	
E_2	D_3		$c_{5,7}$	$c_{4,6}$	$c_{3,5}+c_{7,1}$	$c_{2,4}+c_{6,0}$	$c_{1,3}$	$c_{0,2}$		
	D_4		$c_{7,5}$	$c_{6,4}$	$c_{5,3}+c_{1,7}$	$c_{4,2}+c_{0,6}$	$c_{3,1}$	$c_{2,0}$		
E_3	D_5			$c_{4,7}$	$c_{3,6}+c_{7,2}$	$c_{2,5}+c_{6,1}$	$c_{1,4}+c_{5,0}$	$c_{0,3}$		
	D_6			$c_{7,4}$	$c_{6,3}+c_{2,7}$	$c_{5,2}+c_{1,6}$	$c_{4,1}+c_{0,5}$	$c_{3,0}$		
E_4	D_7			$c_{3,7}+c_{7,3}$	$c_{2,6}+c_{6,2}$	$c_{1,5}+c_{5,1}$	$c_{0,4}+c_{4,0}$			

Cuối cùng ta có

$$A.B = D_0 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6 + D_7. \quad (2.17)$$

Nhận xét: Nếu ký hiệu $D_i = T(C_i)$, từ sự trùng nhau vị trí của các cặp C_{2j-1}, C_{2j} (j = 1, 2, 3) nên ta có $D_{2j-1} + D_{2j} = T\left(C_{2j-1} + C_{2j}\right)$ và điều này có nghĩa:

$$A.B = D_0 + T(C_1 + C_2) + T(C_3 + C_4) + T(C_5 + C_6) + D_7. \quad (2.18)$$

Cách thực hiện từng thành phần như sau:

- $E_0 = D_0 = C_0$.
- Ký hiệu $E_i l$, $E_i h$ là nửa thấp và nửa cao của E_i. Các cặp (D_1, D_2), (D_3, D_4), (D_5, D_6) và D_7 được tiến hành cùng một thao tác như sau:
  1. $E_j = D_{2j-1}$ xor D_{2j} (j = 1, 2, 3)
  2. $E_j l = E_j h$ xor $E_j l$ (nửa thấp của E_j bằng nửa cao của E_j XOR với nửa thấp của E_j)
  3. $E_j h = E_j h$ and $M[k]$. (ở đây $k = \left\lfloor \frac{j+1}{2} \right\rfloor$)
  4. $E_j l = E_j h$ xor $E_j l$.
  5. $E_j = E_j \ll 8.k$.

Với $M[k]$ là thanh ghi 64-bits, trong đó 16.k bít cao nhất bằng 0, các bit thấp còn lại bằng 1.

2.3.3.3. Xây dựng phép nhân các đa thức bậc không quá 32 trên GF(2).

Trong phần này luận án xây dựng một thuật toán nhân hai đa thức bậc không quá 32 trên GF(2), ký hiệu là vmull32, trên các thiết bị ARMv7 có hỗ trợ hai lệnh "vmull.p8" và "vext". Thuật toán thực hiện vmull32 được thực hiện theo phương pháp nhân phổ thông, tức là nếu:

$AL = \sum_{i=0}^{3} a_i x^{8.i}$ hay (a_3, a_2, a_1, a_0) và

$BL = \sum_{i=0}^{3} bx^{8.i}$ hay (b_3, b, b_1, b_0)

với a_i và b_i là các đa thức bậc nhỏ hơn 4 thì tích của *AL* và *BL* là *C* được xác định theo công thức dưới đây:

$$C = \sum_{i=0}^{i=6}\left(\sum_{j+k=i, 0\le j,k<8} a_j . b_k\right) \quad (2.19)$$

Thuật toán được thực hiện qua các bước sau:

Bước 1. Lập bảng chứa tất cả các tích $c_{j,k} = a_j . b_k$ với $0 \le j,\ k < 4$.

Ký hiệu A là vector 64 bit, do vậy AL được biểu diễn thông qua A là

$(0,0,0,0, a_3, a_2, a_1, a_0)$

Ký hiệu B là vector 64 bit, do vậy BL được biểu diễn thông qua B là

$(0,0,0,0, b_3, b_2, b_1, b_0)$

Gọi A_i, B_i thu được từ *A*, *B* (là véc tơ 8 phần tử, mỗi phần tử 8 bit) bằng phép xoay vòng bên phải *i* vị trí và

$C_0 = \text{vmull. p8}(A, B)$, $C_1 = \text{vmull. p8}(A, B_1)$, $C_2 = \text{vmull. p8}(A_1, B)$,

$C_3 = \text{vmull. p8}(A, B_2)$, $C_4 = \text{vmull. p8}(A_2, B)$, $C_5 = \text{vmull. p8}(A, B_3)$,

$C_6 = \text{vmull. p8}(A_3, B)$

Ký hiệu $c_{j,k} = a_j b_k$ với $0 \le j,\ k < 8$ thì giá trị các tọa độ của các véc tơ C_i được cho trong bảng sau.

Bảng 2.8: Bảng chứa tất cả các tích $c_{j,k} = a_j . b_k$ với $0 \le j,\ k < 8$.

C_0	$c_{3,3}$	$c_{2,2}$	$c_{1,1}$	$c_{0,0}$
C_1	$c_{3,4}$	$c_{2,3}$	$c_{1,2}$	$c_{0,1}$
C_2	$c_{4,3}$	$c_{3,2}$	$c_{2,1}$	$c_{1,0}$
C_3	$c_{3,5}$	$c_{2,4}$	$c_{1,3}$	$c_{0,2}$
C_4	$c_{5,3}$	$c_{4,2}$	$c_{3,1}$	$c_{2,0}$
C_5	$c_{3,6}$	$c_{2,5}$	$c_{1,4}$	$c_{0,3}$
C_6	$c_{6,3}$	$c_{5,2}$	$c_{4,1}$	$c_{3,0}$

Do các phần tử $c_{j,k}$ với $4 \le j, k < 8$ là bằng 0, do vậy ta có bảng chứa tất cả các tích $c_{j,k} = a_j . b_k$ với $0 \le j,\ k < 4$.

Bảng 2.9: Bảng chứa tất cả các tích $c_{j,k} = a_j.b_k$ với $0 \le j,\ k < 4$.

C_0	$c_{3,3}$	$c_{2,2}$	$c_{1,1}$	$c_{0,0}$
C_1	0	$c_{2,3}$	$c_{1,2}$	$c_{0,1}$
C_2	0	$c_{3,2}$	$c_{2,1}$	$c_{1,0}$
C_3	0	0	$c_{1,3}$	$c_{0,2}$
C_4	0	0	$c_{3,1}$	$c_{2,0}$
C_5	0	0	0	$c_{0,3}$
C_6	0	0	0	$c_{3,0}$

Bước 2. Thực hiện tính *C* theo công thức (2.19).

Biến đổi các véc tơ C_i thành D_i sao cho chỉ số $j + k$ đúng bằng chỉ số mô tả trong bảng sau

Bảng 2.10: Biến đổi các véc tơ C_i thành D_i trong 2 đa thức bậc không quá 32

	$j+k$	7	6	5	4	3	2	1	0
E_0	D_0	$c_{3,3}$		$c_{2,2}$		$c_{1,1}$		$c_{0,0}$	
E_1	D_1		$c_{2,3}$		$c_{1,2}$		$c_{0,1}$		
	D_2		$c_{3,2}$		$c_{2,1}$		$c_{1,0}$		
E_2	D_3			$c_{1,3}$		$c_{0,2}$			
	D_4			$c_{3,1}$		$c_{2,0}$			
E_3	D_5				$c_{0,3}$				
	D_6				$c_{3,0}$				

Cuối cùng ta có

$$A.B = D_0 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6. \quad (2.20)$$

Nhận xét: Nếu ký hiệu $D_i = T(C_i)$, từ sự trùng nhau vị trí của các cặp C_{2j-1}, C_{2j} (j = 1, 2, 3) nên ta có $D_{2j-1} + D_{2j} = T\left(C_{2j-1} + C_{2j}\right)$ và điều này có nghĩa

$$A.B = D_0 + T(C_1 + C_2) + T(C_3 + C_4) + T(C_5 + C_6). \quad (2.21)$$

Cách thực hiện từng thành phần như sau:

- $E_0 = D_0 = C_0$.
- Ký hiệu $E_i l$, $E_i h$ là nửa thấp và nửa cao của E_i. Các cặp (D_1, D_2), (D_3, D_4), (D_5, D_6) được tiến hành cùng một thao tác như sau:

1. $E_j = D_{2j-1}$ xor D_{2j} (j = 1, 2, 3)
2. $E_j = E_j \ll 8.k.$

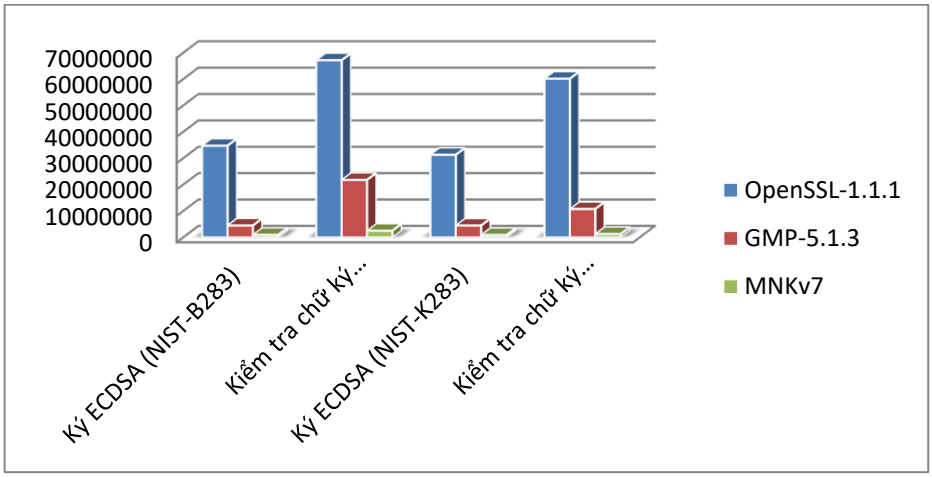
Nhật xét: Thuật toán đề xuất ở đây hiệu quả hơn so với thuật toán nhân 64 bit trình bày ở mục trên, thuật toán này giảm đi 3 phép tính (hai phép xor và một phép and) ở mỗi cặp (D_1, D_2), (D_3, D_4), (D_5, D_6) và giảm đi 1 phép tính vmull.p8.

2.3.4 Thực nghiệm, đánh giá thuật toán đề xuất

Luận án sử dụng ngôn ngữ Assembly/C và thư viện mật mã GMP để cài đặt thuật toán nhân 64bit đã đề xuất. Sau đó kết hợp giữa thuật toán nhân 64bit, thuật toán 32 bit cơ bản với thuật toán Karasuba để thực hiện phép nhân hai đa thức đầy đủ trên $GF(2^{283})$ – thuật toán đầy đủ này gọi là MNKv7. Để kiểm tra hiệu quả của phép nhân đã đề xuất trong mục 2.3.3, luận án sử dụng thuật toán mật mã ECDSA. Các chương trình được biên dịch và chạy trên hai nền tảng thiết bị nhúng là: Xilinx Zynq-7000 SoC ZC702 (Linux) [49] và IMX6Q-SABRE (Linux) [50]. Ngoài ra, để đánh giá được tốc độ thực hiện của thuật toán cải tiến, luận án có so sánh sự thực thi giữa thuật toán cải tiến với sự thực thi tương ứng khi sử dụng các hàm mặc định của thư viện GMP [61] và thư viện OpenSSL [55]. Các kết quả lần lượt được thể hiện trong Bảng 2.11, Bảng 2.12, Hình 2.7 và Hình 2.8 dưới đây.

Bảng 2.11: Kết quả đánh giá trên Xilinx Zynq-7000 SoC ZC702 (Linux)

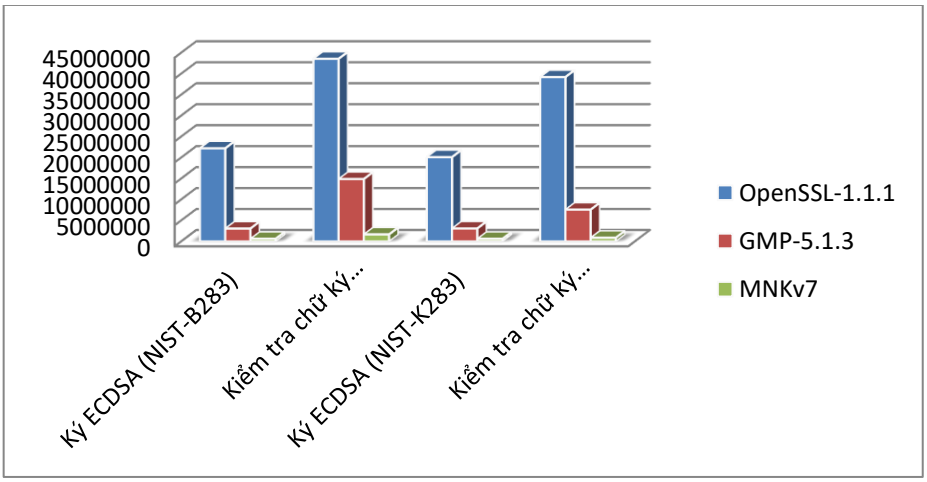
Thuật toán	OpenSSL-1.1.1	GMP-5.1.3	MNKv7
Ký ECDSA (NIST-B283)	34771922 ns	4521351 ns	973160 ns
Kiểm tra chữ ký ECDSA (NIST-B283)	67183606 ns	21786237 ns	2517141 ns
Ký ECDSA (NIST-K283)	31328528 ns	4494423 ns	913152 ns
Kiểm tra chữ ký ECDSA (NIST-K283)	60183606 ns	10760133 ns	1273322 ns



Hình 2.4. Hiệu quả của thuật toán đề xuất trên phần cứng Xilinx Zynq-7000 SoC ZC702 (Linux)

Bảng 2.12: Kết quả đánh giá trên IMX6Q-SABRE (Linux)

Thuật toán	OpenSSL-1.1.1	GMP-5.1.3	MNKv7
Ký ECDSA (NIST-B283)	22207513 ns	3066908 ns	627888 ns
Kiểm tra chữ ký ECDSA (NIST-B283)	43556776 ns	14889229 ns	1669725 ns
Ký ECDSA (NIST-K283)	20099884 ns	3090268 ns	616321 ns
Kiểm tra chữ ký ECDSA (NIST-K283)	39185785 ns	7582955 ns	870943 ns



Hình 2.5. Hiệu quả của thuật toán đề xuất trên phần cứng IMX6Q-SABRE (Linux)

2.4. Nhân phân tầng trong trường nhị phân trên vi xử lý ARMv8

Trong mục này, luận án đề xuất thuật toán nhân phân tầng trên trường nhị phân GF(2^{283}), GF(2^{409}), GF(2^{571}) dựa trên thuật toán nhân phân tầng đề xuất ở **mục 2.2** ở trên, bằng việc kết hợp giữa ba thành phần chính: thuật toán Karatsuba, bộ nhân 128-bit và bộ nhân 192-bit trên nền vi xử lý ARMv8 64-bit (AArch64) và 32-bit (AArch32). Gọi là phân tầng do: tầng thứ nhất là các bộ nhân cơ bản 128-bit và 192-bit lập trình bằng ngôn ngữ máy (assembly) được cài đặt theo **thuật toán nhân phổ thông**; tầng thứ 2, 3.. sẽ sử dụng hai bộ nhân cơ bản (128-bit và 192-bit) trong thuật toán **karatsuba** để thực hiện phép nhân với số hạng có độ dài thích hợp. Một số đặc điểm của thuật toán đề xuất là:

- Thuật toán đề xuất được coi là một trong những thuật toán nhân trong trường nhị phân hiệu quả nhất trên vi xử lý ARMv8
- Đơn giản trong triển khai cài đặt và tích hợp. Thành phần reduced modulo sẽ sử dụng thuật toán 2 trong [29].
- Là đề xuất đầu tiên tích hợp đầy đủ phép nhân trong trường nhị phân sử dụng thành phần NEON vào trong ứng dụng ECDSA, ECDH và ECMQV trên vi xử lý ARMv8 32-bit (AArch32) và 64-bit (AArch64)
- Đánh giá, so sánh giữa thuật toán đề xuất với thuật toán được coi là nhanh nhất LD [29] trên cùng một nền tảng phần cứng (Samsung Galaxy Tab S2 cho AArch32, NXP IMX8M KIT cho AArch64): Kết quả, thuật toán đề xuất nhanh hơn so với LD lần lượt là 4; 5; 5 lần trên trường GF(2^{283}), GF(2^{409}), GF(2^{571}).

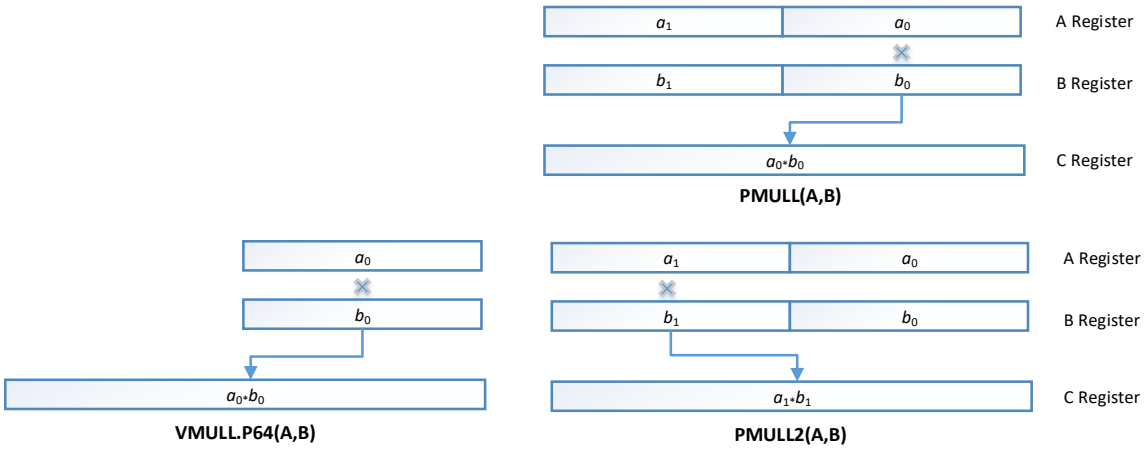
2.4.1 Hỗ trợ nhân đa thức nhị phân trên vi xử lý ARMv8

2.4.1.1 Bộ nhân đa thức trên vi xử lý ARMv8

Khi sử dụng ARMv7, chỉ lệnh VMULL.P8 rất quan trọng để cài đặt hiện quả phép nhân trên trường nhị phân, tham khảo [17]. Đầu vào của chỉ lệnh này là các thanh ghi NEON 64-bit, ở đó mỗi một thanh ghi được coi là tám đa thức nhị phân 8-bit và đầu ra là một thanh ghi NEON 128-bit và được coi là 8 đa thức nhị phân 16-bit chứa kết quả là phép nhân của tám cặp.

AArch32 cung cấp chỉ lệnh mới là VMULL.P64, nó nhận đầu vào là hai thanh ghi NEON 64-bit và được coi là đa thức nhị phân 64-bit, đầu ra được lưu trong một thanh ghi NEON 128-bit là kết quả của phép nhân nhị phân hai đa thức. Bởi vậy, chỉ cần một chỉ lệnh ta có thể thực hiện phép nhân nhị phân 64 x 64-bit.

AArch64 cung cấp hai chỉ lệnh là PMULL và PMULL2, cả hai chỉ lệnh đều thực hiện phép nhân 64x64-bit. Trong cả hai chỉ lệnh, đầu vào là các thanh ghi 128-bit, sự khác nhau giữa chúng đó là: trong PMULL sẽ sử dụng thành phần 64-bit thấp làm đầu vào, trong khi đó PMULL2 sử dụng thành phần 64-bit cao là đầu vào. Hoạt động của PMULL và PMULL2 được minh họa trong hình 2.13.



Hình 2.6: Hoạt động của VMULL.P64, PMULL và PMULL2 trên vi xử lý ARMv8

2.4.1.2. Đánh giá tỉ số giữa chi phí giữa chỉ lệnh nhân và chỉ lệnh cộng trên thành phần NEON

Theo mục 2.1.4.2 (tỷ số giữa chỉ lệnh nhân và chỉ lệnh cộng trong trường nhị phân), đối với vi xử lý ARMv7/v8 có tích hợp thành phần NEON. Tỷ số thời gian thực thi giữa VMULL.P8 và VEOR:

$$\frac{VMULL.P8}{VEOR} = 1.$$

Kiến trúc ARMv8 là kiến trúc 64-bit. Các dòng vi xử lý Cortex-A53 và Cortex-A57 có cơ chế pipeline tương tự như Cortex-A7 và Cortex-A15 tương ứng (xem mục 1.2.3 - chương 1). Do vậy,

$$\frac{PMULL}{EOR} = 1.$$

2.4.2. Xây dựng thuật toán nhân đa thức nhị phân phân tầng trên vi xử lý ARMv8

Trong mục này, luận án đề xuất thuật toán nhân phân tầng trên trường nhị phân GF(2^{283}), GF(2^{409}), GF(2^{571}) bằng việc kết hợp giữa ba thành phần chính (gọi là MKNv8): Thuật toán Karatsuba, bộ nhân 128-bit và bộ nhân 192-bit trên nền vi xử lý ARMv8 32-bit (AArch32) và 64-bit (AArch64). Phương pháp nhân phân tầng trong trường nhị phân được mô tả trong mục 2.2:

- Tầng thứ nhất là các bộ nhân cơ bản 128-bit và 192-bit được lập trình bằng ngôn ngữ máy (assembly) theo phương pháp **nhân phổ thông**;
- Tầng thứ 2 được triển khai bằng ngôn ngữ bậc cao (C/C++): sử dụng bộ nhân cơ bản (128-bit và 192-bit) theo phương pháp **nhân karatsuba** ở tầng khác nhau để thực hiện phép nhân các số hạng có độ dài thích hợp trên trường GF(2^{283}), GF(2^{409}), GF(2^{571}).

2.4.2.1 Đề xuất thuật toán nhân phân tầng

Bây giờ luận án sẽ áp dụng thuật toán phân tầng để thực hiện phép nhân trong các trường hữu hạn $\mathrm{GF}(2^{283}), \mathrm{GF}(2^{409})$ và $\mathrm{GF}(2^{571})$ trên vi xử lý ARMv8.

- Kết quả đầu tiên (và cũng là quan trọng nhất) đó là tham số $\frac{m}{a}$ (tỷ lệ chi phí cho phép nhân và phép cộng cơ sở với giả thiết chi phí thực hiện lệnh "veor" và "vmull.p64"; tương tự "eor" và "pmull" hoặc "pmull2" là như nhau) trong phép cộng và nhân đa thức sẽ bằng 1 trong AArch32 và AArch64.
- Từ $\frac{m}{a} = 1$ (xem đánh giá tỷ số từ mục 2.4.1.2), áp dụng đánh giá (i) của Định lý 1 trong mục 2.2.5 ta có nếu phép nhân được phân thành nhiều tầng thì tầng thấp nhất được thực hiện theo phương pháp nhân phổ thông, còn các tầng tiếp sau được thực hiện theo phương pháp Karatsuba.

Giới hạn cho các thuật toán nhân nhị phân ở đây là hai thuật toán, một theo phương pháp nhân phổ thông (ký hiệu là $M_S[t_i]$) và một theo phương pháp Karatsuba ($M_K[t_i]$) cho tầng thứ t_i. Phân tích cụ thể cho các trường cần khảo sát như sau:

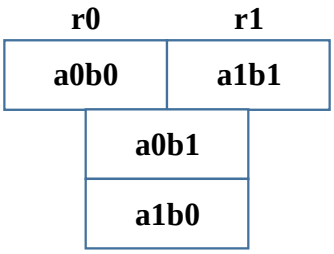
- Đối với GF(2^{283}) thì mỗi đa thức bậc ≤ 282 được tách thành 5 đa thức bậc 63. Khi này phân tầng tốt nhất có thể được xét đến là 2×3 ký hiệu là $M_K[2]: M_S[3]$ hoặc 3x2 ký hiệu là $M_K[3]: M_S[2]$
- Đối với GF(2^{409}) thì mỗi đa thức bậc ≤ 408 được tách thành 7 đa thức bậc 63. Khi này phân tầng tốt nhất là 2×2×2, ký hiệu là $M_K[2]: M_K[2]: M_S[2]$
- Đối với GF(2^{571}) thì mỗi đa thức bậc ≤ 570 được tách thành 9 đa thức bậc 63. Khi này phân tầng tốt nhất là 3×3, ký hiệu là $M_K[3]: M_S[3]$.

Như vậy, để thực hiện phép nhân phân tầng trong trường nhị phân GF(2^{283}), GF(2^{409}) và GF(2^{571}) thì ta cần xây dựng 2 bộ nhân là: bộ nhân 128-bit và bộ nhân 192-bit bằng phương pháp nhân phổ thông. Các bộ nhân này được trình bày ở các mục tiếp theo.

2.4.2.2 Xây dựng bộ nhân cơ bản trên vi xử lý ARMv8 32-bit (AArch32)

a) Thuật toán nhân khối 128-bit trên vi xử lý ARMv8 32-bit

Để nhân nhị phân hai đa thức 128-bit trên vi xử lý ARMv8 32-bit sử dụng chỉ lệnh vmull.p64. Đầu vào hai đa thức *a* và *b* có bậc 127 được lưu vào hai thanh ghi 128-bit tương ứng. Cụ thể, đa thức *a* được lưu vào thanh ghi là a và đa thức b được lưu vào thanh ghi b. Kết quả của phép nhân hai đa thức *a* và *b* là *r* (r0, r1) theo phương pháp nhân phổ thông được mô tả trong hình dưới sử dụng 3 thanh ghi tạm là t0, t1.



Hình 2.7: Bộ nhân 128-bit theo phương pháp nhân phổ thông

Thuật toán 2.6: Macro (aarch32_mul128_p64) nhân hai đa thức 128-bit trên AARCH32

Đầu vào: 128-bit đa thức *a* (al, ah), *b* (bl, bh); Các thanh ghi tạm 128-bit t0, t1

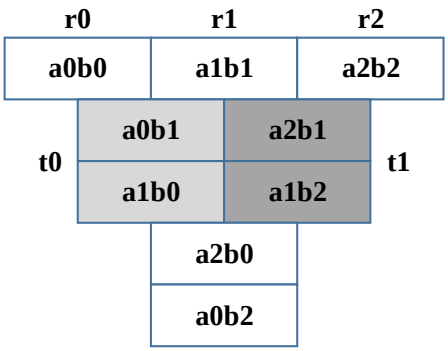
Đầu ra: Trả về kết quả *r* (r0q, r1q) có độ dài 256-bit (phần thấp r0q, phần cao là r1q)

```
1: vmull.p64 \r0q, \al, \bl      //r0 = a0 * b0
2: vmull.p64 \r1q, \ah, \bh      //r1 = a1 * b1
3: vmull.p64 \t0q, \ah, \bl      //t0 = a1 * b0
```

```
4: vmull.p64 \t1q, \al, \bh      //t0 = a0 * b1
5: veor  \t1q, \t0q              //(a1*b0)+(a0+b1)
6: veor  \r0h, \t1l              //r0h = r0h + t1l
7: veor  \r1l, \t1h              //r1l = r1l + t1h
```

b) Thuật toán nhân khối 192-bit trên vi xử lý ARMv8 32-bit

Đầu vào hai đa thức *a* và *b* có bậc nhỏ hơn 192-bit được phân thành 3 thành phần có bậc 64-bit, mỗi thành phần 64-bit được lưu vào một phần của thanh ghi 128-bit tương ứng. Cụ thể, đa thức *a* được lưu vào 3 thanh ghi là a0l, a0h, a1l và đa thức *b* được lưu vào 3 thanh ghi b0l, b0h, b1l. Kết quả của phép nhân hai đa thức *a* và *b* là *r* (r0, r1, r2) theo phương pháp nhân phổ thông được mô tả trong hình 2.15 dưới sử dụng 3 thanh ghi tạm là t0, t1, t2.



Hình 2.8: Bộ nhân 192-bit theo phương pháp phổ thông

Thuật toán 2.7: Macro (aarch32_mul192_p64) nhân hai đa thức 192-bit trên AARCH32

Đầu vào: 192-bit đa thức a (a0l, a0h, a1l), b (b0l, b0h, b1l); Các thanh ghi tạm 128-bit t0, t1, t2

Đầu ra: Trả về kết quả r (r0q, r1q, r2q) có độ dài 384-bit (phần thấp r0q, phần cao là r2q)

```
1: vmull.p64 \r0q, \a0l, \b0l      //r0 = a0*b0
2: vmull.p64 \t0q, \a0l, \b0h      //t0 = a0*b1
3: vmull.p64 \r1q, \a0l, \b1l      //r1 = a0*b2
4: vmull.p64 \t2q, \a0h, \b0l      //t2 = a1*b0
5: veor  \t0q, \t0q, \t2q          //t0 = a0*b1 + a1*b0
6: vmull.p64 \t2q, \a0h, \b0h      //t2 = a1*b1
7: veor  \r1q, \r1q, \t2q          //r1 = a0*b2 + a1*b1
8: vmull.p64 \t1q, \a0h, \b1l      //t1 = a1*b2
```

```
9: vmull.p64 \t2q, \a1l, \b0l        //t2 = a2*b0
10: veor \r1q, \r1q, \t2q            //r1 = a0*b2 + a1*b1 + a2*b0
11: vmull.p64 \t2q, \a1l, \b0h       //t2 = a2*b1
12: veor \t1q, \t1q, \t2q            //t1 = a1*b2 + a2*b1
13: vmull.p64 \r2q, \a1l, \b1l       //r2 = a2*b2
14: veor \r0h, \r0h, \t0l            //Thực hiện cộng vào các vị trí thích hợp theo hình 2.15
15: veor \r1l, \r1l, \t0h
16: veor \r1h, \r1h, \t1l
17: veor \r2l, \r2l, \t1h
```

2.4.2.3 Xây dựng bộ nhân cơ bản trên vi xử lý ARMv8 64-bit (AArch64)

a) Thuật toán nhân hai đa thức 128-bit trên AARCH64

Đầu vào hai đa thức *a* và *b* có bậc 127 được lưu vào hai thanh ghi 128-bit tương ứng. Cụ thể, đa thức *a* được lưu vào thanh ghi là *a* và đa thức *b* được lưu vào thanh ghi *b*. Kết quả của phép nhân hai đa thức *a* và *b* là *r* (r0, r1) theo phương pháp nhân phổ thông được mô tả trong hình 3 sử dụng 3 thanh ghi tạm là t0, t1, z.

Thuật toán 2.8: Macro (aarch64_mul128_p64) nhân hai đa thức 128-bit

Đầu vào: 128-bit đa thức *a*, *b*; Các thanh ghi tạm 128-bit t0, t1, z (zero)

Đầu ra: Trả về kết quả r có độ dài 256-bit (phần thấp r0, phần cao là r1)

```
1: pmull \r0\().1q, \a\().1d, \b\().1d          //r0 = a0 * b0
2: pmull2 \r1\().1q, \a\().2d, \b\().2d         //r1 = a1 * b1
3: ext \t0\().16b, \b\().16b, \b\().16b, #8     //t0 = b0 || b1
4: pmull \t1\().1q, \a\().1d, \t0\().1d         //t1 = a0 * b1
5: pmull2 \t0\().1q, \a\().2d, \t0\().2d        //t0 = a1 * b0
6: eor \t0\().16b, \t0\().16b, \t1\().16b       //t0 = (a0 * b1) + (a1 * b0)
7: ext \t1\().16b, \z\().16b, \t0\().16b, #8    //t1 = 0 || t0L
8: eor \r0\().16b, \r0\().16b, \t1\().16b       //r0 = a0 * b0 + (0 || t0L)
9: ext \t1\().16b, \t0\().16b, \z\().16b, #8    //t1 = t0H || 0
10: eor \r1\().16b, \r1\().16b, \t1\().16b      //r1 = a1*b1 + (t0H || 0)
```

b) Thuật toán nhân hai đa thức 192-bit trên AARCH64

Đầu vào hai đa thức *a* và *b* có bậc nhỏ hơn 192-bit được phân thành 3 thành phần có bậc 64-bit, mỗi thành phần 64-bit được lưu vào thanh ghi 128-bit tương ứng. Cụ thể, đa thức *a* được lưu vào 3 thanh ghi là a0, a1, a2 và đa thức *b* được lưu vào 3 thanh

ghi b0, b1, b2. Kết quả của phép nhân hai đa thức *a* và *b* là r (r0, r1, r2) theo phương pháp nhân phổ thông được mô tả trong hình 4 sử dụng 3 thanh ghi tạm là t0, t1, t2.

Thuật toán 2.9: Macro (aarch64_mul192_p64) nhân hai đa thức 192-bit trên AARCH64

Đầu vào: 192-bit đa thức *a* (a0, a1, a2), *b* (b0, b1, b2); Các thanh ghi tạm 128-bit t0, t1, t2

Đầu ra: Trả về kết quả *r* (r0, r1, r2) có độ dài 384-bit (phần thấp r0, phần cao là r2)

```
1: pmull \r0\().1q, \a0\().1d, \b0\().1d          //r0 = a0 * b0
2: pmull \t0\().1q, \a0\().1d, \b1\().1d          //t0 = a0 * b1
3: pmull \r1\().1q, \a0\().1d, \b2\().1d          //r1 = a0 * b2
4: pmull \t2\().1q, \a1\().1d, \b0\().1d          //t2 = a1 * b0
5: eor \t0\().16b, \t0\().16b, \t2\().16b         //t0 = a0*b1 + a1*b0
6: pmull \t2\().1q, \a1\().1d, \b1\().1d          //t2 = a1*b1
7: eor \r1\().16b, \r1\().16b, \t2\().16b         //r1 = a0*b2 + a1*b1
8: pmull \t1\().1q, \a1\().1d, \b2\().1d          //t1 = a1*b2
9: pmull \t2\().1q, \a2\().1d, \b0\().1d          //t2 = a2*b0
10: eor \r1\().16b, \r1\().16b, \t2\().16b        //r1 = a0*b2 + a1*b1 + a2*b0
11: pmull \t2\().1q, \a2\().1d, \b1\().1d         //t2 = a2*b1
12: eor \t1\().16b, \t1\().16b, \t2\().16b        //t1 = a1*b2 + a2*b1
13: pmull \r2\().1q, \a2\().1d, \b2\().1d         //r2 = a2*b2
14: movi \t2\().16b, #0                           //t2 = 0
15: ext \t2\().16b, \t2\().16b, \t0\().16b, #8    //t2 = 0 || t0L
16: ext \t0\().16b, \t0\().16b, \t1\().16b, #8    //t0 = t0H || t1L
17: ext \t1\().16b, \t1\().16b, \t2\().16b, #8    //t1 = t1H || 0
18: eor \r0\().16b, \r0\().16b, \t2\().16b        //r0 = r0 + (0 || t0L)
19: eor \r1\().16b, \r1\().16b, \t0\().16b        //r1 = r1 + (t0H || t1L)
20: eor \r2\().16b, \r2\().16b, \t1\().16b        //r2 = r2 + (t1H || 0)
```

2.4.2.4 Nhân Karatsuba

a) Nhân Karatsuba trên trường GF(2^{283})

Đối với GF(2^{283}) thì mỗi đa thức bậc ≤ 282 được tách thành 5 đa thức bậc 63. Luận án sử dụng thuật toán phân tầng xét đến là 2×3 ký hiệu là $M_K[2]: M_S[3]$. Như vậy, áp dụng thuật toán nhân Karatsuba ở đây sẽ là: Nhân hai đa thức, mỗi đa thức có hai khối để biểu diễn các phần tử của đa thức (mỗi khối có độ dài 192-bit).

Trên vi xử lý ARMv8, các hệ số của đa thức a (tương tự đa thức b) được lưu trong một mảng gồm 6 phần tử 64-bit là: $A = (A_5 \dots , A_2, A_1, A_0)$, trong đó bit ngoài cùng bên phải của $A[0]$ là a_0 và s bit ngoài cùng bên trái của A_4 không được sử dụng (luôn được thiết lập là 0), phần tử A_5 được thiết lập bằng 0

Thuật toán 2.10: Nhân Karatsuba hai đa thức với 2 khối (192-bit) trên GF(2^{283})

Đầu vào: Hai đa thức $a(x)$ và $b(x)$ với $T = x^{192}$; A_i, B_i $(0 \leq i < 2)$ là các khối đa thức con 192-bit;

Đầu ra: $c(x) = a(x) * b(x)$

25. $a(x) = A(T) = A_1T + A_0$
26. $b(x) = B(T) = B_1T + B_0$
27. t0 $= A_1 * B_1$
28. t1 $= (A_0 + A_1) * (B_0 + B_1)$
29. t2 $= A_0 * B_0$
30. c0 $=$ t2
31. c1 $=$ t0 + t1 + t2
32. c2 $=$ t0
33. $c(x) = C(T) = c_2T^2 + c_1T^1 + c_0T^0$

b) Nhân Karatsuba trên trường GF(2^{409})

Đối với GF(2^{409}) thì mỗi đa thức bậc ≤ 408 được tách thành 7 đa thức bậc 63. Khi này phân tầng tốt nhất là 2×2×2, ký hiệu là $M_K[2]: M_K[2]: M_S[2]$. Như vậy, sẽ phải áp dụng thuật toán Karatsuba với 2 mức đệ quy.

Mức 1: Tương tự như thuật toán 5 nhưng áp dụng với $T = x^{128}$

Mức 2: Tương tự như thuật toán 5 nhưng áp dụng với $T = x^{256}$

c) Nhân Karatsuba trên trường GF(2^{571})

Đối với GF(2^{571}) thì mỗi đa thức bậc ≤ 570 được tách thành 9 đa thức bậc 63. Khi này phân tầng tốt nhất là 3×3, ký hiệu là $M_K[3]: M_S[3]$. Như vậy, áp dụng thuật toán nhân Karatsuba ở đây sẽ là: Nhân hai đa thức, mỗi đa thức có ba khối để biểu diễn các phần tử của đa thức (mỗi khối có độ dài 192-bit).

Thuật toán 2.11: Nhân Karatsuba hai đa thức có 3 khối (192-bit) trên GF(2^{571})

Đầu vào: Hai đa thức $a(x)$ và $b(x)$ với $T = x^{192}$; A_i, B_i $(0 \leq i < 3)$ là các khối đa thức con 192-bit;

Đầu ra: $c(x) = a(x) * b(x)$

1. $a(x) = A(T) = A_2T^2 + A_1T + A_0$
2. $b(x) = B(T) = B_2T^2 + B_1T + B_0$
3. $t0 = (A_1 + A_2) * (B_1 + B_2)$
4. $t1 = (A_0 + A_2) * (B_0 + B_2)$
5. $t2 = A_2 * B_2$
6. $t3 = (A_0 + A_1) * (B_0 + B_1)$
7. $t4 = A_1 * B_1$
8. $t5 = A_0 * B_0$
9. $c0 = t5$
10. $c1 = t3 + t4 + t5$
11. $c2 = t1 + t2 + t4 + t5$
12. $c3 = t0 + t2 + t4$
13. $c4 = t2$
14. $c(x) = C(T) = 4T^4 + c_3T^3 + c_2T^2 + c_1T^1 + c_0T^0$

2.4.3. Thực nghiệm và đánh giá thuật toán đề xuất

Luận án sử dụng thư viện RELIC [54] để tích hợp cài đặt thuật toán đề xuất (MKNv8) mô tả ở trong mục trên. Tiếp theo để đánh giá tính hiệu quả của MKNv8, luận án sử dụng thuật toán mật mã là ECDSA, ECDH và ECMQV. Các thử nghiệm được thực hiện trên hai nền tảng là Samsung Galaxy Tab S2 (AArch32) chạy hệ điều hành Android 7 và NXP IMX8M Kit (AArch64) chạy hệ điều hành Linux nhúng (hoặc Android 9).

Mã thực thi được viết bằng ngôn ngữ C và ngôn ngữ assembly sử dụng thư viện RELIC [54]. Mỗi chức năng được đo nhờ sử dụng hai vòng lặp lồng nhau, với n lần lặp của mỗi vòng: trong đó vòng lặp bên ngoài, thì các đầu vào được sinh ngẫu nhiên và vòng lặp bên trong sẽ thực hiện phép tính toán cần thiết với đầu vào lấy từ vòng lặp bên ngoài. Thời gian tổng cộng được thực hiện bởi thủ tục này, được đo bởi hàm clock_gettime theo đơn vị nano giây (nanosecond), sẽ được chia cho n^2 để tạo kết quả

thời gian trung bình của phép toán. Chúng ta chọn n = 128 để đo các phép toán mà có thời gian thực thi nhanh (như phép cộng, nhân... trong số học trường hữu hạn), và n = 32 cho các phép toán có thời gian thực hiện lâu như phép nhân điểm.

Về trình biên dịch và các tham số khi biên dịch:

- Đối với thiết bị Samsung Galaxy Tab S2 (AArch32) chạy hệ điều hành Android 7, luận án sử dụng trình biên dịch arm-linux-androideabi-gcc phiên bản 32bit với tùy chọn được thiết lập khi biên dịch như sau:

  arm-linux-androideabi-gcc -D_GNU_SOURCE -pipe -std=c99 -Wall -O2 -g -march=armv8-a -mfpu=crypto-neon-fp-armv8 -mtune=cortex-a53 -pie

- Đối với thiết bị NXP IMX8M Kit (AArch64) chạy hệ điều hành Linux nhúng, luận án sử dụng trình biên dịch aarch64-linux-gnu-gcc phiên bản 64bit với các tùy chọn được thiết lập khi biên dịch như sau:

  aarch64-linux-gnu-gcc -D_GNU_SOURCE -pipe -std=c99 -Wall -O2 -g -march=armv8-a+crypto+crc -mtune=cortex-a53

Ngoài ra, để đánh giá được tốc độ thực hiện của thuật toán đề xuất, luận án thực hiện so sánh tốc độ thực thi giữa thuật toán đề xuất với thuật toán LOHAD [29] trong thư viện RELIC. Các kết quả lần lượt được thể hiện trong bảng 2.13, bảng 2.14 dưới đây.

Bảng 2.13: Kết quả đánh giá ECDSA trên NXP IMX8M Kit

Algorithm	**LOHAD [29] (nanosec)**	**MKNv8 (nanosec)**
Sign ECDSA (NIST-B283)	2157937	504856
Verify signature ECDSA (NIST-B283)	10463561	2426928
Sign ECDSA (NIST-B409)	4740125	1042230
Verify signature ECDSA (NIST-B409)	22585278	4920991
Sign ECDSA (NIST-B571)	9947431	2051932
Verify signature ECDSA (NIST-B571)	46434713	9791679

Bảng 2.14: Kết quả đánh giá ECDH và ECMQV trên NXP IMX8M Kit

Algorithm	**LOHAD [29] (nanosec)**	**MKNv8 (nanosec)**

ECDH (NIST-B283)	7949461	4631675
ECMQV (NIST-B283)	10392029	2402613
ECDH (NIST-B409)	17122435	10343700
ECMQV (NIST-B409)	22685284	4850183
ECDH (NIST-B571)	37713264	22220776
ECMQV (NIST-B571)	45770904	9633735

2.5. Kết luận chương 2

Phép nhân số học là phép tính quan trọng nhất trong các phép toán số học trên trường hữu hạn. Có nhiều phương pháp để thực hiện cài đặt bằng phần mềm cho thuật toán nhân trên trên trường hữu hạn. Do trong các thiết bị vi xử lý hiện đại thì thời gian thực hiện giữa phép nhân và phép cộng sẽ không còn chênh lệnh nhiều như trên các vi xử lý trước đây (thường trên vi xử lý mới thời gian thực hiện phép nhân gấp 2-3 lần thời gian thực hiện phép cộng), nên việc đề xuất thuất cải tiến hiệu quả phép nhân số học cho những vi xử lý mới là cần thiết.

Dựa trên hai thuật toán nhân hai số hạng cơ bản là thuật toán nhân theo phương pháp phổ thông và thuật toán nhân theo phương pháp Karatsuba, chương này đã đề xuất một phương pháp nhân hai số hạng trên trường hữu hạn, gọi là phương pháp phân tầng. Với phương pháp phân tầng, luận án đã xây dựng được thuật toán có chi phí tốt nhất trong các trường hợp cụ thể (tùy thuộc vào độ dài số hạng) cũng như đưa ra công thức để xác định chi phí của thuật toán đó.

Các kết quả chính của chương 2 được trình bày trong ba công trình công bố [J1, C1, C2] với hai kết quả chính là:

- Đề xuất thuật toán nhân phân tầng với chi phí thấp nhất và phương pháp đánh giá hiệu quả thuật toán nhân này.
- Áp dụng thuật toán nhân phân tầng kết hợp với bộ nhân trên trong thành phần NEON để nâng cao hiệu quả của phép nhân số học trong trường nhị phân trên vi xử lý ARMv7 và ARMv8.

Kết quả thực nghiệm cho thấy thuật toán đề xuất nhanh hơn các kỹ thuật trước đó lần lượt là 4, 5, 5 lần trong trường GF(2^{283}), GF(2^{409}), GF(2^{571}) khi thực hiện trên cùng một nền tảng phần cứng ARMv7 hoặc ARMv8.

Trong chương 3 của luận án sẽ đề xuất cải tiến thuật toán nhân vô hướng (nhân giữa một điểm và một số nguyên dương) của mật mã đường cong Elliptic trên trường nguyên tố, bằng cách nâng cao hiệu quả của thuật toán tìm NAF, cũng như nâng cao hiệu quả của phép cộng điểm và nhân đôi điểm.

CHƯƠNG 3. NÂNG CAO HIỆU QUẢ PHÉP NHÂN VÔ HƯỚNG CỦA HỆ MẬT ECC TRONG TRƯỜNG NGUYÊN TỐ TRÊN VI XỬ LÝ ARM

Các ứng dụng bảo mật thực tế thường hay sử dụng các tham số miền của đường cong ellitpic trên trường nguyên tố, do vậy việc nghiên cứu nâng cao hiệu quả của thuật toán ECDSA và ECDH trên các trường nguyên tố là cần thiết. Trong chương 3, luận án sẽ đề xuất một số giải pháp nhằm cải tiến và nâng cao hiệu quả thực hiện phép nhân điểm trên đường cong ECC trong trường nguyên tố dựa trên kết hợp giữa cải tiến NAF và cải tiến phép cộng và nhân đôi điểm nhờ sử dụng phép nhân kép. Các kết quả chính của chương 3 được trình bày trong hai công trình công bố [J2, J3].

3.1. Cơ sở để nâng cao hiệu quả của phép nhân điểm vô hướng trên ECC trong trường nguyên tố.

3.1.1 Nâng cao hiệu quả của phép tính số học trong trường nguyên tố trên vi xử lý nhúng

Để triển khai trên những nền tảng phần cứng có năng lực xử lý hạn chế hoặc trên các nền tảng phần cứng với các bộ vi xử lý thế hệ mới, thuật toán số học cần được cải tiến nhằm khai thác tối đa các tính năng mới của phần cứng như đã chỉ ra trong một số nghiên cứu như [6, 16, 30, 31, 33]. Trong trường nguyên tố, việc thay đổi thuật toán số học dựa trên khai thác các tính năng mới (ví dụ như ứng dụng NEON trong vi xử lý ARM Cortex-A) sẽ gặp phải khó khăn trong vấn đề xử lý lan truyền phần dư. Để xử lý lan truyền phần dư khi thay đổi thuật toán số học, thường có hai phương pháp:

a) Phương pháp biểu diễn cơ số không dư thừa
b) Phương pháp biểu diễn cơ số dư thừa

Tuy nhiên, biểu diễn cơ số rút gọn (còn gọi với tên khác là biểu diễn dư thừa) sẽ yêu cầu tính toán các tích trung gian nhiều hơn, do đó sẽ phải thực hiện nhiều phép nhân hơn so với biểu diễn chính tắc (biểu diễn không dư thừa). Cho ví dụ, nếu ta sử dụng một biểu diễn với cơ số là 2^{24} (tức là sử dụng 24bit trên một từ 32bit) cho số hạng 192-bit, thì tổng số các tích trung gian cần tính là 8 x 8 = 64. Ngược lại, nếu ta sử

dụng một biểu diễn không dư thừa với cơ số là 2^{32} thì số phép nhân trung gian cần tính chỉ là 6 x 6 = 36

Đã có một số công trình nghiên cứu khai thác các chỉ lệnh NEON trên bộ vi xử lý ARM để cài đặt các thuật toán mật mã nhằm tăng tốc độ, điển hình như [14, 37, 39, 17]. Một số nghiên cứu mới đây như trong [31, 33, 34, 35] đã chỉ ra những hiệu quả nhất định khi ứng dụng kiến trúc SIMD với NEON vào triển khai các phép toán số học trên ECC. Các chỉ lệnh NEON, cụ thể là việc phân rã thuật toán để thực hiện song song trong SIMD đã mang lại hiệu quả tốt. Tuy nhiên, vẫn còn tồn tại một số vấn đề cần giải quyết như còn nhiều tích trung gian làm giảm hiệu năng, khả năng xử lý phần dư trên trường nguyên tố, khả năng giảm bớt các phép tính trung gian dư thừa trong tính toán.

3.1.2 Nâng cao hiệu quả của các phép toán số học điểm

Phép toán cơ bản của ECC là phép nhân điểm vô hướng *kP*, trong đó *k* là một số nguyên, *P* là một điểm trên đường cong elliptic. Để thực hiện phép nhân điểm vô hướng, ta có thể sử dụng các phép cộng điểm (point addition) và nhân đôi điểm (point doubling multiplication). Trong các phép toán số học điểm, thì phép nhân điểm vô hướng có độ phức tạp cao hơn, chiếm nhiều thời gian tính toán nhất. Hiệu quả của hệ mật ECC phụ thuộc chủ yếu vào độ phức tạp và tốc độ của các phép toán này [40]. Việc giải quyết hai mục tiêu có tính đối lập này là một thách thức do hiệu quả còn phụ thuộc vào năng lực tính toán của nền tảng triển khai, đặc biệt là đối với những nền tảng phần cứng với các bộ xử lý có hiệu năng thấp.

Những nghiên cứu lý thuyết về nâng cao hiệu quả các phép toán số học điểm trên đường cong elliptic thường tập trung vào các chủ đề: 1) giảm số phép toán cộng và phép nhân đôi điểm cần trong phép nhân vô hướng; 2) tăng hiệu quả công thức cộng và nhân đôi điểm qua khai thác các phương pháp biểu diễn điểm của đường cong elliptic; 3) giảm thiểu chi phí tính toán bằng cách nâng cao hiệu quả của phép tính số học [33, 35]. Trên thực tế, ta có thể kết hợp các phương pháp này với nhau để đạt hiệu quả tốt hơn trên bất kỳ một nền tảng phần cứng nào. Do vậy, một số nghiên cứu đi theo hướng kết hợp các phương pháp nêu trên, ví dụ như trong [16, 36, 37].

Trong chương 3 của luận án, sẽ kết hợp cả 3 phương pháp này để nâng cao hiệu quả của phép tính số học điểm cụ thể:

a) Trong mục 3.3 sẽ trình bày về phương pháp biểu diễn eNAF để giảm số phép cộng và nhân đôi điểm

b) Trong mục 3.4 sẽ kết hợp cả phương pháp 2 và 3 đó là nâng cao hiệu quả của phép toán nhân số học (phép nhân kép) và đề xuất tích hợp triển khai cho phép cộng điểm và nhân đôi điểm.

3.2. Một số thuật toán nhân điểm vô hướng trên đường cong Elliptic.

3.2.1 Thuật toán nhị phân Right – to – Left.

Thuật toán nhị phân là thuật toán được biết đến đầu tiên dùng để thực hiện phép nhân vô hướng. Giả sử k có biểu diễn nhị phân $(k_{l-1}k_{l-2}\dots k_0)_2$ với $k_i \in \{0,1\}$ thì $k = \sum_{i=0}^{l-1} k_i 2^i$. Khi đó với điểm P nằm trên đường cong Elliptic ta có:

$$\begin{aligned} kP &= \sum_{i=0}^{l-1} k_i 2^i P \\ &= k_0P + k_1 2^1 P + k_2 2^2 P + \dots + k_{l-1} 2^{l-1} P \\ &= k_0P + 2\left(k_1P + 2\left(k_2P + \dots + 2\left(k_{l-2}P + 2(k_{l-1}P)\right)\right)\dots\right) \end{aligned}$$

Thuật toán luỹ thừa nhị phân từ phải qua trái được thực hiện như sau:

Thuật toán 3.1:

INPUT: $k = (k_{l-1}k_{l-2}\dots k_0)_2$, $P \in E(\mathbb{F}_p)$

OUTPUT: $Q = kP$

1. $Q = \infty$
2. For (i = 0) to $l - 1$ do
   - 2.1 If $(k_i = 1)$ $Q = Q + P$
   - 2.2 Else $P = 2P$
3. Ouput Q.

3.2.2 Thuật toán NAF (non-adjacent form)

Theo định nghĩa, nếu điểm P có tọa độ (x, y) trên (E) thì điểm $–P$ sẽ có tọa độ (x, $–y$). Như vậy, phép trừ hai điểm cũng thực hiện giống như phép cộng. Do đó ta có một cách biểu diễn có dấu số nguyên k (signed digit representation - SDR): $k =$

$\sum_{i=0}^{l-1} k_i 2^i$ với $k_i \in \{-1,0,1\}$. Biểu diễn này giúp ta tính toán nhanh hơn phép nhân vô hướng *kP*. Một biểu diễn SDR đặc biệt hữu ích đó là dạng *non-adjacent form* (NAF). Biểu diễn này có tính chất: không có hai hệ số liên tiếp trong biểu diễn của *k* là khác không, có nghĩa là $k_i k_{i+1} = 0, \forall 0 \leq i \leq l-1$. Mọi số nguyên *k* luôn có duy nhất một biểu diễn NAF, chúng ta ký hiệu là NAF(*k*). Hơn nữa, NAF(*k*) có các hệ số khác không ít nhất trong tất cả các cách biểu diễn có dấu của *k*. NAF(*k*) có thể được tính theo thuật toán 3.4 trong mục 3.3.1.

3.2.3 Thuật toán NAF cửa sổ trượt cho tính phép nhân vô hướng trên đường cong Elliptic.

Thuật toán này tác động từ trái qua phải các bit của *k* với cỡ cửa sổ lớn nhất là *w*, trong đó cửa sổ nhận giá trị lẻ. Ngược lại với phương pháp cửa sổ, phương pháp này không đưa ra chính xác cỡ cửa sổ, tuy nhiên phương pháp này cũng có điểm chung với phương pháp cửa sổ đó là loại bỏ đi các bit 0. Thuật toán này kết hợp thuật toán cửa sổ trượt với biểu diễn NAF(*k*) của *k*. Do đó chúng ta cần tính trước các giá trị $P_i = [i]P$, $i < I$ với giá trị *I* nào đó. Ta sẽ tìm *I* như sau: với cỡ cửa sổ *w* bất kì, một dãy các bít của *k* chỉ có thể nhận 2 dạng là 101010…10101 hoặc 101010…101001 (*w* bit), tương ứng với giá trị *w* là lẻ hay chẵn. Vì vậy số các dãy bit có thể nhận được là $\frac{(2^{w+1}-1)}{3}$ hoặc $\frac{(2^{w+1}-5)}{3}$. Do đó số các bước cần tính trước là $I = 2\frac{2^w-(-1)^w}{3} - 1$.

Thuật toán được thực hiện như sau:

Thuật toán 3.2: Thuật toán NAF cửa sổ trượt

INPUT: Cỡ cửa sổ *w*, NAF(*k*) và một điểm $P \in E(\mathbb{F}_p)$.

OUTPUT: kP

1. Tính $P_1 = P$, $P_2 = [2]P$
2. For (i=3) to $2\frac{2^w-(-1)^w}{3} - 1$

   $P_i = P_{i-2} + P_2$
3. $Q = \infty$, $i = length(NAF(k)) - 1$
4. While ($i \geq 0$) do

   4.1 If ($k_i = 0$) then t = 1, u = 0

   4.2 else

4.2.1 t = 1, j = w

4.2.2 While (t = 1 và j > 1) do

If $(k_i \ldots . k_{i-j+1})$ lẻ t = j, $u = (k_i \ldots . k_{i-j+1})$

4.3 $Q = 2^t Q$

4.4 If (u > 0) then $Q = Q + P_u$

else if (u < 0) then $Q = Q - P_{-u}$

4.5 i = i - t

5. Return Q

3.3. Mở rộng cho dạng biểu diễn NAF của số nguyên dương

Trong mục này, luận án đề xuất một thuật toán cho việc tìm dạng biểu diễn không liền kề NAF của một số nguyên k cho trước. Thuật toán này tiết kiệm được ba phép tính số học so với thuật toán nguyên thủy. Tiếp theo, với dạng biểu diễn hầu không liền kề aNAF mới của số k, chúng ta đã rút gọn một phép tính bình phương khi thực hiện phép "bình phương – nhân" đối với phép tính lũy thừa k trên các trường hữu hạn, hoặc tương ứng một phép tính gấp đôi khi thực hiện phép "gấp đôi - cộng" trong phép nhân điểm trên đường cong elliptic.

Biết rằng nếu $k = \sum_{i=0}^{\ell-1} k_i 2^i$ với $k_{\ell-1} \neq 0$ và $k_i \in (0, \pm 1)$ thì để tính $g^k \bmod p$ ta có thể thực hiện theo thuật toán sau.

Thuật toán 3.3. (tính $g^k \bmod p$ theo phương pháp "bình phương-nhân")

Input: Số nguyên dương $k = \sum_{i=0}^{\ell-1} k_i 2^i$ với $k_i \in (0, \pm 1)$, (3.1)

$g \in GF^*(p), g' = g^{-1} \bmod p$.

Output: $g^k \bmod p$.

1. $a \leftarrow 1$.
2. For i from $\ell - 1$ downto 0 do

   2.1 $a \leftarrow a^2 \bmod p$.

   2.2 if $k_i = 1$ then $a \leftarrow a * g \bmod p$.

   2.3 if $k_i = -1$ then $a \leftarrow a * g' \bmod p$.
3. return a.

Theo thuật toán trên thì chi phí tính toán sẽ bằng đúng $\ell - 1$ phép bình phương và w – 1 phép nhân, trong đó w là số các ký tự $k_i \neq 0$ trong công thức 1. Như vậy bài toán tìm biểu diễn của k theo công thức (1) sao cho ℓ + w bé nhất được đặt ra một cách tự nhiên nhằm giảm chi phí tính toán cho thuật toán 3.3.

Trong mục này sau khi giới thiệu lại khái niệm NAF và các kết quả đã có về nó trong mục 3.3.1 thì tại mục 3.3.2 luận án đưa ra một thuật toán mới để tính NAF(k). Thuật toán mới đưa ra được đánh giá qua mệnh đề 4 là hiệu quả hơn thuật toán đã có. Cuối cùng, tại mục 3.3.3, đưa ra một khái niệm mới cho dạng biểu diễn, ký hiệu là aNAF, với các ý nghĩa quan trọng đó là:

- Tổng w + ℓ của aNAF(k) luôn không vượt quá của NAF(k).
- Số các giá trị k trong $[2^{\ell-1}, 2^{\ell})$ với $\ell > 1$ là không dưới $\frac{2^{\ell-1}}{8}$.

3.3.1 Dạng không liền kề (NAF)

Tại mục này luận án sẽ trình bày lại khái niệm NAF và các kết quả đã có về NAF theo (xem trang 98 trong [7])

A. Khái niệm NAF và các tính chất của NAF(k)

Định nghĩa 1. (xem [7]) Dạng NAF (non-adjacent form) của một số nguyên dương k là biểu thức $k = \sum_{i=0}^{\ell-1} k_i 2^i$ ở đây $k_{\ell-1} \neq 0, k_i \in (0, \pm 1)$ và không có hai ký tự k_i liền nhau nào đều khác 0. Khi này ℓ được gọi là độ dài của NAF còn $k_{\ell-1}k_{l-2} \dots k_0$ được gọi là biểu diễn NAF của k, ký hiệu là NAF(k).

Định lý 1. (xem [7]) (tính chất của các NAF)

(i) Với mọi k có duy nhất NAF(k).

(ii) NAF(k) có số ký tự khác 0 là nhỏ nhất trong mọi biểu diễn nhị phân có dấu của k.

(iii) Nếu $2^{\ell-1} \leq k < 2^{\ell}$ thì độ dài NAF(k) bằng $\ell + \delta$ với $\delta \in \{0,1\}$.

(iv) Nếu độ dài NAF(k) = ℓ thì $\frac{2^{\ell}}{3} < k < \frac{2^{\ell+1}}{3}$.

(v) Số các ký tự khác 0 trung bình trong tất cả các NAF độ dài ℓ là xấp xỉ $\frac{\ell}{3}$.

B. Thuật toán tính NAF(k)

Cũng trong [7], thuật toán tìm NAF(k) được trình bày như sau.

Thuật toán 3.4. (xem [7])

Input: Số nguyên dương k.

Output: NAF(k).

1. $\ell \leftarrow 0$.
2. While $k \geq 1$ do

   2.1 if k is odd then: $k_\ell \leftarrow 2 - (k \bmod 4)$, $k \leftarrow k - k_\ell$;

   2.2 else: $k_\ell \leftarrow 0$.

   2.3 $k \leftarrow \frac{k}{2}, \ell \leftarrow \ell + 1$.
3. return $k_{\ell-1}k_{\ell-2} \dots k_0$.

3.3.2 Thuật toán mới tìm NAF(k)

Trong mục này luận án đưa ra thêm một thuật toán mới (thuật toán 3.5) để tính NAF(k).

A. Thuật toán tìm NAF(k) mới

Thuật toán 3.5:

Input: Số nguyên dương k.

Output: NAF(k).

1. $k_{\ell-1}k_{\ell-2} \dots k_0 \leftarrow \text{Binary}(k)$.
2. $j \leftarrow 0$.
3. While $j < l$ do:

   3.1 While $k_j = 0$ do $j \leftarrow j + 1$.

   3.2 if $(j = \ell - 1)$ or $(k_{j+1} = 0)$ then $j \leftarrow j + 1$;

   3.3 else:

      3.3.1 $k_j \leftarrow -1$.

      3.3.2 $j \leftarrow j + 1$.

      3.3.3 while $(k_j = 1)$ do: $k_j \leftarrow 0, j \leftarrow j + 1$.

      3.3.4 $k_{j+1} \leftarrow 1$.

      3.3.5 If $(j = \ell)$ $\ell \leftarrow \ell + 1$.
4. return $k_{\ell-1}k_{\ell-2} \dots k_0$.

Ở trên Binary(k) là biểu diễn nhị phân không dấu của k.

B. Tính đúng đắn của thuật toán 3.5

Tính đúng đắn của thuật toán 3.5 được cho bởi kết quả sau.

Mệnh đề 2. *Thuật toán 3.5 là đúng đắn tức là dãy $k_{\ell-1}k_{\ell-2}\dots k_0$ thu được tại bước 4 chính là NAF(k).*

Chứng minh. Trước hết ta có dãy $k_{\ell-1}k_{\ell-2}\dots k_0$ thu được sau bước 1 thỏa mãn $k_j \in \{0,1\}$ với mọi j = 0, …, ℓ – 1 và sự thay đổi các giá trị này chỉ xảy ra tại 3.3.1, 3.3.3 và 3.3.4 mà các giá trị được thay đổi tương ứng trong các bước này là – 1, 0 và 1 nên dãy $k_{\ell-1}k_{\ell-2}\dots k_0$ thu được sau bước 3 thỏa mãn $k_j \in \{0, \pm 1\}$. Như vậy để chứng minh mệnh đề này ta chỉ cần chỉ ra rằng không tồn tại hai ký tự khác 0 liền nhau của dãy đầu ra.

Biết rằng trong dãy đầu ra thì ký tự $k_j \neq 0$ chỉ xuất hiện ngay sau bước 3.1. Ta có:

Nếu điều kiện của bước 3.2 được thỏa mãn thì:

- Hoặc là ta đã xét đến j = ℓ – 1 và khi này dãy $k_{\ell-1}k_{\ell-2}\dots k_0 = 10k_{\ell-3}\dots k_0$.
- Hoặc là đoạn $\dots k_{j+1}k_jk_{j-1}\dots k_0 = \cdots 010\dots k_0$ và do k_{j+1} sẽ vẫn bằng 0 theo bước 3.1 của vòng sau nên ký tự $k_j = 1$ này không có ký tự bên cạnh nào cũng khác 0.

Ngược lại, theo các bước của 3.3 thì nếu có t > 1 ký tự liên tiếp (tính từ k_j) bằng 1 tức là $k_{j+t-1}k_{j+t-2}\dots k_j = 11\dots1$ thì bước này sẽ biến đổi đoạn dãy trên thành $k_{j+t-1}k_{j+t-2}\dots k_j = 1\underbrace{0\dots0}_{t-1} - 1$ như vậy trước ký tự $k_j = -1$ có ít nhất t – 1 ký tự bằng 0. Lại do nếu j = 0 thì không tồn tại ký tự sau k_j còn ngược lại thì theo 3.1 thì ký tự ngay trước k_j phải bằng 0. Tóm lại ta cũng có k_j không liền kề với một ký tự khác 0 nào.

Tóm lại mệnh đề đã được chứng minh. ■

Ngoài ra chúng ta dễ dàng kiểm tra được rằng thuật toán 3.5 có tính chất sau.

Tính chất 3.1. *Với mọi j, sau khi xác định được ký tự thứ j của NAF(k) thì tất cả các ký tự k_i với i > j vẫn được giữ nguyên chỉ trừ ra j là vị trí cuối cùng của vòng lặp 3.3 (khi này k_{j+1} phải là 0 và được đổi thành 1).*

C. So sánh tính hiệu quả của hai thuật toán 3.4 và 3.5

Việc so sánh tính hiệu quả giữa hai thuật toán 3.4 và 3.5 được cho trong kết quả sau.

Mệnh đề 4. *Thuật toán 3.5 là hiệu quả hơn thuật toán 3.4.*

Chứng minh. Hai thuật toán đều phải thực hiện xác định ℓ giá trị k_j (j = 0, …, ℓ – 1).

Đối với thuật toán 3.4 cần:

- Một phép tính k mod 4, một phép trừ 2 cho k mod 4 ($k_j = 2 - (k \bmod 4)$), một phép trừ k cho k_j trong trường hợp k lẻ.
- Một phép chia k cho 2 cho mọi trường hợp.

Đối với thuật toán 3.5 cần:

- Một phép chia k cho 2 cho mọi trường hợp (để tìm khai triển nhị phân của k).
- Một phép đặt $k_j = -1$ khi j bắt đầu vào vòng 3.3, một phép đảo bít (0 thành 1 hoặc ngược lại) khi j ở bước tiếp theo cho đến khi ra ngoài vòng 3.3.

Như vậy nếu $k_j = 0$ thì cả hai thuật toán đều có chi phí tính toán như nhau. Ngược lại thì thuật toán 3.4 cần thực hiện thêm ba phép tính số học so với thuật toán 3.5. Tóm lại mệnh đề đã được chứng minh. ■

D. Ví dụ

Tính NAF(348).

*Các bước thực hiện theo thuật toán 3.4.

Bảng 3.1: Các bước thực hiện theo thuật toán 3.4

j	k_j	$k = (k - k_j)/2$	j	k_j	$k = (k - k_j)/2$
0	0	174	5	-1	6
1	0	87	6	0	3
2	-1	44	7	-1	2
3	0	22	8	0	1
4	0	11	9	1	0

*Các bước thực hiện theo thuật toán 3.5.

Bước 1. Binary(348) = 101011100, ℓ = 9.

Bước 3.

Bảng 3.2: Các bước thực hiện theo thuật toán 3.5

	Các bước thực hiện	$k_{\ell-1}k_{\ell-2} \ldots k_0$	j	ℓ

Vòng 1	3.1 3.3	1 0 1 0 1 1 1 0 0 1 0 1 1 0 0–1 0 0	2 5	9
Vòng 2	3.3	1 1 0–1 0 0–1 0 0	7	9
Vòng 3	3.3	1 0–1 0–1 0 0–1 0 0	8	10
Vòng 4	3.2	1 0–1 0–1 0 0–1 0 0	10	10

3.3.3. Dạng biểu diễn hầu không liền kề

Trong phần này, luận án đưa ra một dạng biểu diễn mới cho một số nguyên dương k, được gọi hầu không liên kề, được kí hiệu là aNAF(k). Chú ý, trong dạng biểu diễn này, sẽ giữ nguyên dạng biểu diễn của NAF của k chỉ có một sự điều chỉnh khi dạng biểu diễn NAF(k) có mẫu $k_{l-1}k_{l-2}k_{l-3}$ có dạng mẫu $10-1$ khi đó được thay thế bởi 11. Cụ thể, được định nghĩa như sau:

Định nghĩa 2. *Cho k là một số nguyên dương. Khi đó ta gọi khai triển NAF mở rộng của k, ký hiệu là $aNAF(k)$, được xác định theo công thức sau*

$$aNAF(k) = \begin{bmatrix} 11k_{\ell-4}\dots k_0 & khi\ NAF(k) = 10-1k_{\ell-4}\dots k_0 \\ NAF(k) & trong\ trường\ hợp\ còn\ lại \end{bmatrix}. \quad (3.2)$$

Như vậy, dạng aNAF(k) sẽ không là NAF(k) khi $k_{l-1}k_{l-2}k_{l-3}$ có dạng mẫu $10-1$. Tuy nhiên, ta vẫn thấy có quan hệ 1-1 trong dạng biểu diễn aNAF và NAF. Hơn nữa, ta có tính chất như sau.

Định lý 2.

(i) *Với mọi số nguyên dương k thì số ký tự khác 0 của aNAF(k) và NAF(k) là bằng nhau.*

(ii) *Với $2^{\ell-1} \le k < 2^{\ell}$ ($\ell > 1$) thì số các aNAF(k) có độ dài nhỏ hơn độ dài của NAF(k) là không dưới $\frac{2^{\ell-1}}{8}$.*

Chứng minh. Từ định nghĩa 2 ta thấy kết quả (i) là hiển nhiên.

Với $\ell = 2$. Ta có 2 giá trị k thỏa mãn $2^{\ell-1} \le k < 2^{\ell}$ đó là 2 và 3.

NAF(2) = 1 0, NAF(3) = **1 0 –1** . Dãy sau có độ dài lớn hơn dãy aNAF, trong khi $\frac{2^{\ell-1}}{4} = \frac{2}{4} < 1$ vậy (ii) đã thỏa mãn.

Với $\ell = 3$. Ta có 4 giá trị k thỏa mãn $2^{\ell-1} \le k < 2^{\ell}$ đó là 4, 5, 6 và 7.

k	4	5	6	7
NAF(k)	1 0 0	1 0 1	**1 0 –1 0**	1 0 0 – 1

Số các NAF(k) có độ dài lớn hơn dãy aNAF(k) là 1, trong khi $\frac{2^{\ell-1}}{4} = \frac{2^2}{4} = 1$ vậy (ii) đã thỏa mãn.

Với ℓ = 4. Ta có 4 giá trị k thỏa mãn $2^{\ell-1} \leq k < 2^{\ell}$ đó là 8, 9, 10, 11, 12, 13, 14 và 15.

k	8	9	10	11
NAF(k)	1 0 0 0	1 0 0 1	1 0 1 0	**1 0–1 0–1**

k	12	13	14	15
NAF(k)	**1 0–1 0 0**	**1 0–1 0 1**	1 0 0– 1 0	1 0 0 0 – 1

Số các NAF(k) có độ dài lớn hơn dãy aNAF(k) là 3, trong khi $\frac{2^{\ell-1}}{4} = \frac{2^3}{4} < 3$ vậy (ii) đã thỏa mãn.

Với ℓ > 4.

Xét tập hai tập sau

$R(\ell) = \{k: 2^{\ell-1} \leq k < 2^{\ell}$ và Binary(k) $= 1011k_{\ell-5} \dots k_0\}$. (3.3)

$S(\ell) = \{k: 2^{\ell-1} \leq k < 2^{\ell}$ và Binary(k) $= 1100k_{\ell-5} \dots k_0\}$. (3.4)

Ta có hai tập trên đều có đúng $\frac{2^{\ell-1}}{8}$ phần tử cho nên nếu ta chỉ ra được mọi phần k tử thuộc các tập trên đều thỏa mãn NAF(k) có độ dài lớn hơn dãy aNAF(k) thì (ii) đã được chứng minh.

Nếu k ∈ R, tức là Binary(k) $= 1011k_{\ell-5} \dots k_0$. Theo tính chất 3.1 thì khi sau khi xác định được $k_{\ell-5}$ thì do ký tự $k_{\ell-4}$ (vốn bằng 1) nên bước $\ell - 5$ chỉ xảy ra một trong hai khả năng sau:

- Là bước trung gian của vòng 3.3 , điều này chỉ xảy ra với $k_{\ell-5} = 1$. Khi này bước cuối cùng của 3.3 sẽ ứng với j = ℓ – 2 và sau vòng này dãy trở thành $1100k_{\ell-5} \dots k_0$. Tiếp đây sẽ là bước tiếp theo của 3 với vòng lặp 3.3 để thu được dãy đầu ra là $10 - 100k_{\ell-5} \dots k_0$.
- Là bước cuối của 3.1, điều này chỉ xảy ra với $k_{\ell-5} = 0$. Khi này dãy chính là $10110k_{\ell-6} \dots k_0$ và bước tiếp theo của 3 chính là 3.3 chúng sẽ chuyển dãy trên trở thành $10 - 101k_{\ell-6} \dots k_0$.

Tóm lại cho dù k thuộc R hay S thì độ dài NAF(k) đều lớn hơn độ dài aNAF(k).

Tương tự, nếu k ∈ S, tức là Binary(k) $= 1100k_{\ell-5} \ldots k_0$. Theo tính chất 3.1 thì khi sau khi xác định được $k_{\ell-5}$ thì ký tự $k_{\ell-4}$ (vốn bằng 0) nên trong mọi trường hợp nó sẽ nhận giá trị trong {0, 1} cho nên từ $k_{\ell-3} = 0$ giá trị $k_{\ell-4}$ và $k_{\ell-3}$ sẽ không thay đổi từ đó vòng lặp tiếp theo của bước 3 sẽ bắt đầu từ 3.1 cho đến khi j = $\ell - 2$. Tiếp đến sẽ là bước lặp 3.3 (điều kiện của 3.2 không được thỏa mãn) sẽ biến $k_{\ell-1}k_{\ell-1} = 1\ 1$ thành 1 0 –1 tức là độ dài NAF(k) dài hơn độ dài aNAF(k).

Qua trên ta có số các giá trị k có độ dài NAF(k) lớn hơn độ dài aNAF(k) là không dưới #R + #S = $2\frac{2^{\ell-1}}{8} = \frac{2^{\ell-1}}{4}$ vậy định lý đã được chứng minh.

3.4. Nâng cao hiệu quả của phép toán số học điểm trên hệ mật ECC trong trường nguyên tố

Trong phần này, sẽ trình bày một phương pháp nâng cao hiệu quả các phép toán số học (bao gồm phép cộng, phép nhân đôi, phép nhân điểm vô hướng) trên đường cong elliptic. Phương pháp này tập trung khai thác tối đa đặc điểm phần cứng SIMD trên nền vi xử lý ARM có tích hợp thành phần NEON nhằm tăng tốc độ các phép toán số học. Các đặc trưng chính của phương pháp gồm: 1) kết hợp phép nhân thông thường (phương pháp quét số hạng) với thuật toán nhân Karatsuba cho các số hạng dài; 2) thực hiện nhân song song (nhân kép) trên trường số nguyên tố kết hợp với ghép cặp để giảm thiểu chi phí đọc, ghi dữ liệu giữa bộ nhớ và thành phần NEON; 3) sử dụng các thư viện số tính toán lớn có sẵn RELIC [54] nhằm tăng tốc độ tính toán. Phương pháp đã đề xuất đã được tích hợp đầy đủ vào tính toán của các giao thức ECDH (Elliptic Curve Diffie Hellman) và ECDSA (Elliptic Curve Digital Signature Algorithm) trên trường GF(p) có kích thước là 256, 384, và 521-bit.

Mô hình thực hiện như sau:

Nguyên thủy mật mã	Ký số/Kiểm tra chữ ký số	Mã hóa/giải mã	Trao đổi khóa
Phép toán trên đường cong Elliptic	Nhân vô hướng (Scalar Multiplication)		
	Cộng điểm (Point Add)	Nhân đôi điểm (Point Double)	
Phép toán số học trong GF(p)	Phép toán thông thường (cộng trừ, nhân…)	Phép nhân kép (Song song hai phép nhân)	
ARM/NEON	VMULL.U32, VLD, VST, VADD...		

Hình 3.1: Thực hiện tại nhân kép các tầng trong ECC

Trước hết, ở tầng các phép toán số học GF(p), luận án sẽ đề xuất xây dựng ba phép nhân kép mà số hạng đầu vào có độ dài lần lượt là 256-bit, 384-bit và 521-bit. Sau đó, luận án tích hợp phép nhân kép vào phép toán cộng điểm và nhân đôi điểm nhờ tổ chức lại các bước của thuật toán để ghép cặp phép nhân, phép bình phương. Sau đó luận án sử dụng thuật toán nhân vô hướng dựa trên NAF để thực hiện phép nhân vô hướng.

3.4.1 Các chỉ lệnh sử dụng trên vi xử lý ARM

Trong mục này, các chỉ lệnh sau để thực hiện nhân song song hai phép nhân:

- vmlal_u32 (nhân và tích lũy số nguyên không dấu ở dạng vector):
  Q0 = vmlal_u32(Q0, D2, D3[0]) tính
  D1 = D1 + D2[1] x D3[0] và D0 = D0 + D2[0] x D3[0]
- vshrq_n_u64 (dịch phải vector đi n bit)
  Q0 = vshrq_n_u64(Q0, 32) tính
  D0 = D0>>32 và D1 = D1 >>32
- vaddq_u64 (thực hiện phép cộng dạng vector):
  Q0 = vaddq_u64(Q0, Q1) tính
  D0 = D0 + D2 và D1 = D1 + D3
- vandq_u64 (thực hiện phép AND dạng vector):
  Q0 = vandq_u64(Q0, Q1) tính
  D0 = D0 & D2 và D1 = D1 & D3
- vmovn_u64 (phép chuyển dữ liệu trong vector ở chế độ Narrow)
  D0 = vmovn_u64(Q1) tính
  D0[0] = D2[0] và D0[1] = D3[0]

- vld1_u32 (tải dữ liệu từ bộ nhớ vào thanh ghi trong NEON)
  D0 = vld1_u32([N]) tính
  D0[0] = [N] và D0[1] = [N+1]
- vget_lane_u32 (sao chép dữ liệu từ thanh ghi NEON đến bộ nhớ)
  R1 = vget_lane_u32(D0, 0) tính R1 = D0[0]
  R2 = vget_lane_u32(D0, 1) tính R2 = D0[1]

Dựa trên các chỉ lệnh trên, luận án cài đặt thuật toán nhân kép, thuật toán cộng điểm và nhân đôi điểm như đã đề xuất trong mục dưới đây dựa trên thư viện mật mã RELIC phiên bản 0.5.0 [54]. RELIC là một thư viện mật mã được đánh giá là có tốc độ thực thi khá nhanh trong số những thư viện mật mã có mã nguồn mở khác như OpenSSL, GMP. Tiếp theo, các thử nghiệm và đánh giá được thực hiện giữa thư viện mà đã tích hợp các cải tiến đề xuất và thư viện nguyên thủy sử dụng thuật toán mặc định. Để đo thời gian của một phép toán, luận án lấy thời gian trung bình của 100 lần thực thi phép toán đó.

3.4.2. Đề xuất thuật toán song song hai phép nhân trên trường GF(p)

Thành phần NEON có 16 thanh ghi 128-bit, do vậy chúng ta có thể cài đặt trực tiếp theo phương pháp nhân quét số hạng (còn gọi là nhân phổ thông) có kích thước là 256-bit và 384-bit. Đối với phép nhân có kích thước 521-bit, do không thể đủ số thanh ghi để cài đặt phép nhân này trực tiếp trên NEON, do vậy luận án đề xuất sử dụng thuật toán Karatsuba thông thường cho phép toán này dựa trên kết hợp giữa cài đặt ở mức C và cài đặt trong NEON.

Chú ý: Thuật toán tính modulo sử dụng thuật toán nguyên thủy trong [47]

Đối với kiến trúc SIMD, do đặc điểm hỗ trợ tính hai phép nhân 32-bit sử dụng một chỉ lệnh đơn, luận án ứng dụng phép toán này trong thuật toán 3.6.

Thuật toán 3.6: Nhân đồng thời hai phép nhân trên trường GF(p)

Đầu vào: $A = (A_{s-1}, \dots A_1, A_0),\ B = (B_{s-1}, \dots, B_1, B_0)$ và $C = (C_{s-1}, \dots, C_1, C_0), D = (D_{s-1}, \dots, D_2, D_1, D_0)$

Đầu ra: $M = (M_{2s-1}, \dots, M_1, M_0) = A \cdot B$ và $N = (N_{2s-1}, .., N_1, N_0)$

1. $M = 0$, $N = 0$

2. for i = 0 to s - 1 do

3. $T_1 = 0, T_2 = 0$

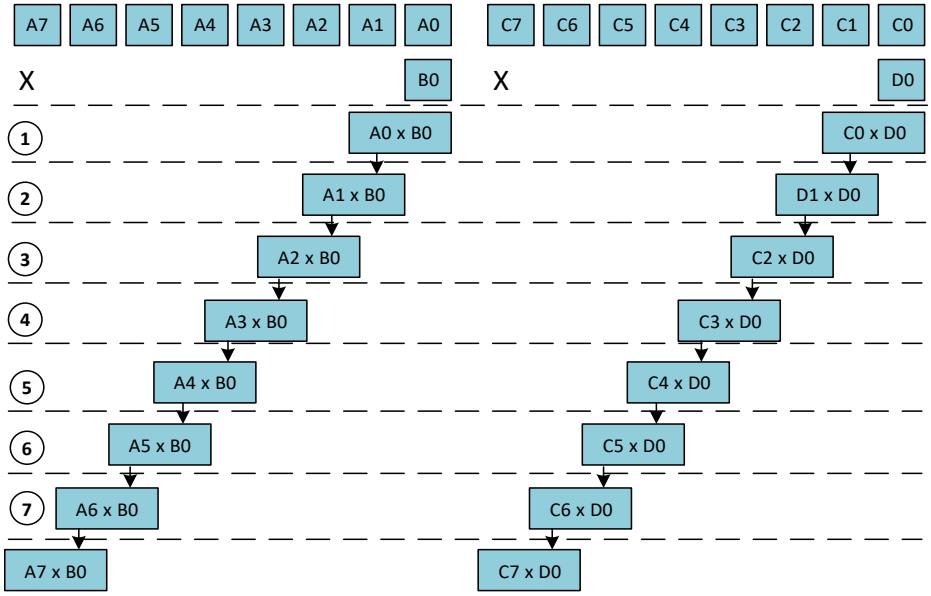
4. for j = 0 to s - 1 do

5. $(T_1, S_1) = M_{i+j} + A_i.B_j + T_1, (T_2, S_2) = M_{i+j} + C_i.D_j + T_2$

6. $M_{i+j} = S_1, N_{i+j} = S_2$

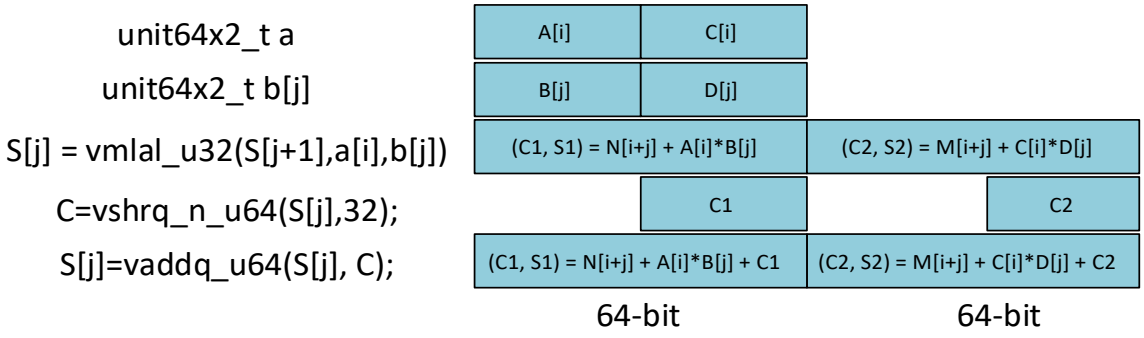
7. $M_{i+s} = T_1, N_{i+s} = T_2$

8. return (M, N)



Hình 3.2: Thực hiện nhân song song hai phép nhân

Các chỉ lệnh NEON để thực hiện bên trong vòng lặp của bước thứ 4 trong thuật toán 3.6 như sau:



Hình 3.3: Tính toán trong vòng lặp j của thuật toán 3.6 sử dụng NEON

Theo chứng minh ở trong tài liệu mục 14.15 [25] thì giá trị (C_1, S_1) và (C_2, S_2) được biểu diễn trong 2 từ (đối với trường hợp này cơ số là 2^{32}). Do khả năng thực hiện

đồng thời hai phép nhân, chỉ lệnh NEON rất hữu ích để tăng tốc cho tính toán các giải thuật số học. Tuy nhiên, việc tải (loading) và lưu (storing) dữ liệu giữa thanh ghi NEON và bộ nhớ có chi phí cao, bởi vậy nhằm mục đích có được lợi thế thực sự từ tính toán song song trong NEON thì các chỉ lệnh tải và lưu dữ liệu (load/store) cần hạn chế tối đa. Điểm khác biệt trong đề xuất ở đây so với mô tả trong bài báo [33] (trong đó các chỉ lệnh này được sử dụng quá nhiều) là hạn chế tối đa sử dụng chỉ lệnh này, cụ thể trong bước 7 đến bước 11 của thuật toán 3 [33] thì các biến dữ liệu từ bộ nhớ &t[i+j], &u[i+j] liên tục được tải vào thanh ghi NEON và lấy ra bộ nhớ trong hai vòng lặp, do vậy sẽ làm chậm hiệu suất thực thi phép nhân song song trong NEON (sẽ không đạt được cơ chế Pipeline).

Đối với trường hợp nhân 256-bit và 384-bit:

Do việc đọc vào và ghi ra dữ liệu giữa bộ nhớ và thanh ghi NEON mất khá nhiều thời gian (sử dụng chỉ lệnh vld và vst), nên luận án thực hiện đọc toàn bộ dữ đầu vào từ bộ nhớ, rồi áp dụng **thuật toán 3.6** để tính toán. Sau đó kết quả từ thanh ghi NEON được ghi trở lại bộ nhớ. Điều này sẽ giảm thiểu thời gian đọc vào ghi dữ liệu giữa bộ nhớ và thanh thi NEON. Cụ thể như sau:

+ Với trường hợp 256-bit: Các số hạng đầu vào 256-bit được lưu trong mảng bộ nhớ 8 từ, mỗi từ 32-bit. Sẽ cần 11 thanh ghi 128-bit, trong đó 9 thanh ghi để lưu các tích trung gian và 2 thanh thi để lưu các biến tạm (carray, biến để lấy dữ liệu phần thấp). Ngoài ra, sẽ cần 10 thanh ghi 64-bit, trong đó 9 thanh ghi để lưu các số hạng đầu vào và 1 thanh ghi để lưu biến tạm. Chi tiết cài đặt được trình bày ở trong Phụ Lục 1.

+ Với trường hợp 384-bit: Các số hạng đầu vào 384-bit được lưu trong mảng bộ nhớ 12 từ, mỗi từ 32-bit. Sẽ cần 15 thanh ghi 128-bit, trong đó 13 thanh ghi để lưu các tích trung gian và 2 thanh thi để lưu các biến tạm (carry, biến để lấy dữ liệu phần thấp). Ngoài ra, sẽ cần 14 thanh ghi 64-bit, trong đó 13 thanh ghi để lưu các số hạng đầu vào và 1 thanh ghi để lưu biến tạm. Chi tiết cài đặt được trình bày ở trong Phụ Lục 1.

Đánh giá số chỉ lệnh tải và lưu dữ liệu

Số chỉ lệnh để nạp và lưu dữ liệu cụ thể trong bước 7 đến bước 11 của thuật toán 3 [33] là gấp đôi phương pháp nhân quét số hạng (vì thực hiện phép nhân kép), do vậy cần tổng chỉ lệnh truy nhập đến bộ nhớ là: $2 \times (3s^2 + 2s)$ (xem mục 2.1.2 – chương 1). Đối với thuật toán 4, ở vòng lặp *i* và *j* chỉ cần nạp số hạng *A*, *B*, *C*, *D*; sau đó cần ghi kết quả cuối cùng (*M*, *N*) đến từ thanh ghi NEON đến bộ nhớ, do vậy cần 4*s* chỉ lệnh tải dữ liệu và $2 \times 2s$ chỉ lệnh lưu dữ liệu. Bảng dưới đây so sánh chỉ lệnh tải dữ liệu và lưu dữ liệu giữa thuật toán mà luận án cải tiến và thuật toán 3 [33].

Bảng 3.3: So sánh chỉ lệnh tải dữ liệu và lưu dữ liệu giữa thuật toán cải tiến và thuật toán 3 [33].

Phương pháp	Chỉ lệnh tải dữ liệu	Chỉ lệnh lưu dữ liệu	Tổng số chỉ lệnh truy cập bộ nhớ
Thuật toán 3 [33]	$2(2s^2 + s)$	$2(s^2 + s)$	$6s^2 + 4s$
Thuật toán 3.6 (nhân 256-bit và 384-bit)	$4s$	$4s$	$8s$

Đối với trường hợp nhân 521-bit:

Do thành phần NEON chỉ có 16 thanh ghi 128-bit, nên không đủ để cài đặt phép nhân song song có độ dài số hạng đầu vào là 521-bit (được lưu trong mảng bộ nhớ 17 từ, mỗi từ 32-bit) mà cần cỡ 24 thanh ghi 128-bit. Bởi vậy, luận án áp dụng thuật toán Karatsuba thông thường kết hợp giữa ngôn ngữ lập trình C/NEON để thực hiện phép nhân 521-bit như sau:

- Đầu tiên, ta cần thực hiện thêm hai bộ nhân là bộ nhân kép cho số hạng nhân 288-bit (9 từ) và bộ nhân kép cho hai số hạng 320-bit (10 từ). Cách thực hiện tương tự như đối với phép nhân kép có số hạng 256-bit.
- Tiếp theo, ta áp dụng thuật toán Karatsuba phổ thông với 1 tầng (level). Phương pháp Karatsuba 1 tầng dựa trên công bố [17]. Theo đó, chúng ta tách số hạng 521-bit (được lưu trong 17 từ 32 bit) thành 2 nửa: Nửa đầu có 8 từ và nửa sau có 9 từ. Có hai cách tiếp cận thường dùng để cài đặt Karatsuba được gọi là Karatsub cộng (additive) và Karatsub trừ (subtractive). Giả sử, ta nhân kép 2 cặp số hạng là $M = A \cdot B$ và $N = C \cdot D$, trong đó $A = A_H \cdot 2^{256} + A_L$,

$B = B_H \cdot 2^{256} + B_L$ và $C = C_H \cdot 2^{256} + C_L$, $D = D_H \cdot 2^{256} + D_L$. Phép nhân kép $M = A \cdot B$ và $N = C \cdot D$ được tính theo công thức cộng sau:

$$A_H \cdot B_H \cdot 2^{521} + [(A_H + A_L)(B_H + B_L) - A_H \cdot B_H - A_L \cdot B_L] \cdot 2^{256} + A_L \cdot B_L$$

$$C_H \cdot D_H \cdot 2^{521} + [(C_H + D_L)(C_H + D_L) - C_H \cdot D_H - C_L \cdot D_L] \cdot 2^{256} + C_L \cdot D_L$$

Các số hạng A_L, B_L, C_L, D_L có độ dài 256-bit (8 từ), do vậy các cặp phép nhân $A_L \cdot B_L$ và $C_L \cdot D_L$ được thực hiện bằng thuật toán nhân kép 8 từ kế thừa từ bộ nhân kép với độ dài số hạng 256-bit. Các số hạng A_H, B_H, C_H, D_H có độ dài 288-bit (9 từ), do vậy các cặp phép nhân $A_H \cdot B_H$ và $C_H \cdot D_H$ được thực hiện bằng thuộc toán nhân kép 9 từ bằng bộ nhân kép với độ dài số hạng 288-bit. Các số hạng $(A_H + A_L), (B_H + B_L), (C_H + C_L), (D_H + D_L)$ có độ dài 320-bit (10 từ), do vậy các cặp phép nhân $(A_H + A_L) \cdot B_H(B_H + B_L)$ và $(C_H + C_L) \cdot (D_H + D_L)$ được thực hiện bằng thuộc toán nhân kép 10 từ bằng bộ nhân kép với độ dài số hạng 320-bit.

Thuật toán 3.7: Nhân karatsuba kép cho 521-bit

Đầu vào: Bốn số hạng có độ dài 17 từ 32bit $A = A_H||A_L$, $B = B_H||B_L$ và $C = C_H||C_L$, $D = D_H||D_L$

Đầu ra: $M = A \cdot B$ và $N = C \cdot D$ có độ dài 34 từ 32-bit

1. Tính song song $M_L = A_L \cdot B_L$ và $N_L = C_L \cdot D_L$	{NEON, 256bit}
2. Tính song song $M_H = A_H \cdot B_H$ và $N_H = C_H \cdot D_H$	{NEON, 288bit }
3. Tính $A_{HL} = (A_H + A_L), B_{HL} = (B_H + B_L)$	{C, 320bit}
4. Tính $C_{HL} = (C_H + C_L), D_{HL} = (D_H + D_L)$	{C, 320bit}
5. Tính song song $M_M = A_{HL} \cdot B_{HL}$ và $N_M = C_{HL} \cdot D_{HL}$	{NEON, 320bit}
6. Tính $M = M_H.2^{521} + (M_M - M_H - M_L).2^{256} + M_L$	{C}
7. Tính $N = N_H.2^{521} + (N_M - N_H - N_L).2^{256} + N_L$	{C}

3.4.3. Cải tiến thuật toán số học trên đường cong Elliptic

3.4.3.1 Cộng điểm

Như đã trình bày ở các mục trên, bất cứ ở đâu xuất hiện cặp phép nhân trên F_p mà dữ liệu của chúng không phụ thuộc vào nhau thì ta có thể sử dụng các thuật toán nhân kép đã trình bày ở trên để thực hiện nhân song song cặp phép nhân này. Nguyên lý này cũng có thể áp dụng cho phép bình phương. Để đơn giản cho quá trình cài đặt,

luận án không thực hiện cài đặt phép bình phương kép (song song hai phép bình phương) mà sử dụng phép nhân kép để thực hiện song song hai phép bình phương. Do trong phép cộng điểm và nhân đôi điểm trên hệ tọa độ Jacobi, các phép nhân và phép bình phương (thậm trí kết hợp giữa nhân và bình phương) có thể phân tách, tổ chức thành các phép toán độc lập nhau, nên ta có thể áp dụng thuật toán nhân kép vào trong thuật toán nhân đôi điểm và cộng điểm. Dựa trên thuật toán cộng điểm (thuật toán 6) được trình bày trong *Explicit-Formulas Database* [52], luận án tổ chức lại và đề xuất thuật toán cộng điểm sử dụng phép nhân kép theo thuật toán 3.7, với đặc điểm là các phép nhân, bình phương thông thường được tổ chức thành các cặp phép nhân, cặp phép bình phương hoặc cặp phép nhân và bình phương mà có dữ liệu độc lập với nhau như sau:

Thuật toán 3.8: Cộng hai điểm sử dụng thuật toán nhân tuần tự

Đầu vào: $P_1 = (X_1, Y_1, Z_1), P_2 = (X_2, Y_2, Z_2)$ biểu diễn trong hệ tọa độ Jacobi

Đầu ra: $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$

1. $T_1 = Z_1^2$
2. $T_2 = Z_2^2$
3. $U_1 = X_1 \cdot T_1$
4. $U_2 = X_2 \cdot T_2$
5. $S_1 = Y_1 \cdot Z_2 \cdot T_2$
6. $S_2 = Y_2 \cdot Z_1 \cdot T_1$
7. $H = U_2 - U_1$
8. $I = (2 \cdot H)^2$
9. $J = H \cdot I$
10. $R = 2 \cdot (S_2 - S_1)$
11. $V = U_1 \cdot I$
12. $X_3 = R^2 - J - 2 \cdot V$
13. $Y_3 = R \cdot (V - X_3) - 2 \cdot S_1 \cdot J$
14. $Z_3 = ((Z_1 + Z_2)^2 - T_1 - T_2) \cdot$H

Thuật toán 3.9: Cộng hai điểm sử dụng thuật toán nhân kép dựa trên SIMD

Đầu vào: $P_1 = (X_1, Y_1, Z_1), P_2 = (X_2, Y_2, Z_2)$ biểu diễn trong hệ tọa độ Jacobi

Đầu ra: $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$

1. $T_1 = Z_1^2, T_2 = Z_2^2$ {NEON}

2. $U_1 = X_1 \cdot T_1, U_2 = X_2 \cdot T_2$ {NEON}

3. $S_1 = Z_2 \cdot T_2, S_2 = Z_1 \cdot T_1$ {NEON}

4. $S_1 = Y_1 \cdot S_1, S_2 = Y_2 \cdot S_2$ {NEON}

5. $H = U_2 - U_1$ {C}

6. $I = 2 \cdot H$ {C}

7. $I = I^2, T_3 = R^2$ {NEON}

7. $J = H \cdot I, V = U_1 \cdot I$ {NEON}

8. $R = 2 \cdot (S_2 - S_1)$ {C}

10. $X_3 = T_3 - J - 2 \cdot V$ {C}

11. $T_3 = V - X_3$ {C}

12. $T_3 = R \cdot T_3, T_4 = S_1 \cdot J$ {NEON}

11. $Y_3 = T3 - 2 \cdot T_4$ {C}

12. $Z_3 = ((Z_1 + Z_2)^2 - T_1 - T_2) \cdot$H {C}

3.4.3.2 Nhân đôi điểm

Tương tự như phép cộng điểm, dựa trên thuật toán nhân đôi điểm (thuật toán 3.10) được trình bày trong *Explicit-Formulas Database* [56], luận án tổ chức lại và đề xuất thuật toán nhân đôi điểm sử dụng phép nhân kép theo thuật toán 3.11, với đặc điểm là các phép nhân, bình phương thông thường được tổ chức thành các cặp phép nhân, cặp phép bình phương hoặc cặp phép nhân và bình phương mà có dữ liệu độc lập với nhau như sau

Thuật toán 3.10: Nhân đôi điểm sử dụng thuật toán nhân tuần tự

Đầu vào: $P_1 = (X_1, Y_1, Z_1)$ biểu diễn trong hệ tọa độ Jacobi

Đầu ra: $P_3 = 2 * P_1 = (X_3, Y_3, Z_3)$

1. $T_0 = delta = Z_1^2$

2. $T_1 = gamma = Y_1^2$

3. $T_2 = beta = X_1 * T_1$

4. $T_3 = X_1 - T_0$

5. $T_4 = X_1 + T_0$

6. $T_3 = alpha = 3 * T_3 * T_4$

7. $X_3 = T_3^2 - 8 * T_2$

8. $Z_3 = (Y_1 + Z_1)^2 - T_1 - T_0$

9. $Y_3 = T_3 * (4 * T_2 - X_3) - 8 * T_1^2$

Thuật toán 3.11: Nhân đôi điểm sử dụng thuật toán nhân kép dựa trên SIMD

Đầu vào: $P_1 = (X_1, Y_1, Z_1)$ biểu diễn trong hệ tọa độ Jacobi

Đầu ra: $P_3 = 2 * P_1 = (X_3, Y_3, Z_3)$

1. $T_0 = delta = Z_1^2, T_1 = gamma = Y_1^2$	{NEON}
2. $T_3 = X_1 - T_0$	{C}
3. $T_4 = X_1 + T_0$	{C}
4. $T_2 = beta = X_1 * T_1, T_3 = T_3 * T_4$	{NEON}
5. $T_3 = alpha = 3 * T_3$	{C}
6. $T_5 = Y_1 + Z_1$	{C}
7. $T_4 = T_3^2; T_5 = T_5^2$	{NEON}
8. $X_3 = T_4 - 8 * T_2$	{C}
9. $Z_3 = T_5 - T_1 - T_0$	{C}
10. $T_4 = 4 * T_2 - X_3$	{C}
12. $T_4 = T_3 * T_4, T_5 = T_1^2$	{NEON}
13. $Y_3 = T_4 - 8 * T_5$	{C}

So sánh thuật toán nhân gốc và thuật toán đề xuất sử dụng nhân song song

Ta ký kiệu M, S là chi phép của phép nhân và phép bình phương thông thường; Mt, St là chi phí của phép nhân kép (thực hiện hai phép nhân song song) theo mô tả ở trong mục 3.4.2.

- Chi phí của phép toán cộng điểm theo mô hình tuần tự là 11M + 5S [56], trong khi đó chi phí của phép toán cộng đề xuất là 5Mt + 2St + 1M + 1S. Do Mt < 2M và Mt < 2S, nên ta thấy rằng chi phí của thuật toán đề xuất (thuật toán 7) có hiệu quả tốt hơn chi phí của thuật toán cộng điểm thông thường.
- Chi phí của phép nhân đôi điểm tuần tự là 3M + 5S [56], chi phí của phép nhân đôi điểm đề xuất (thuật toán 9) là 4Mt.

Bảng 3.4: So sánh chi phí của thuật toán đề xuất

Thuật toán	Phép toán trên điểm	
	Cộng điểm	Nhân đôi điểm
Thuật toán tuần tự [56]	11M+5S	3M + 5S
Thuật toán song song theo đề xuất	5Mt + 2Mt + 1M +1S	4Mt

3.4.4. Nhân vô hướng

Luận án sử dụng thuật toán 3.3 để nhân một số nguyên dương với một điểm, trong đó phép cộng điểm và phép nhân đôi điểm sử dụng thuật toán 3.9 và 3.11 dựa trên thuật toán nhân kép SIMD. Ngoài ra, để tìm NAF(k) sẽ sử dụng thuật toán 3.5.

3.5. Thử nghiệm và đánh giá

3.5.1 Mô tả phương thực nghiệm, kịch bản

Mã thực thi được viết bằng ngôn ngữ C và ngôn ngữ assembly sử dụng thư viện RELIC [54]. Mỗi chức năng được đo nhờ sử dụng hai vòng lặp lồng nhau, với n lần lặp của mỗi vòng: trong đó vòng lặp bên ngoài, thì các đầu vào được sinh ngẫu nhiên và vòng lặp bên trong sẽ thực hiện phép tính toán cần thiết với đầu vào lấy từ vòng lặp bên ngoài. Thời gian tổng cộng được thực hiện bởi thủ tục này, được đo bởi hàm clock_gettime theo đơn vị nano giây (nanosecond), sẽ được chia cho n^2 để tạo kết quả thời gian trung bình của phép toán. Chúng ta chọn n = 128 để đo các phép toán mà có thời gian thực thi nhanh (như phép cộng, nhân... trong số học trường hữu hạn), và n = 32 cho các phép toán có thời gian thực hiện lâu như phép nhân điểm.

Về trình biên dịch và các tham số khi biên dịch:

- Đối với thiết bị Xilinx Zynq (ARMv7) chạy hệ điều hành Linux nhúng, luận án sử dụng trình biên dịch arm-xilinx-linux-gnueabi-gcc với tùy chọn được thiết lập khi biên dịch như sau:

  ```
  arm-xilinx-linux-gnueabi-gcc -O2 -g -mcpu=cortex-a9 -march=armv7-a -mfpu=neon -mfloat-abi=softfp -marm -Deb_choose=eb_choose_asm -DCORTEX=9
  ```

- Đối với thiết bị NXP IMX8M Kit (ARMv8) chạy hệ điều hành Linux nhúng, luận án sử dụng trình biên dịch aarch64-linux-gnu-gcc phiên bản 64bit với các tùy chọn được thiết lập khi biên dịch như sau:

  ```
  aarch64-linux-gnu-gcc -pipe -std=c99 -Wall -O2 -g -march=armv8-a+crypto+crc -mtune=cortex-a53
  ```

Các macro thực hiện đo thời gian thực thi của một chức năng:

- Macro BENCH_BEGIN: Bắt đầu thực hiện đo thời gian thực thi của một chức năng

```
#define BENCH_BEGIN(LABEL)                                          \
        bench_reset();                                              \
        util_print("BENCH: " LABEL "%*c = ", (int)(32 - strlen(LABEL)), ' '); \
        for (int _b = 0; _b < BENCH; _b++)   {                      \
```

- Macro BENCH_END: Hiển thị thời gian trung bình của mỗi lần thực thi một chức năng

```
#define BENCH_END                                                   \
        }                                                           \
        bench_compute(BENCH * BENCH);                               \
        bench_print()                                       \
```

- Macro BENCH_ADD: Đo thời gian thực thi của từng lần chạy một chức năng, cộng thời gian này vào tổng thời gian thực thi để tính thời gian trung bình

```
#define BENCH_ADD(FUNCTION)                 \
        FUNCTION;                           \
        bench_before();                     \
        for (int _b = 0; _b < BENCH; _b++) {    \
                FUNCTION;                   \
        }                                   \
        bench_after();                      \
```

Ví dụ để đo thời gian của phép toán nhân trên trên trường GF(p) trong hai trường hợp:

```
        /* Trường hợp 1: Đo thời gian thực hiện hai phép nhân chưa cải tiến */
        BENCH_BEGIN("dual fp_mul") {
                fp_rand(a);
                fp_rand(b);
                BENCH_ADD(fp_mul(c, a, b));
                BENCH_ADD(fp_mul(c, a, b));
        }
        BENCH_END;
```

```
/* Trường hợp 2: Đo thời gian thực hiện 2 phép nhân song song sử sụng NEON */
BENCH_BEGIN("fp_mul_dual_neon") {
        fp_rand(a);
        fp_rand(b);
        BENCH_ADD(fp_mul_dual_neon(c, d, a, b, a, b));
}
BENCH_END;
```

Kịch bản tiến hành đánh giá hiệu quả:

Sau khi cài đặt thuật toán đề xuất, luận án tiến hành đánh hiệu quả của thuật toán so với thuật toán chưa cải tiến (thuật toán mặc định trong thư viện RELIC). Các so sánh được lần lượt chạy trên hai nền vi xử lý ARMv7 và ARMv8 (lưu ý là thực nghiệm ở đây không so sánh kết quả của thuật toán đề xuất giữa hai nền vi xử lý ARMv7 và ARMv8). Dưới đây là kết quả chạy thuật toán đề xuất (sử dụng thành phần NEON/C) và thuật toán mặc định (viết bằng ngôn ngữ C) trong trường GF(P) và trên đường cong Curve NIST-P256.

3.5.2 Thử nghiệm trên ARMv7

Các kết quả thử nghiệm sau đây được thực hiện trên nền tảng phần cứng: Xilinx Zynq Kit [49] với vi xử lý ARMv7 có tốc độ 1.3 GHz, chạy hệ điều hành Linux nhúng. Công cụ phát triển được sử dụng cho thử nghiệm là arm-xilinx-linux-gnueabi-gcc, phiên bản 4.8.3 để biên dịch chương trình. Luận án thực hiện các thử nghiệm cho thuật toán nhân đã đề xuất đối với ba đường cong: Curve NIST-P256, Curve NIST-P384, và Curve NIST-P521.

Các kết quả thử nghiệm với ARMv7 được trình bày trong các Bảng 3.5, 3.6, và 3.7. Bảng 3.5 là kết quả thực hiện với các phép toán số học trên $\boldsymbol{F_p}$. Bảng 3.6 là kết quả thực hiện với phép toán số học điểm trên đường cong $\boldsymbol{E}(\boldsymbol{F_p})$. Bảng 3.7 là kết quả thực hiện với các hàm nguyên thủy mật mã dựa trên đường cong $\boldsymbol{E}(\boldsymbol{F_p})$.

Sau bảng thống kê, là các hình biểu đồ so sánh hiệu quả thực thi của các phép toán (số học, giao thức) trên đường cong elliptic trong hai trường hợp:

- Khi tích hợp thuật toán đề xuất (nhân kép) thể hiện bằng màu xanh

- Khi sử dụng các phép tính mặc định (nhân tuần tự) thể hiện bằng màu vàng trong thư viện RELIC.

Như trình bày trên Bảng 3.5, hình 3.4 và hình 3.5 là thử nghiệm đánh giá so sánh giữa thời gian thực hiện một phép nhân kép và thời gian thực hiện hai phép nhân (Mul) tuần tự hoặc hai phép bình phương (Sqr) tuần tự (phép toán mặc định trong thư viện RELIC). Kết quả ở cột cuối cùng cho ta thấy, đối với phép nhân và phép bình phương: thuật toán đề xuất nhanh hơn lần lượt là khoảng 30% và 20% so với thuật toán mặc định trong thư viện RELIC.

Giải thích: Ở cột cuối cùng của bảng dưới là tỷ số giữa phép nhân kép và phép nhân tuần tự, tỷ số này phụ thuộc vào kích thước (độ dài bit) của các số hạng và loại phép toán (nhân hay bình phương). Như trong bảng, thì tỷ số này chạy từ 0.69 đến 0.78, nghĩa là phép toán đề xuất (nhân kép) thực hiện chỉ bằng 69% đến 78% so với phép toán thông thường (nhân tuần tự), hay phép toán đề xuất nhanh hơn phép toán thông thường cỡ khoảng 20% đến 30%.

Bảng 3.5: Thời gian thực hiện của phép toán số học trên ARMv7

Độ dài bit	Phép toán số học trên F_p	Thời gian (10^3 nano giây)		Tỷ số (nhân kép / nhân tuần tự)
		Nhân kép	Nhân tuần tự	
256	mul	4.9	6.9	0.71
	sqr	4.9	6.3	0.78
384	mul	9.2	13.5	0.68
	sqr	9.2	13.0	0.71
521	mul	19.6	28.6	0.69
	sqr	19.6	28	0.7

Như trình bày trên Bảng 3.6 là thử nghiệm đánh giá so sánh giữa thời gian thực hiện của phép cộng điểm (Add), nhân đôi điểm (Dbl) và nhân vô hướng (k.P) của thuật toán đề xuất (sử dụng nhân kép) với thời gian thực hiện các phép toán mặc định (nhân tuần tự) tương ứng trong thư viện RELIC. Kết quả ở cột cuối cùng cho ta thấy, đối

với thuật toán đề xuất (sử dụng thành phần NEON) nhanh hơn khoảng từ 20 đến 30% so với thuật toán mặc định trong thư viện RELIC.

Bảng 3.6: Thời gian thực hiện của phép toán số học điểm trên ARMv7

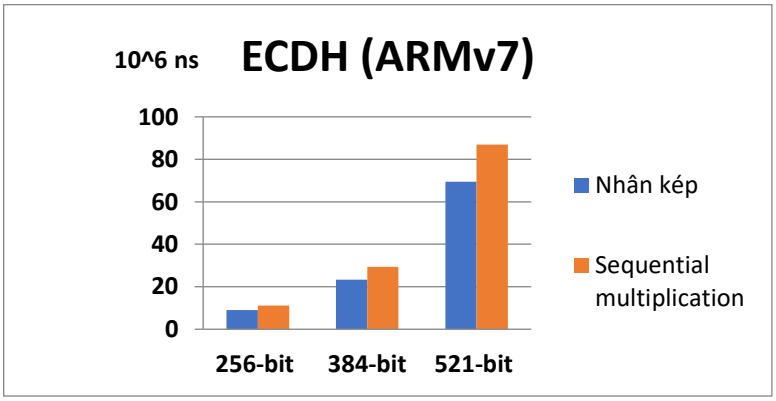
Độ dài bit	Phép toán số học $E(F_p)$	Thời gian (10^3 nano giây)		Tỷ số (nhân kép / nhân tuần tự)
		Nhân kép	Nhân tuần tự	
256	Cộng điểm (Add)	43	57.2	0.75
	Nhân đôi điểm (Dbl)	21.8	30.3	0.72
	Nhân vô hướng (k.P)	8742	10770	0.81
384	Cộng điểm (Add)	81.5	107.8	0.76
	Nhân đôi điểm (Dbl)	40.6	57.3	0.71
	Nhân vô hướng	23672	29476	0.80
521	Cộng điểm (Add)	176.5	238.5	0.74
	Nhân đôi điểm (Dbl)	87.8	125.2	0.70
	Nhân vô hướng (k.P)	68243	85689	0.80

Như trình bày trên Bảng 3.7 là đánh giá hiệu quả thực thi của hai giao thức mật mã (ECDH, ECDSA) khi đã tích hợp thuật toán đề xuất (dựa trên NEON) và khi thực thi giao thức nguyên thủy trong thư viện RELIC. Trên nền ARMv7, các tính hợp thuật toán đề xuất cho giao thức ECDH và ECDSA tăng khoảng từ 10% đến 20% so với nguyên thủy.

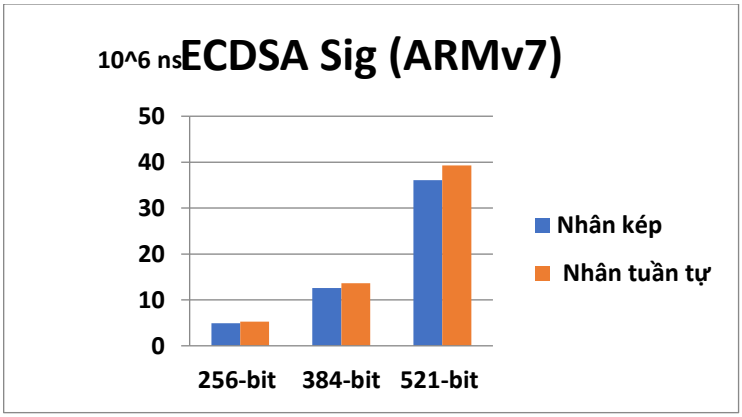
Bảng 3.7: Thời gian thực hiện của tính toán trong giao thức ECDSA và ECDH trên ARMv7

Độ dài bit	Giao thức dựa trên $E(F_p)$	Thời gian (10^6 nano giây)		Tỷ số (nhân kép / nhân tuần tự)
		Nhân kép	Nhân tuần tự	
256	ECDH	9.0	11.2	0.8
	ECMQV	12.1	14.2	0.85
	ECDSA Sig	4.9	5.3	0.92

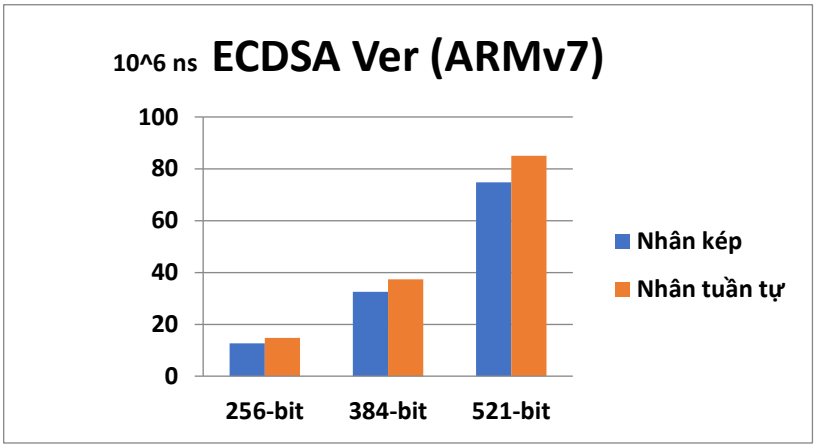
	ECDSA Ver	12.6	14.7	0.86
384	ECDH	23.4	29.3	0.80
	ECMQV	31.2	36.7	0.85
	ECDSA Sig	12.6	13.6	0.93
	ECDSA Ver	32.5	37.3	0.87
521	ECDH	69.4	86.9	0.80
	ECDSA Sig	36.1	39.3	0.92
	ECDSA Ver	74.8	85.0	0.88



Hình 3.4. So sánh thời gian thực hiện của giao thức ECDH trên ARMv7



Hình 3.5: So sánh thời gian thực hiện của thuật toán tạo chữ ký số ECDSA trên ARMv7



Hình 3.6: So sánh thời gian thực hiện của thuật toán kiểm tra chữ ký số ECDSA trên ARMv7

3.5.3 Thử nghiệm trên ARMv8

Các kết quả thử nghiệm sau đây được thực hiện trên nền tảng phần cứng: NXP IMX8M Kit [47] (vi xử lý ARMv8 tốc độ 1.5 GHz) chạy hệ điều hành Linux nhúng. Luận án sử dụng công cụ phát triển aarch64-linux-gnu-gcc để biên dịch chương trình. Các thử nghiệm được thực hiện cho thuật toán nhân đã đề xuất và thuật toán mặc định trong thư viện RELIC đối với ba đường cong: Curve NIST-P256, Curve NIST-P384, và Curve NIST-P521.

Các kết quả thử nghiệm với ARMv8 được trình bày trong các Bảng 3.8, 3.9, và 3.10. Bảng 3.8 là kết quả thực hiện với các phép toán số học trên $\boldsymbol{F_p}$. Bảng 3.9 là kết quả thực hiện với phép toán số học điểm trên đường cong $\boldsymbol{E(F_p)}$. Bảng 3.10 là kết quả thực hiện với các hàm nguyên thủy mật mã dựa trên đường cong $\boldsymbol{E(F_p)}$ với ARMv8. Như trình bày trên Bảng 3.8 là thử nghiệm đánh giá so sánh giữa thời gian thực hiện một phép nhân kép và thời gian thực hiện hai phép nhân (Mul) hoặc hai phép bình phương (Sqr) tuần tự (phép toán mặc định trong thư viện RELIC). Kết quả ở cột cuối cùng cho ta thấy, đối với phép nhân và phép bình phương: thuật toán đề xuất nhanh hơn lần lượt là khoảng 40% và 25% so với thuật toán mặc định trong thư viện RELIC.

Bảng 3.8: Thời gian thực hiện của phép toán số học trên ARMv8

Độ dài bit	Phép toán số học trên F_p	Thời gian (10^3 nano giây)		Tỷ số
		Nhân kép	Nhân tuần tự	

				(nhân kép / nhân tuần tự)
256	mul	1.8	2.9	0.62
	sqr	1.8	2.4	0.75
384	mul	3.6	6.1	0.60
	sqr	3.6	5.1	0.71
521	mul	8.4	13.8	0.61
	sqr	8.4	11.0	0.76

Như trình bày trên Bảng 3.9 là thử nghiệm đánh giá so sánh giữa thời gian thực hiện của phép cộng điểm (Add), nhân đôi điểm (Dbl) và nhân vô hướng (k.P) của thuật toán đề xuất với thời gian thực hiện các phép toán mặc định tương ứng trong thư viện RELIC. Kết quả ở cột cuối cùng cho ta thấy, đối với thuật toán đề xuất (sử dụng thành phần NEON) nhanh hơn khoảng từ 20 đến 30% so với thuật toán mặc định trong thư viện RELIC.

Bảng 3.9: Thời gian thực hiện của phép toán số học điểm trên ARMv8

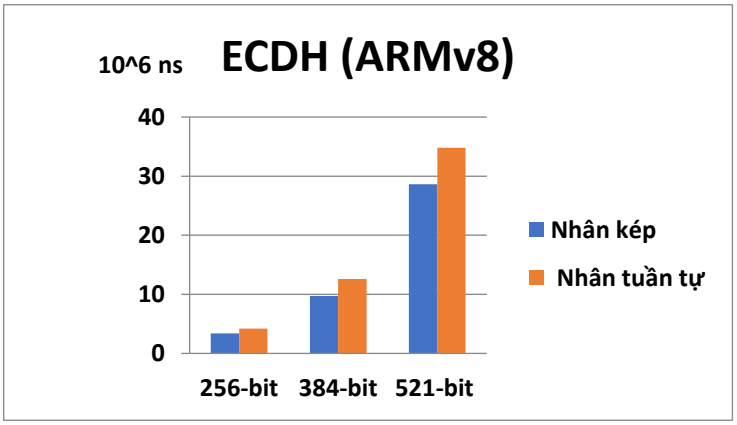
Độ dài bit	Phép toán số học $E(F_p)$	Thời gian (10^3 nano giây)		Tỷ số (nhân kép / nhân tuần tự)
		Nhân kép	Nhân tuần tự	
256	Cộng điểm (Add)	16.2	22.8	0.71
	Nhân đôi điểm (Dbl)	8.4	12.2	0.69
	Nhân vô hướng (k.P)	3376	4149	0.81
384	Cộng điểm (Add)	32.7	47.6	0.69
	Nhân đôi điểm (Dbl)	17.5	24.0	0.73
	Nhân vô hướng	9645	12409	0.78
521	Cộng điểm (Add)	74.9	100.8	0.74
	Nhân đôi điểm (Dbl)	37.6	50.3	0.75
	Nhân vô hướng (k.P)	28696	34825	0.82

Như trình bày trên Bảng 3.10 là đánh giá hiệu quả thực thi của hai giao thức mật mã (ECDH, ECDSA) khi đã tích hợp thuật toán đề xuất (dựa trên NEON) và khi thực

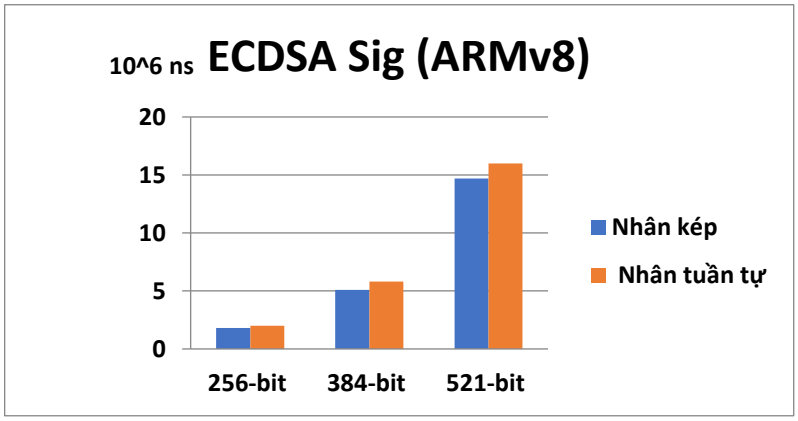
thi giao thức nguyên thủy trong thư viện RELIC. Trên nền ARMv8, các tính hợp thuật toán đề xuất cho giao thức ECDH và ECDSA tăng khoảng từ 10% đến 20% so với nguyên thủy.

Bảng 3.10: Thời gian thực hiện của tính toán trong giao thức ECDSA và ECDH trên ARMv8

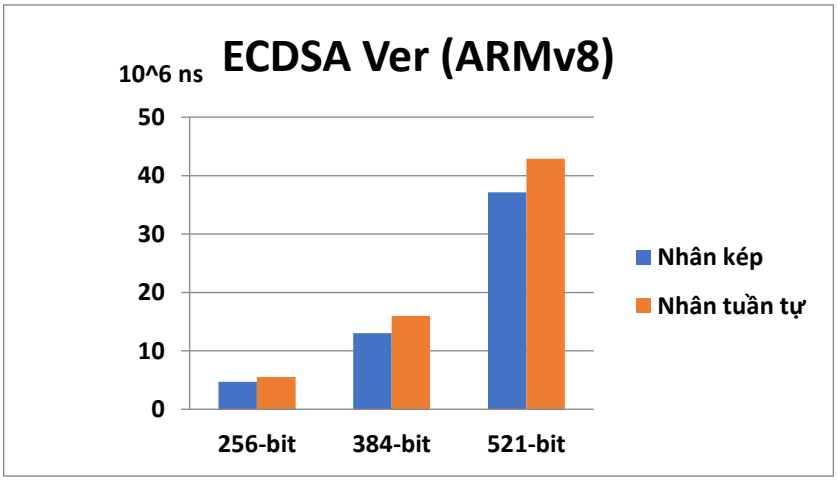
Độ dài bit	Giao thức dựa trên $E(F_p)$	Thời gian (10^6 nano giây)		Tỷ số (nhân kép / nhân tuần tự)
		Nhân kép	Nhân tuần tự	
256	ECDH	3.4	4.2	0.81
	ECMQV	4.6	5.3	0.86
	ECDSA Sig	1.8	2.0	0.90
	ECDSA Ver	4.7	5.5	0.85
384	ECDH	9.7	12.6	0.77
	ECMQV	12.8	15.8	0.81
	ECDSA Sig	5.1	5.8	0.88
	ECDSA Ver	13.0	16.0	0.81
521	ECDH	28.6	34.8	0.82
	ECMQV	37.8	43.1	0.88
	ECDSA Sig	14.7	16.0	0.92
	ECDSA Ver	37.1	42.9	0.86



Hình 3.7: So sánh thời gian thực hiện của giao thức ECDH trên ARMv8



Hình 3.8: So sánh thời gian thực hiện của thuật toán tạo chữ ký số ECDSA trên ARMv8



Hình 3.9: So sánh thời gian thực hiện của thuật toán kiểm tra chữ ký số ECDSA trên ARMv8

3.6. Kết luận chương 3

Chương 3 của luận án tập trung cải tiến tốc độ cho các thuật toán mật mã dựa trên đường cong Elliptic trên trường số nguyên tố dựa trên đề xuất cải tiến thuật toán NAF và khai thác đặc điểm phần cứng trên nền vi xử lý ARM (cả ARMv7 và ARMv8) để nâng cao hiệu quả của các phép toán số học (cộng, nhân đôi và nhân vô hướng) trên đường cong Elliptic trong trường số nguyên tố. Các kết quả chính của chương 3 được trình bày trong hai công trình công bố [J2, J3].

Trên cơ sở nghiên cứu, luận án đã đề xuất một thuật toán cho việc tìm dạng biểu diễn không liền kề NAF của một số nguyên k cho trước. Thuật toán này tiết kiệm được ba phép tính số học so với thuật toán nguyên thủy. Tiếp theo, với dạng biểu diễn hầu

không liền kề aNAF mới của số k, luận án đã đề xuất rút gọn một phép tính bình phương khi thực hiện phép "bình phương – nhân" đối với phép tính lũy thừa k trên các trường hữu hạn, hoặc tương ứng một phép tính gấp đôi khi thực hiện phép "gấp đôi - cộng" trong phép nhân điểm trên đường cong elliptic.

Dựa trên đặc điểm SIMD trên vi xử lý ARM, luận án đã đề xuất cải tiến thuật toán cộng điểm và nhân đôi điểm nhờ phương pháp nhóm theo từng cặp phép nhân, theo cặp phép bình phương hoặc kết hợp cặp của phép nhân và phép bình phương. Các kết quả thực nghiệm đã chứng tỏ hiệu quả của thuật toán đề xuất như sau:

- Tăng khoảng từ 20% đến 30% đối với phép toán cơ bản (cộng, nhân đôi, nhân vô hướng) so với thuật toán mặc định trong thư viện RELIC
- Tăng từ 10% đến 20% trong các tính toán của giao thức ECDH và ECDSA so với các tính toán mặc định trong thư viện RELIC.

Các thực nghiệm đánh giá kết quả được so sánh trên cùng một nền tảng phần cứng (ARMv7 hoặc ARMv8) và phần mềm

KẾT LUẬN

Các hệ thống nhúng dựa trên vi xử lý ARM đang được ứng dụng rộng rãi trên nhiều dòng thiết bị hiện nay như: Thiết bị di động, máy tính bảng, thiết bị IoT, các hệ thống nhúng trên phương tiện giao thông…. Đặc điểm chung của hệ thống nhúng là hạn chế về tài nguyên và năng lực xử lý. Mặc dù cấu hình của nhiều thiết bị đã được cải thiện đáng kể, song so với các máy tính có năng lực xử lý mạnh, các hệ thống nhúng thường có kích thước nhỏ hơn, khả năng xử lý yếu hơn, bộ nhớ nhỏ hơn và yêu cầu tiêu thụ ít năng lượng do sử dụng nguồn pin. Việc triển khai các giải pháp an toàn bảo mật thông tin cho các hệ thống nhúng có thêm nhiều thách thức so với các hệ thống máy tính truyền thống.

Hệ mật đường cong Elliptic đã được đề xuất cho các hệ thống nhúng do kích thước khóa nhỏ hơn nhiều so với các hệ mật khóa công khai khác. Các lược đồ mã hóa ECC đã được đề xuất điển hình là lược đồ thỏa thuận khóa Diffie Hellman trên đường cong elliptic (ECDH) và thuật toán chữ ký số trên đường cong elliptic (ECDSA). ECC có nhiều ưu việt và phù hợp cho bảo mật dữ liệu trên các thiết bị nhúng. Tuy nhiên, việc nghiên cứu đưa ra các thuật toán an toàn và hiệu quả cho ECC trong các hệ thống nhúng có tài nguyên hạn chế vẫn đang là vấn đề có nhiều thách thức (thực thi, đáp ứng thời gian thực, tài nguyên sử dụng...)

Luận án tập trung vào nghiên cứu các giải pháp cải tiến về hiệu quả khi triển khai các thuật toán, lược đồ thỏa thuận khóa, ký số dựa trên hệ mật đường cong Elliptic trên các hệ thống, vi xử lý nhúng (chủ yếu là dòng ARM Cortex-A) được sử dụng phổ biến trong thực tiễn. Các giải pháp được đề xuất trong luận án đã góp phần tăng cường khả năng bảo mật thông tin trên các thiết bị nhúng, đáp ứng nhu cầu cấp thiết trong thực tiễn hiện nay về bảo mật dữ liệu, ngăn chặn tấn công nghe lén và rò rỉ thông tin cá nhân.

A. Đóng góp mới của luận án

- Đề xuất phương pháp nhân phân tầng hai số hạng trên trường hữu hạn dựa trên hai thuật toán nhân cơ bản là thuật toán nhân theo phương pháp phổ thông và thuật toán nhân theo phương pháp Karatsuba. Với phương pháp phân tầng, luận án đã xây dựng

được thuật toán nhân có chi phí tốt nhất trong các trường hợp cụ thể cũng như đưa ra công thức để xác định chi phí của thuật toán đó. Thuật toán đề xuất đã được kiểm chứng về tính hiệu quả trên nền vi xử lý nhúng ARMv7 và ARMv8.

- Đề xuất, cải tiến thuật toán nhân vô hướng (nhân giữa một điểm và một số nguyên dương) của hệ mật đường cong Elliptic trên trường nguyên tố dựa trên đề xuất cải tiến thuật toán NAF và đề xuất nâng cao hiệu quả của các phép toán số học (cộng điểm, nhân đôi điểm) theo phương pháp song song hai phép nhân.

B. Đánh giá ưu nhược điểm của các đề xuất trong luận án và hướng phát triển tiếp theo của đề tài

- ❖ Ưu điểm: Các đề xuất nâng cao hiệu quả về sử dụng mật mã đường cong elliptic trên thiết bị nhúng trong luận án sẽ góp phần tăng cường khả năng bảo mật thông tin trên các thiết bị nhúng. Đặc biệt có ý nghĩa trong lĩnh vực an ninh quốc phòng: Khả năng mềm dẻo ở tùy biến tham số, bản địa hóa thuật toán, khả năng đáp ứng thời gian thực đối với các phần mềm bảo mật.
- ❖ Nhược điểm và hạn chế: Thuật toán nhân phân tầng được xây dựng trên trên cơ sở hai thuật toán cơ bản là thuật toán theo phương pháp phổ thông và thuật toán nhân theo phương pháp Karatsuba. Tuy nhiên, cần có những nghiên cứu đề xuất thuật toán phân tầng dựa trên các thuật toán cơ sở khác cũng như cho phép toán khác như: modulo, bình phương...
- ❖ Đề xuất hướng nghiên cứu tiếp theo:
    - Nghiên cứu đề xuất thuật toán phân tầng dựa trên các thuật toán cơ sở khác cũng như cho phép toán khác trong trường hữu hạn: modulo, bình phương...
    - Nâng cao hiệu quả các các lược đồ, thuật toán mật mã dựa trên ECC cho các hệ vi xử lý nhúng khác như dòng ARM Cortex-M.
    - Nghiên cứu đảm bảo an toàn chống lại tấn công kênh kề cho các thuật toán mật mã của hệ mật ECC hoạt động trên nền vi xử lý nhúng.

DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC ĐÃ CÔNG BỐ

[**J1**] Phạm Văn Lực, Võ Tùng Linh, "*Phép nhân số nguyên lớn*", Tạp chí Nghiên cứu KH&CN quân sự, Số 63, 2019, pp.111-120.

[**C1**] Pham Van Luc, Vo Tung Linh, Hoang Dang Hai, Leu Duc Tan, "*Fast Binary Field Mutiplication on ARMv7 Embedded Processors*", in Proc. of 4th IEEE Int. Conf. on Recent Advances in Signal Processing, Telecommunications & Computing (SigTelCom 2020), Hanoi, Vietnam, Aug. 2020.

[**C2]** Pham Van Luc, Hoang Dang Hai, Leu Duc Tan, "*Multi-layer Multiplication in Binary Field on ARMv8 Processors*", in Proc. of IEEE Int. Conf. on Advanced Technologies for Communications (ATC 2020), Nhatrang, Vietnam, Oct. 2020.

[**J2**] Pham Van Luc, Leu Duc Tan, "*Extensions for NAF representation of positive integers*", Journal of Science in Information Technology and Communications, vol. CS.01, No.1, 2021, pp.62-66.

[**J3**] Pham Van Luc, Hoang Dang Hai, Leu Duc Tan, "*Improving the Efficiency of Point Arithmetic on Elliptic Curves using ARM Processors and NEON*", ISI Q2, International Journal of Network Security, Vol. 24, No. 2, 2022, pp. 364-376

TÀI LIỆU THAM KHẢO

Tiếng Anh

[1]. Tammy Noergaard, "*Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers"*, Newnes, 2nd edition, 2012.

[2]. ARM Holdings, "*Q3 2020 roadshow slides",* 2020. [Online]: Accessed on 26 Aug. 2021, Available: https://group.softbank/system/files/pdf/ir/-presentations/2020/arm-roadshow-slides_q3fy2020_01.pdf.

[3]. Jason D. Bakos, "*Embedded Systems ARM Programming and Optimization"*, Morgan Kaufmann, 1st edition, 2015.

[4]. ARM, "*Introducing neon development article"*, ARM Limited, 2009.

[5]. Sarah L. Harris, David Harris, *Digital Design and Computer Architecture ARM Edition*, Morgan Kaufmann, 1st edition 2015.

[6]. A. Weimerskirch, C. Paar, "Generalizations of the karatsuba algorithmfor efficient implementations," University of Ruhr, Bochum, Germany, Tech. Rep., 2003.

[7]. Darrel Hankerson, Alfred Menezes, Scott Vanstone, *Guide to elliptic Curve Cryptography* , Springer , 2004.

[8]. National Institute of Standards and Technology (NIST), "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", Gaithersburg, Technical Report, 2018.

[9]. National Institute of Standards and Technology (NIST), "*DSS Digital Signature Standard (DSS)*", Federal Information Processing Standards Publication 186-2, 2000.

[10]. Mohit Arora, "*Embedded System Design - Introduction to SoC System Architecture*", ISBN 978-0-9972972-0-1, Learning Bytes Publishing, 2016.

[11]. Tammy Noergaard, "*Embedded Systems Architecture - A Comprehensive Guide for Engineers and Programmers*", ISBN: 0-7506-7792-9, Second edition, Copyright © Elsevier Inc, 2013.

[12]. M. Amara, A. Siad, "*Hardware Implementation of Elliptic Curve Point Multiplication over GF(2 m) for ECC protocols"*, International Journal for Information Security Research (IJISR), Vol. 2, No. 1, 2012, pp.106-112.

[13]. Shuai Liu, Lei Ju, Xiaojun Cai, Zhiping Jia, Zhiyong Zhang High, *Performance FPGA Implementation of Elliptic Curve Cryptography over Binary Fields*, in Proc. of 13th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, Beijing, China, Sept. 2014.

[14]. D. J. Bernstein, P. Schwabe, *NEON crypto*, in Proc. of Int. Workshop on Cryptographic Hardware and Embedded Systems, CHES 2012, Springer, 2012, pp. 320-339.

[15]. Mike Hamburg, *Fast and compact elliptic-curve cryptography*, Cryptology ePrint Archive, Report 2012/309, 2012.

[16]. L.Kowada, R.Portugal, C.M.H. Figueiredo, "*Reversible Karatsuba's Algorithm*", Journal of Universal Computer Science, vol 12, no.5, 2006, pp. 499-511.

[17]. D. Câmara, C.P. Gouvêa, J. López, R. Dahab, "*Fast software polynomial multiplication on ARM Processors using the NEON Engine*", in Proc of Int. Conf. on Security Engineering and Intelligence Informatics, Springer, 2013, pp. 137-154.

[18]. H. Seo, Z. Liu, Y. Nogami, J. Choi, and H. Kim, *Binary field multiplication on ARMv8*, Security and Communication Networks, vol. 9, no. 13, 2016, pp. 2051-2058.

[19]. ARM, *NEON Programmer's Guide*, ARM Limited, 2013.

[20]. Michael Hutter and Erich Wenger, *Fast multi-precision multiplication for public-key cryptography on embedded microprocessors*, in Proc. of Int. Workshop on Cryptographic Hardware and Embedded Systems, CHES 2011, Springer, 2012, pp. 459-474.

[21]. G. Mitra, B. Johnston, A.P. Rendell, *Use of SIMD Vector Operations to Accelerate Application Code Performance on Low-Powered ARM and Intel Platforms*, in Proc. of IEEE Int. Conf. on on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 2013.

[22]. H. Seo, Z. Liu, J. Grobschadl, J. Choi, H. Kim, "*Montgomery Modular Multiplication on ARM-NEON Revisited*", in Proc of Int. Conf. on Information Security and Cryptology, ICISC 2014, LNCS 8949; 2015, p. 328-342.

[23]. ARM, *Cortex-A7 instruction cycle timings*, 2014, [Online]: Accessed on 10 Aug. 2021, Available: https://hardwarebug.org/2014/05/15/cortex-a7-instruction-cycle-timings/

[24]. Nicolas Sklavos, *On the Hardware Implementation Cost of CryptoProcessors Architectures*, Information Security Journal A Global Perspective, Vol. 19, No. 2, 2010, pp.53-60.

[25]. Alfred Menezes, Paul van Oorschot, Scott Vanstone, "*Handbook of Applied Cryptography*", CRC Press, 1996.

[26]. Paul G. Comba, "*Exponentiation cryptosystems on the IBM PC*", IBM Systems Journal, Vol. 29, No. 4, 1990, pp. 526-538.

[27]. ARM, *Cortex™-A9 Technical Reference Manual*, ARM Limited, 2010

[28]. ARM, *Cortex-A9 NEON Media Processing Engine Technical Reference Manual Revision: r4p1*, ARM Limited, 2012

[29]. J. Lopez and R. Dahab, "*High-speed software multiplication in F(2^m)*", Int. Conf. on Cryptology in India, Springer, 2000, pp. 203-212.

[30]. H. Seo, Z. Liu, J. Choi, and H. Kim. "*Karatsuba-block-comb technique for elliptic curve cryptography over binary fields*", Journal of Security and Communication Networks, vol.8, no.17, Nov. 2015, pp. 3121-3130.

[31]. C.Y. Lee, C.C. Fan, J. Xie, S.M. Yuan. "*Efficient Implementation of Karatsuba Algorithm based Three-Operand Multiplication over Binary Extension Field*", IEEE Access. Vol.6, June 2018, pp. 38234-38242.

[32]. Darrel Hankerson, Julio López Hernandez, and Alfred Menezes, *Software Implementation of Elliptic Curve Cryptography over Binary Fields*, in Proc. of Int. Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, Springer, 2000, pp. 1-24.

[33]. R.C. Marquez, A.J.C. Sarmiento, S. Sánchez-Solano, "*Speeding up elliptic curve arithmetic on ARM processors using NEON instructions*", RIELAC, vol. 41, no.3, Sept.–Dec. 2020, ISSN: 1815-5928, pp. 1-20.

[34]. Z. Liu, J. Groschdl, Z. Hu, K. Jrvinen, H. Wang, and I. Verbauwhede, "*Elliptic curve cryptography with efficiently computable endomorphisms and its hardware implementations for the internet of things,*" IEEE Trans. Computers, vol. 66, no. 5, May 2017, pp. 773–785

[35]. H.Cheng, J. Grosschaedl, J. Tian, P.B. Ronne, P.Y. Ryan, "*High-Throughput Elliptic Curve Cryptography Using AVX2 Vector Instructions*", Intl. Conf. on Selected Areas in Cryptography (SAC 2020), LNCS, vol. 12804, Springer, 2020, pp. 698-719.

[36]. J.W. Bos, P. L. Montgomery, D. Shumow, G. M. Zaverucha, "*Montgomery multiplication using vector instructions*", In Proc. of Conf. Selected Areas in Cryptography (SAC 2013), LNCS Vol. 8282, Springer, 2013, pp. 471-489.

[37]. A. Faz-Hernandez, P. Longa, A.H. Sanchez, "*Efficient and secure algorithms for glv-based scalar multiplication and their implementation on GLV-GLS curves*", in Proc. of RSA Conference: Topics in Cryptology CT-RSA, LNCS Vol. 8366, Springer, 2014, pp. 1-27.

[38]. Intel Corporation, "*Using streaming SIMD extensions (SSE2) to perform big multiplications*", Application note AP-941, July 2000. [Online]: Accessed on 10 Aug. 2021, https://www.intel-vintage.info/intelotherresources.htm.

[39]. A. H. Sánchez, F. Rodríguez-Henríquez, "*NEON implementation of an attribute based encryption scheme*", in Proc of Int. Conf. on Applied Cryptography and Network Security, ACNS 2013, Springer, 2013, pp. 322-338.

[40]. A. J. Menezes, "*Elliptic curve public key cryptosystems*," The Springer International Series in Engineering and Computer Science, ISBN 978-1-4615-3198-2, Vol.234, Springer US, 1993.

[41]. Ruan de Clercq, Leif Uhsadel, Anthony Van Herrewege, Ingrid Verbauwhede, "*Ultra Low-Power implementation of ECC on the ARM Cortex-M0+*", IEEE Design Automation Conference (DAC), June 2014.

[42]. Shujie Cui, Johann Großschädl, Zhe Liu, Qiuliang Xu, "*High-Speed Elliptic Curve Cryptography on the NVIDIA GT200 Graphics Processing Unit*," Springer International Publishing, 2014

[43]. Patrick Longa, "*FourQNEON: Faster Elliptic Curve Scalar Multiplications on ARM Processors*," SAC 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, pp.501-519

[44]. S. G Liu, Y.Y Hu, and L. Wei, "*Elliptic Curve Scalar Multiplication Algorithm Based on Side Channel Atomic Block over GF(2^m)*," International Journal of Network Security, Vol.23, No.6, 2021, pp.1005-1011

[45]. S. G Liu, S.J An, and Y.W Du, "*Efficient and Secure Elliptic Curve Scalar Multiplication Based on Quadruple-and-Add*," International Journal of Network Security, Vol.23, No.5, 2021, pp.750-757

Tiếng Việt

[46]. Hoàng Văn Quân, "*Nghiên cứu giải pháp nâng cao hiệu quả bảo mật thông tin trên mạng truyền số liệu đa dịch vụ,*" Luận án tiến sỹ của Viện Khoa học Công nghệ Quân sự, 2016

[47]. Nguyễn Đức Mạnh, *"Nghiên cứu giải pháp nâng cao tính an toàn cho giao thức SSL/TLS*," Luận án tiến sỹ của Viện Khoa học - Công nghệ Quân sự, 2021

[48]. Trần Duy Lai, "*Nghiên cứu xây dựng chuẩn mật mã Quốc gia Việt Nam,*" Đề tài cấp Nhà nước, 2009

Địa chỉ Website

[49]. Xilinx, *Zynq-7000 SoC ZC702 Evaluation Kit*. [Online]: Accessed on 10 Aug. 2021, https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html

[50]. NXP, *RD-IMX6Q-SABRE: SABRE Board for Smart Devices Based on the i.MX 6Quad Applications Processors*. [Online]: Accessed on 10 Aug. 2021, https://www.nxp.com/design/development-boards/i.mx-evaluation-and-development-boards/sabre-board-for-smart-devices-based-on-the-i.mx-6quad-applications-processors:RD-IMX6Q-SABRE

[51]. NXP, *Evaluation Kit for the i.MX 8M Mini Applications Processor*. [Online]: Accessed on 10 Aug. 2021, https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/evaluation-kit-for-the-i-mx-8m-mini-applications-processor:8MMINILPD4-EVK

[52]. SAMSUNG, *Samsung Galaxy Tab S2 9.7*. [Online]: Accessed on 10 Aug. 2021, https://www.gsmarena.com/samsung_galaxy_tab_s2_9_7-7438.php

[53]. BlueKrypt, *Cryptographic Key Length Recommendation*. [Online]: Accessed on 10 Aug. 2021, https://www.keylength.com/en/4/

[54]. RelicToolKit, [Online]: Accessed on 10 Aug. 2021, https://github.com/relic-toolkit

[55]. OpenSSL Cryptography and SSL/TLS Toolkit. [Online]: Accessed on 10 Aug. 2021, https://github.com/openssl/openssl

[56]. Daniel J. Bernstein, Tanja Lange, etal., "Explicit-Formulas Database," [Online]. Available from: https://hyperelliptic.org/EFD/.

[57]. IETF, Internet Key Exchange Protocol Version 2 (IKEv2), RFC 5996. [Online]: Accessed on 10 Aug. 2021, https://datatracker.ietf.org/doc/html/rfc5996

[58]. IETF, The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446. [Online]: Accessed on 10 Aug. 2021, https://datatracker.ietf.org/doc/html/rfc8446

[59]. OpenVPN. [Online]: Accessed on 10 Aug. 2021, https://openvpn.net/

[60]. OpenMP. [Online]: Accessed on 10 Aug. 2021, https://www.openmp.org/

[61]. The GNU Multiple Precision Arithmetic Library (GMP). [Online]: Accessed on 10 Aug. 2021, https://gmplib.org/