

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA ĐIỆN TỬ - VIỄN THÔNG

Hoàng Hữu Tú

**TRIỂN KHAI GIAO THỨC TRUYỀN DỮ LIỆU BẢO
MẬT CHO HỆ THỐNG IOT GIỚI HẠN TÀI NGUYÊN
SỬ DỤNG TRAO ĐỔI KHÓA ECDH VÀ MÃ HÓA
NHE ASCON-128a**

KHÓA LUẬN TỐT NGHIỆP CỦ NHÂN
CHƯƠNG TRÌNH CHÍNH QUY

Tp. Hồ Chí Minh, tháng 07/2025

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐIỆN TỬ - VIỄN THÔNG

Hoàng Hữu Tú - 21200249

**PHÁT TRIỂN KIẾN TRÚC KHUNG TRUYỀN DỮ
LIỆU BẢO MẬT ĐA LỚP CHO IOT KẾT HỢP CƠ
CHẾ TRAO ĐỔI KHÓA ĐỘNG VÀ MẬT MÃ HÓA
NHE ASCON-128a**

KHÓA LUẬN TỐT NGHIỆP CỦ NHÂN
CHƯƠNG TRÌNH CHÍNH QUY

GIÁO VIÊN HƯỚNG DẪN
PGS.TS. Lê Đức Hùng

Tp. Hồ Chí Minh, tháng 07/2025

Lời cam đoan

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu và kết quả nghiên cứu trong luận văn này là trung thực và không trùng lặp với các đề tài khác.

Lời cảm ơn

Tôi xin chân thành cảm ơn ...

Mục lục

Lời cam đoan	i
Lời cảm ơn	ii
Đề cương chi tiết	iii
Mục lục	iii
Tóm tắt	vii
1 Giới thiệu	1
1.1 Đặt vấn đề	1
1.2 Mục tiêu của đề tài	2
1.3 Giải pháp đề xuất	2
1.4 Đóng góp của đề tài	2
1.5 Bố cục của khóa luận	3
2 Các công trình liên quan	4
2.1 Các công trình nghiên cứu về kiến trúc IoT bảo mật	4
2.2 Các mối đe dọa và loại tấn công phổ biến	6
2.2.1 Dò khóa bí mật của hệ thống	6
2.2.2 Tấn công Man-in-the-Middle	8
2.2.3 Tấn công phát lại (Replay Attack)	9
2.3 Khoảng trống nghiên cứu	9
3 Phương pháp đề xuất	10
3.1 Cơ sở lý thuyết	10
3.1.1 Kiến trúc IoT bốn lớp	10
3.1.2 Giao thức MQTT	11
3.1.3 Mật mã hóa nhẹ ASCON-128a	12
3.1.4 Thuật toán trao đổi khóa Elliptic Curve Diffie-Hellman	15
3.1.5 Hàm băm nhẹ ASCON	18
3.2 Phương pháp thực hiện	19
3.2.1 Tổng quan hệ thống IoT	19
3.2.2 Mô hình giao tiếp của hệ thống	22

3.2.3	Phương pháp tạo khóa phiên từ khóa bí mật sử dụng trong mã hóa và giải mã dữ liệu	23
3.2.4	Kiến trúc khung truyền giao tiếp giữa các lớp trong hệ thống IoT	26
3.2.5	Cơ chế safe counter	32
4	Kết quả triển khai và phân tích	34
4.1	Mô hình thực nghiệm	34
4.2	Kết quả, phân tích và đánh giá	34
4.2.1	Kết quả thực hiện	34
4.2.2	Đánh giá	37
4.2.3	Phân tích công suất tiêu thụ	42
4.2.4	Phân tích thông lượng và hiệu suất tải dữ liệu	43
4.2.5	Phân tích khả năng bảo mật của hệ thống	45
5	Kết luận	48
Danh mục công trình của tác giả		49
Tài liệu tham khảo		50

Danh sách hình

2.1	Tấn công MITM trong nghe lén dữ liệu	8
2.2	Tấn công MITM trong trao đổi khóa	8
2.3	Minh họa cuộc tấn công phát lại	9
3.1	Mô hình IoT 4 lớp	10
3.2	Giao thức MQTT	12
3.3	So sánh hiệu năng giữa Ascon và AES được trích từ bài báo	13
3.4	Quá trình mã hóa của ASCON với dữ liệu liên kết	14
3.5	Khối 5-bit S-box của ASCON	15
3.6	Đường cong elliptic $y^2 = x^3 - 2x + 5$	16
3.7	Quá trình thực hiện trao đổi khóa ECDH	17
3.8	Chế độ băm Ascon-Hash	19
3.9	Mô hình tổng quan hệ thống	20
3.10	Mô hình giao tiếp của hệ thống	22
3.11	Luồng hoạt động của hệ thống	24
3.12	Quy trình tạo khóa phiên	25
3.13	Khung truyền khởi tạo	26
3.14	Khung truyền trao truyền dữ liệu từ gateway xuống lớp nhận thức	28
3.15	Khung truyền trong truyền dữ liệu từ lớp nhận thức lên gateway	28
3.16	Tính toán thẻ xác thực (Auth Tag) trong trao đổi khóa	29
3.17	Bộ khung truyền trong giao tiếp giữa gateway và máy chủ	30
3.18	Quá trình phân giải khung truyền	31
3.19	Hàm phân giải khung truyền GS2	31
4.1	Kết quả thực hiện trên MCU STM32F411VET6	35
4.2	Quá trình khởi tạo khi reboot	35
4.3	Quá trình trao đổi khóa giữa gateway và máy chủ	36
4.4	Quá trình truyền dữ liệu	36
4.5	Quá trình thực hiện cơ chế <i>safe counter</i>	37
4.6	Đánh giá hiệu suất thực thi của STM32F411VET6	38
4.7	Đánh giá hiệu suất thực thi của ESP32	38
4.8	Thời gian thực thi quá trình trao đổi khóa trên ESP32	39
4.9	Số lượng gói tin truyền mỗi phút	40
4.10	Tỷ lệ truyền gói tin mỗi phút	40
4.11	Dung lượng tiêu thụ của STM32F411VET6	41

4.12 Dung lượng tiêu thụ của ESP32-WROOM-32	41
4.13 Cảm biến dòng INA219	42
4.14 Mức tiêu thụ năng lượng của STM32F411VET6	42
4.15 Mức tiêu thụ năng lượng của ESP32 Gateway	43
4.16 Mức tiêu thụ năng lượng của toàn bộ hệ thống	43

Danh sách bảng

3.1	Bảng tham số cho ASCON	15
3.2	Ratio of DH Security to EC Security at Different Security Levels	18
4.1	Cấu hình máy chủ	34
4.2	Bảng đánh giá hiệu năng trong 120 phút	40
4.3	Dung lượng bộ nhớ RAM và FLASH	40
4.4	Hiệu suất tải với các kích thước tải khác nhau	45

Chương 1: Giới thiệu

Chương 1 đưa ra các nội dung tổng quan của khóa luận tốt nghiệp. Trong đó, các nội dung được trình bày là việc đặt vấn đề, mục tiêu hướng tới, giải pháp đề xuất, đóng góp, và bối cảnh của khóa luận.

1.1 Đặt vấn đề

Internet of Things (IoT) đã và đang trở thành một trong những trụ cột quan trọng trong sự phát triển của các công nghệ hiện đại, mở ra nhiều đột phá trong các lĩnh vực như chăm sóc sức khỏe, tự động hóa và các hệ thống thông minh [3]. Tuy nhiên, phần lớn các thiết bị IoT hiện nay đều hoạt động trong môi trường có tài nguyên hạn chế, bao gồm tốc độ xử lý thấp, dung lượng pin hạn chế, cũng như kết nối mạng không ổn định. Những hạn chế này khiến các thiết bị dễ bị ảnh hưởng trước các mối đe dọa về bảo mật [28], làm gia tăng nguy cơ mất gói tin trong quá trình truyền dữ liệu. Những kẻ tấn công thường lợi dụng các điểm yếu này để khai thác các lỗ hổng bảo mật trong phần cứng, phần mềm và giao thức để nhắm vào hệ sinh thái IoT trong các hệ thống doanh nghiệp, công nghiệp và chính phủ. Do đó, tính bảo mật, tính toàn vẹn và tính khả dụng của các hệ thống này bị suy yếu gây ra các thiệt hại vô cùng lớn.

Để xây dựng một hệ thống IoT bảo mật yêu cầu phải có một thuật toán mã hóa đủ mạnh. Các thuật toán mã hóa truyền thống như AES hoặc RSA, tuy hiệu quả và phổ biến trong các hệ thống có tài nguyên dồi dào, lại gặp nhiều khó khăn khi triển khai trên các thiết bị IoT do yêu cầu cao về tính toán và bộ nhớ [6]. Vì vậy, khi triển khai hệ thống IoT phải cân nhắc các thuật toán mã hóa nhẹ, phù hợp với các thiết bị IoT hiện nay. Một điều cần chú trọng đó là khi thiết kế các hệ thống IoT, cần xem xét các cơ chế bảo mật ở nhiều lớp [30]. Đối với mô hình hệ thống IoT bốn lớp, các lớp trung gian như lớp nhận thức (perceptual layer), lớp mạng (network layer) và lớp xử lý dữ liệu (data processing layer) cần được xem xét để tăng cường bảo mật cũng như đảm bảo tính toàn vẹn của dữ liệu. Tại lớp nhận thức, các giao thức truyền thông đơn giản như UART (Universal Asynchronous Receiver-Transmitter) thường được sử dụng do tính nhẹ và phù hợp với phần lớn các thiết bị tài nguyên thấp. Tuy nhiên, việc truyền dữ liệu thô lại tiềm ẩn nguy cơ mất mát gói tin và tiếp nhận dữ liệu không được xác thực từ các nguồn không tin cậy [4]. Bên cạnh đó, trong quá trình truyền dữ liệu giữa gateway và máy chủ, việc mã hóa dữ liệu là bắt buộc; đồng thời, các khóa bí mật sử dụng trong quá trình mã hóa và giải mã cần được thay đổi định kỳ nhằm giảm thiểu nguy cơ bị tấn công dò khóa. Thực tế, nhiều thiết bị IoT cấp thấp hiện nay gặp vấn đề trong việc quản lý khóa bí mật, thường sử dụng các khóa tĩnh được cài đặt sẵn hoặc dựa vào bộ đếm để tạo khóa, dẫn đến nguy

cơ bị phá vỡ bảo mật do các phương pháp tấn công tinh vi [29].

Do đó, việc nghiên cứu và phát triển các giải pháp bảo mật nhẹ, phù hợp với tài nguyên hạn chế của thiết bị IoT, đồng thời đảm bảo các yếu tố như dữ liệu phải được mã hóa, khả năng kháng lại các cuộc tấn công, đảm bảo tính toàn vẹn của dữ liệu, các lớp trong kiến trúc IoT đều phải có cơ chế bảo mật hợp lý là một yêu cầu cấp thiết để nâng cao độ an toàn và hiệu quả của các hệ thống IoT trong thực tiễn.

1.2 Mục tiêu của đề tài

Các thiết bị IoT tài nguyên thấp thường dễ bị khai thác do thiếu các cơ chế bảo mật mạnh mẽ, chẳng hạn như mã hóa phức tạp hoặc xác thực thời gian thực cũng như một số nhược điểm khác được đề cập ở phần 1.1. Hơn nữa, phần lớn các nghiên cứu trong lĩnh vực bảo mật IoT chỉ tập trung giải quyết các bài toán đơn lẻ hoặc dừng lại ở mức lý thuyết. Vì vậy, đề tài hướng đến việc thiết kế một kiến trúc IoT hoàn chỉnh, có khả năng triển khai trên các thiết bị phần cứng khác nhau, đặc biệt là những thiết bị có tài nguyên giới hạn. Kiến trúc phải đảm bảo khả năng bảo mật đa lớp trong kiến trúc IoT bốn lớp. Từ đó, đề tài hướng đến việc đóng góp cho cộng đồng một kiến trúc có tính tổng quát cao, có thể triển khai trên bất kỳ nền tảng phần cứng nào.

1.3 Giải pháp đề xuất

Đề tài tập trung vào phát triển một kiến trúc IoT bảo mật toàn diện, vừa đảm bảo các yếu tố an toàn như tính toàn vẹn, bí mật dữ liệu, và khả năng chống lại các cuộc tấn công từ bên ngoài, vừa tối ưu để phù hợp với các thiết bị có tài nguyên hạn chế. Kiến trúc đề xuất sẽ tích hợp các giải pháp như sử dụng thuật toán mã hóa nhẹ, cơ chế trao đổi khóa động và kiến trúc khung truyền riêng đảm bảo chỉ có các thiết bị được ủy quyền mới có thể xử lý và giải mã các gói tin hợp lý. Đề tài cũng giới thiệu phương pháp *safe counter* mới nhằm hạn chế khả năng bị brute-force vào các gói tin. Qua đó, hệ thống vừa tăng cường bảo mật truyền thông giữa các lớp vừa duy trì hiệu suất và tính khả thi, đáp ứng nhu cầu thực tiễn trong các lĩnh vực y tế, giao thông và nhà thông minh.

1.4 Đóng góp của đề tài

Kết quả của khóa luận là những đóng góp trong việc thiết kế một kiến trúc IoT bảo mật, tối ưu cho các thiết bị tài nguyên thấp. Trong truyền dữ liệu, thuật toán mã hóa Ascon-128a đã được áp dụng nhằm đảm bảo tính bảo mật trong hệ thống, giảm đáng kể độ trễ mã hóa so với AES. Các gói tin được định dạng trong cấu trúc khung truyền tùy chỉnh, tích hợp các trường như header đồng bộ, định danh, số thứ tự, và mã xác

thực,... giúp xác minh tính toàn vẹn và ngăn chặn một số cuộc tấn công. Trao đổi khóa Elliptic Curve Diffie-Hellman (ECDH) sử dụng đường cong *sect163k1* kết hợp với kỹ thuật tạo khóa phiên dùng hàm băm Ascon Hash được triển khai hiệu quả, cho phép tạo khóa bí mật động giữa gateway (ESP32) và máy chủ, tăng cường bảo mật trước các tấn công Man-in-the-Middle. Các kết quả thử nghiệm sử dụng vi điều khiển tầm trung như STM32F411VET6 và ESP32 cho thấy tỷ lệ truyền gói tin đạt 99.99% trong 120 phút (147,682 gói tin), độ trễ trung bình 51.08 ms, và khả năng từ chối 100% 70,000 gói tin phát lại, khẳng định tính ổn định và an toàn của khung đề xuất.

1.5 Bố cục của khóa luận

Báo cáo khóa luận được chia thành 5 chương gồm các nội dung như sau:

- CHƯƠNG 1: GIỚI THIỆU
- CHƯƠNG 2: CÁC CÔNG TRÌNH LIÊN QUAN
- CHƯƠNG 3: PHƯƠNG PHÁP ĐỀ XUẤT
- CHƯƠNG 4: KẾT QUẢ TRIỂN KHAI VÀ PHÂN TÍCH
- CHƯƠNG 5: KẾT LUẬN

Chương 1 sẽ đưa ra các nội dung tổng quát của khóa luận. Các nội dung chính bao gồm đặt vấn đề, mục tiêu của đề tài, đưa ra giải pháp, đóng góp của đề tài, và cuối cùng là bố cục của khóa luận.

Chương 2 sẽ đưa ra một số công trình liên quan đến đề tài, làm rõ các ưu nhược điểm của từng công trình. Ngoài ra, chương này cũng sẽ phân tích một số cuộc tấn công phổ biến trong hệ thống IoT. Từ đó, đưa ra khoảng trống nghiên cứu, các vấn đề mà đề tài tập trung giải quyết.

Chương 3 bao gồm hai phần chính. Phần thứ nhất tập trung về cơ sở lý thuyết liên quan tới kiến trúc IoT đề xuất. Phần thứ hai sẽ trình bày chi tiết quá trình triển khai từng module trong hệ thống.

Chương 4 đưa ra các kết quả đạt được trong việc triển khai. Đồng thời, chương này sẽ đánh giá hiệu suất và một số thông số quan trọng khác cũng như phân tích và nhận xét về khả năng bảo mật của hệ thống.

Cuối cùng là chương 5 sẽ tóm gọn lại về mục tiêu, phương pháp, kết quả đạt được. Đồng thời sẽ đưa ra các hạn chế của đề tài và đề xuất hướng phát triển trong tương lai.

Chương 2: Các công trình liên quan

Với sự phát triển nhanh của IoT hiện nay, những vấn đề về bảo mật, đặc biệt là trong thiết kế các kiến trúc đảm bảo tính an toàn, toàn vẹn, và quyền riêng tư của dữ liệu đang được rất quan tâm, đặt ra những bài toán cần phải giải quyết. Nghiên cứu về kiến trúc IoT không chỉ tập trung vào các giải pháp kỹ thuật như mã hóa hoặc xác thực, ngoài ra cần phải xem các thách thức thực tiễn như hạn chế tài nguyên và khả năng tương thích của thiết bị. Trong phần này, các công trình nghiên cứu liên quan sẽ được trình bày, bao gồm các mô hình, kiến trúc bảo mật IoT và các loại tấn công thường gặp, nhằm xác định những tiến bộ hiện tại, các hạn chế còn tồn tại, cũng như khoảng trống nghiên cứu mà bài luận này hướng đến.

2.1 Các công trình nghiên cứu về kiến trúc IoT bảo mật

Trước hết, bài báo trong [27] đã thực hiện một đánh giá toàn diện về các mô hình và kiến trúc bảo mật cho IoT. Bài báo tập trung phân tích các nghiên cứu liên quan đến bảo mật IoT trong giai đoạn từ năm 2010 đến năm 2020. Tác giả nhấn mạnh rằng lĩnh vực nghiên cứu bảo mật cho IoT vẫn còn khá mới mẻ. Các đóng góp chính trong các nghiên cứu này chủ yếu tập trung vào kiến trúc (~81%) hoặc các mẫu (~19%). Phần lớn các nghiên cứu đề xuất các giải pháp cụ thể, trong đó khoảng 57% đã tiến hành kiểm thử và đánh giá ý tưởng của mình. Có khoảng 23% các nghiên cứu đưa ra một số use case để minh họa cho ý tưởng, và chỉ khoảng 20% bài báo triển khai thực tế một hệ thống hoàn chỉnh và thực hiện đánh giá dựa trên hệ thống đó. Ngoài ra, bài báo [27] cũng chỉ ra các khía cạnh bảo mật được các nghiên cứu quan tâm, bao gồm: tính bảo mật (~16%), tính toàn vẹn (~19%), tính khả dụng (~8%), tính xác thực (~25%), ủy quyền (~17%) và quyền riêng tư (~15%). Từ nghiên cứu trên cho thấy sự thiếu hụt trong triển khai một hệ thống thực tế. Ngoài ra, có rất nhiều khía cạnh về bảo mật cần phải quan tâm chứ không chỉ tập trung vào mỗi mã hóa dữ liệu khi triển khai một hệ thống IoT bảo mật.

Một nghiên cứu từ bài báo [8] cung cấp cái nhìn chi tiết hơn về các vấn đề bảo mật trong IoT. Bài báo giúp định nghĩa, phân loại và thảo luận về một số các thách thức về bảo mật IoT, đồng thời chỉ ra một số công trình liên quan để giải quyết các vấn đề nêu trên. Trong đó, các vấn đề chính mà bài báo đề cập bao gồm tính riêng tư trong IoT, mã hóa nhẹ, định tuyến an toàn, quản lý độ bền và khả năng phục hồi, cũng như phát hiện các cuộc tấn công từ chối dịch vụ (DoS) và tấn công nội bộ. Đặc biệt, về tính riêng tư, bài báo nhấn mạnh các thách thức như định vị trái phép, và truyền dữ liệu an toàn qua các phương tiện công cộng, đồng thời chỉ ra rằng chưa có khung công tác bảo vệ quyền riêng tư toàn diện cho nhiều ứng dụng IoT. Ngoài ra, bài báo đề xuất các giải pháp như

mã hóa nhẹ và sử dụng mạng định nghĩa bởi phần mềm (SDN) để tăng cường bảo mật, nhưng việc triển khai trên các thiết bị tài nguyên thấp vẫn là một hạn chế.

Nghiên cứu của [10] liên quan đến bảo vệ quyền riêng tư trong IoT, đề xuất hai giao thức bảo mật cho hệ thống chăm sóc sức khỏe dựa trên IoT, bao gồm một lược đồ xác thực dựa trên đăng nhập một lần (SSO) và một cơ chế chứng minh sự cùng tồn tại của các đối tượng gắn thẻ RF. Lược đồ SSO sử dụng hàm băm một chiều và số ngẫu nhiên để đảm bảo xác thực an toàn và truyền dữ liệu trên thiết bị tài nguyên thấp, trong khi cơ chế cùng tồn tại tăng cường an toàn bệnh nhân thông qua quản lý thuốc. Các phân tích bảo mật cho thấy các giao thức này chống lại các cuộc tấn công như giả mạo và phát lại một cách hiệu quả. Tuy nhiên, nghiên cứu chỉ mới tập trung hai giao thức riêng lẻ mà chưa tổng thể tích hợp các lớp khác nhau của hệ thống IoT, điều này hạn chế khả năng áp dụng vào các hệ thống IoT phức tạp. Ngoài ra, các giao thức chỉ mới được đánh giá về mặt lý thuyết, nhưng không có dữ liệu về hiệu suất thực tế như thời gian xử lý, tiêu thụ năng lượng, hoặc tỷ lệ truyền thành công.

Một nghiên cứu khác là [9] triển khai hệ thống IoT bảo mật liên quan đến chăm sóc sức khỏe. Nghiên cứu này tập trung vào quá trình tạo sinh khóa giữa máy chủ và thiết bị dựa trên dấu thời gian (Timestamp-based Secret Key Generation T-SKG) kết hợp với thuật toán DES với các chế độ ECB, CBC và CTR. Kết quả cho thấy tỷ lệ truyền gói tin khá cao từ 99.92% đến 99.97%. Mặc dù nghiên cứu đã triển khai một hệ thống thực tế và có các kết quả thử nghiệm, nghiên cứu này chỉ mới tập trung chủ yếu vào quá trình tạo sinh khóa chứ chưa tập trung vào phát triển một kiến trúc cụ thể. Ngoài ra, các thao tác như xác thực dữ liệu hay khả năng chống lại các cuộc tấn công vẫn chưa được phân tích một cách rõ ràng.

Nghiên cứu của [26] đề xuất khung bảo mật BlinkToSCoAP, tích hợp các giao thức nhẹ DTLS (Datagram Transport Layer Security), CoAP (Constrained Application Protocol), và 6LoWPAN để cung cấp bảo mật đầu cuối (end-to-end) cho các ứng dụng IoT, đặc biệt trên thiết bị hạn chế tài nguyên như cảm biến trong chăm sóc sức khỏe di động, tự động hóa tòa nhà, và lưới điện thông minh. Trong bài nghiên cứu này, nhóm tác giả trình bày các nghiên cứu lý thuyết cũng như triển khai thành công hệ thống. Các kết quả được trình bày trong bài báo cho thấy quá trình bắt tay DTLS có độ trễ trung bình là 880 ms (client) và 800 ms (server), với độ trễ lớn như vậy sẽ không phù hợp cho các ứng dụng IoT y tế yêu cầu phản hồi thời gian thực. Ngoài ra, mỗi gói tin truyền đi sẽ có thêm chi phí mỗi khung khi sử dụng DTLS và CoAP. Một nhược điểm nữa của hệ thống này đó là khóa bí mật chỉ được thiết lập một lần và sử dụng cho đến khi hết hạn, không có các cơ chế để làm mới khóa mỗi phiên mà không cần bắt tay lại, điều này có thể tăng nguy cơ bảo mật nếu khóa phiên bị xâm phạm trong thời gian dài.

Bên cạnh các nghiên cứu đã đề cập ở trên, nhiều công trình khác cũng tập trung vào

bảo mật IoT, đóng góp đáng kể vào việc phát triển các kiến trúc và giải pháp bảo mật cho IoT. Tuy nhiên, các hạn chế vẫn còn tồn tại. Phần lớn các nghiên cứu chỉ dừng ở mức lý thuyết, đề xuất các khung kiến trúc dựa trên cơ sở toán học mà thiếu triển khai thực tế. Một số nghiên cứu có thử nghiệm thực nghiệm, nhưng thường chỉ giải quyết một khía cạnh bảo mật cụ thể, như xác thực hoặc mã hóa, và hiếm có công trình nào đưa ra kiến trúc tổng quan tích hợp đa lớp.

2.2 Các mối đe dọa và loại tấn công phổ biến

Ở phần này sẽ liệt kê một số cuộc tấn công phổ biến trong hệ thống IoT, nhằm đưa ra cái nhìn toàn diện hơn về các mối đe dọa bảo mật mà hệ thống có thể phải đối mặt trong thực tế.

2.2.1 Dò khóa bí mật của hệ thống

Trong nhiều hệ thống IoT, việc áp dụng các thuật toán mã hóa đối xứng là cần thiết nhằm đảm bảo tính bí mật của dữ liệu trong suốt quá trình truyền tải. Tuy nhiên, nếu khóa bí mật của hệ thống bị lộ hoặc bị khai thác, kẻ tấn công có thể dễ dàng giải mã và truy cập vào nội dung dữ liệu. Do đó, trong phần này sẽ trình bày một số phương pháp tấn công phổ biến nhắm vào khóa bí mật, qua đó làm rõ các nguy cơ tiềm ẩn ảnh hưởng đến tính bảo mật của hệ thống.

2.2.1.1 Tấn công brute-force

Tấn công brute-force là phương pháp đơn giản nhất để tìm khóa bí mật trong mã hóa đối xứng, trong đó kẻ tấn công thử tất cả các tổ hợp khóa có thể cho đến khi tìm ra khóa đúng. Với một khóa có độ dài 40 bit, số tổ hợp khóa cần thử là $2^{40} \approx 1.1 \times 10^{12}$. Tùy thuộc vào thuật toán mã hóa và phần cứng sử dụng, tốc độ thử khóa của các máy tính thông thường hiện nay dao động từ 10^4 đến 10^9 khóa mỗi giây [25].

Khi sử dụng các thiết bị phần cứng chuyên dụng như GPU (Graphics Processing Unit) cao cấp, tốc độ brute-force có thể tăng đáng kể, đạt hàng tỷ khóa mỗi giây. Một ví dụ điển hình là hệ thống do chuyên gia bảo mật Jeremi Gosney xây dựng, sử dụng 25 GPU AMD Radeon hoạt động song song thông qua nền tảng Virtual OpenCL. Hệ thống này đạt tốc độ thử nghiệm lên tới 3.48×10^{11} mật khẩu NTLM mỗi giây, tương đương với khoảng 3.5×10^{11} khóa mã hóa mỗi giây [19]. Tương tự, dự án RC5 do Distributed.net tổ chức đã sử dụng hệ thống tính toán phân tán để giải mã dữ liệu được mã hóa bằng khóa 72 bit, đạt tốc độ brute-force lên tới 8×10^{11} khóa mỗi giây [21]. Những ví dụ này cho thấy sự tiến bộ đáng kể trong công nghệ brute-force, với tốc độ thử khóa ngày càng

tăng nhờ vào các hệ thống phần cứng mạnh mẽ và tính toán song song.

Trong bối cảnh các hệ thống IoT, nếu thiết bị sử dụng khóa ngắn (ví dụ: 40 bit) hoặc các thuật toán mã hóa yếu như DES, khả năng bị brute-force thành công là rất cao. Tuy nhiên, điểm yếu chính của phương pháp brute-force là thời gian thực hiện. Đối với các khóa có độ dài từ 128 bit trở lên, chẳng hạn như trong thuật toán AES-128, số tổ hợp khóa cần thử là $2^{128} \approx 3.4 \times 10^{38}$, khiến việc brute-force trở nên bất khả thi với công nghệ hiện tại.

2.2.1.2 Tấn công Side-Channel

Tấn công Side-Channel (kênh bên) khai thác các đặc điểm vật lý hoặc hành vi của thiết bị trong quá trình thực hiện mã hóa để suy ra khóa bí mật, thay vì phân tích cấu trúc toán học của thuật toán [24]. Nguyên lý cơ bản là các hiệu ứng vật lý gây ra bởi hoạt động của một hệ thống mã hóa có thể cung cấp thông tin bổ sung hữu ích về các bí mật trong hệ thống, chẳng hạn như khóa mã hóa, thông tin trạng thái một phần, văn bản gốc hoàn chỉnh hoặc một phần, và những thông tin tương tự [24].

Báo cáo từ Phòng thí nghiệm Hệ thống Nhúng, Viện Khoa học Ấn Độ, tập trung vào việc tấn công kênh bên công suất nhằm vào triển khai phần mềm của chuẩn mã hóa nâng cao (AES) trên vi điều khiển [20] cho thấy kỹ thuật CPA có thể phá vỡ bảo mật của AES triển khai trên vi điều khiển bằng cách tiết lộ khóa bí mật. Từ đó có thể thấy ngay cả AES mạnh mẽ cũng có thể bị tấn công nếu triển khai không an toàn, đặc biệt là trên các thiết bị IoT công suất thấp.

2.2.1.3 Một số cuộc tấn công dò khóa khác

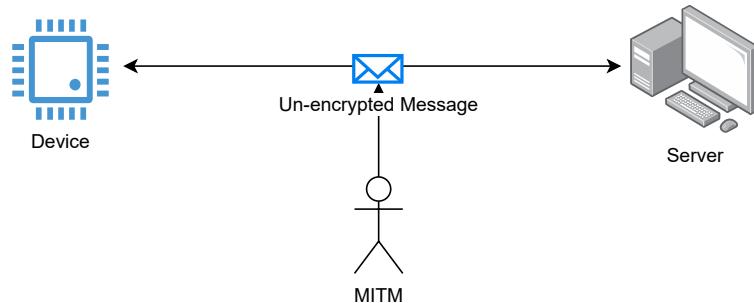
Các phương pháp cryptanalytic khai thác cấu trúc toán học của thuật toán mã hóa để tìm ra khóa bí mật hoặc giảm không gian khóa cần thử bao gồm một số phương pháp phổ biến như phân tích vi sai, phân tích tuyến tính, tấn công dựa trên plaintext đã biết (Known-Plaintext Attack - KPA) [13]. Các phương pháp này hiệu quả đối với các thuật toán mã hóa yếu hoặc các phiên bản giảm số vòng của các thuật toán như DES.

Tấn công related-key khai thác việc sử dụng các khóa có mối quan hệ rõ ràng (ví dụ: các khóa chỉ khác nhau một bit) để suy ra thông tin về khóa chính [22]. Phương pháp này chỉ hiệu quả nếu hệ thống sử dụng nhiều khóa liên quan, trong các hệ thống IoT sử dụng các phương pháp tạo khóa đơn giản (như bộ đếm), tấn công related-key có thể trở nên khả thi hoặc một số thiết bị IoT có thể sử dụng các khóa liên quan để mã hóa dữ liệu giữa các thiết bị hoặc các phiên giao tiếp, tạo cơ hội cho tấn công related-key nếu triển khai không an toàn.

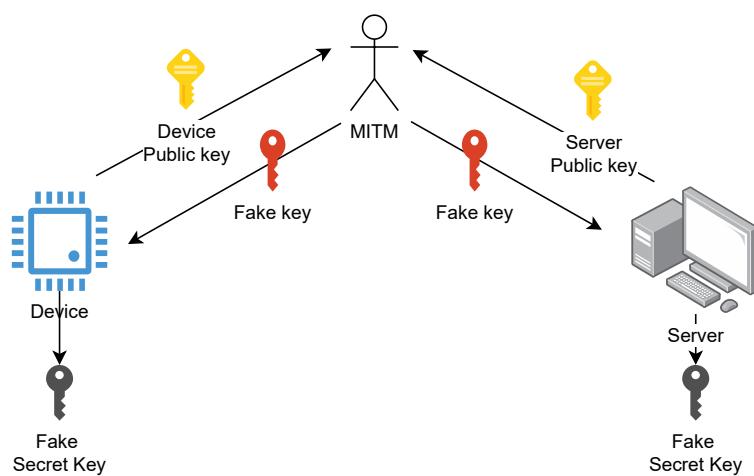
2.2.2 Tấn công Man-in-the-Middle

Tấn công Man-in-the-Middle (MITM) là một dạng tấn công bảo mật trong đó kẻ tấn công chèn mình vào giữa hai bên giao tiếp, chẳng hạn như thiết bị IoT và máy chủ, để nghe lén, chỉnh sửa hoặc giả mạo dữ liệu mà không bị phát hiện [17]. Từ đó, kẻ tấn công có thể bí mật thu thập dữ liệu truyền giữa các thiết bị, như khóa mã hóa hoặc thông tin nhạy cảm.

Trong các hệ thống IoT không áp dụng mã hóa dữ liệu trước khi truyền, nguy cơ bị kẻ tấn công chặn và khai thác thông tin trên đường truyền là rất cao (Hình 2.1). Tấn công Man-in-the-Middle (MITM) đặc biệt nguy hiểm khi kẻ tấn công can thiệp vào quá trình trao đổi khóa giữa thiết bị IoT và máy chủ (Hình 2.2) [18]. Trong một số hệ thống IoT, khóa công khai được truyền trực tiếp qua kênh không an toàn. Nếu kẻ tấn công chặn được luồng giao tiếp và thay thế bằng khóa giả mạo, khóa bí mật có thể bị suy ra, dẫn đến việc hệ thống mất an toàn hoàn toàn.



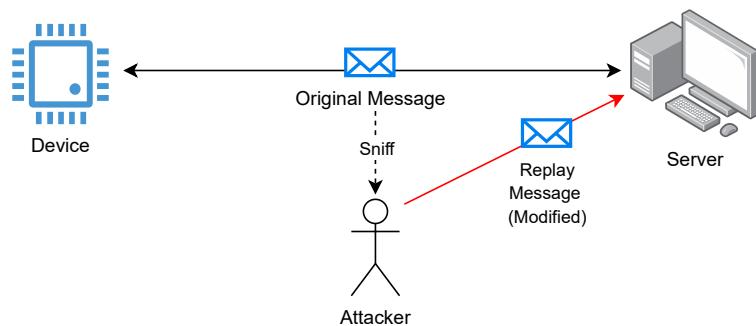
Hình 2.1: Tấn công MITM trong nghe lén dữ liệu



Hình 2.2: Tấn công MITM trong trao đổi khóa

2.2.3 Tấn công phát lại (Replay Attack)

Tấn công phát lại (Replay Attack) là một dạng tấn công bảo mật trong đó kẻ tấn công chặn và ghi lại các gói tin hợp lệ từ quá trình giao tiếp giữa thiết bị IoT và máy chủ, sau đó gửi lại các gói tin này để lừa hệ thống thực hiện hành động không mong muốn, như cấp quyền truy cập trái phép hoặc thực thi lệnh giả mạo [23]. Trong các hệ thống IoT, kẻ tấn công có thể ghi lại dữ liệu hợp lệ, chỉnh sửa nội dung, và gửi lại máy chủ, khiến máy chủ xử lý như gói tin hợp lệ, dẫn đến sai lệch thông tin (Hình 2.3). Trong các hệ thống IoT, đặc biệt là các ứng dụng thời gian thực như y tế hoặc giao thông thông minh, tấn công này đặc biệt nguy hiểm do dữ liệu sai lệch có thể gây hậu quả nghiêm trọng, ảnh hưởng trực tiếp đến an toàn và quyền lợi của người dùng.



Hình 2.3: Minh họa cuộc tấn công phát lại

2.3 Khoảng trống nghiên cứu

Dựa trên việc phân tích các ưu và nhược điểm của những nghiên cứu trước cũng như sự đa dạng và phức tạp của các hình thức tấn công vào hệ thống IoT, việc nghiên cứu và triển khai một kiến trúc bảo mật toàn diện là cần thiết và cũng là mục tiêu mà đề tài hướng đến. Bằng cách kết hợp các thuật toán mã hóa nhẹ, kiến trúc khung truyền độc lập kèm xác thực dữ liệu cùng với các phương pháp trao đổi khóa hiệu quả, đề tài hy vọng sẽ đóng góp một giải pháp có tính ứng dụng cao, đồng thời nâng cao tính bảo mật và khả năng mở rộng của hệ thống IoT trong thực tiễn.

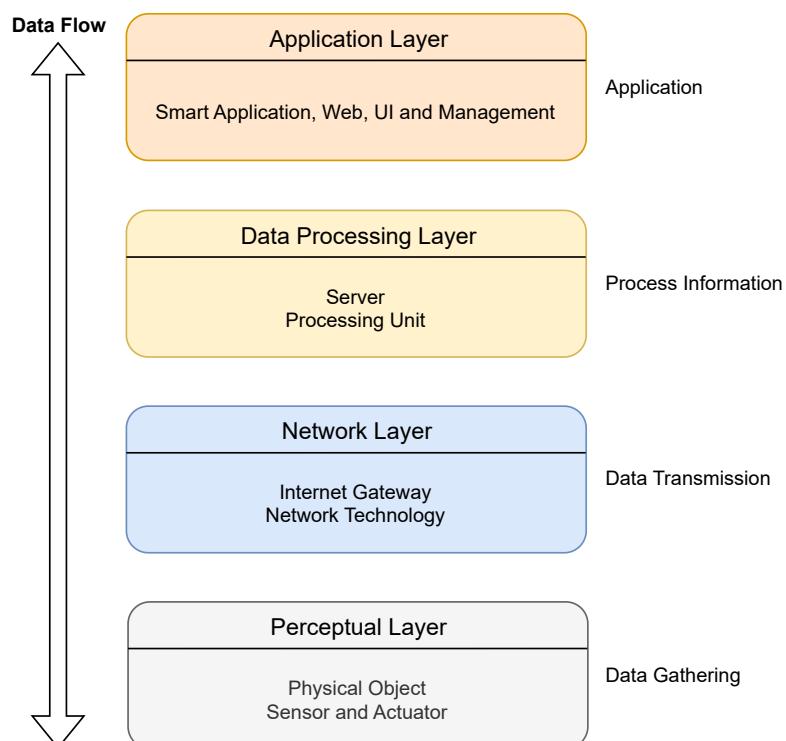
Chương 3: Phương pháp đề xuất

3.1 Cơ sở lý thuyết

Phần này trình bày chi tiết về cơ sở lý thuyết, các khái niệm cần thiết liên quan đến đề tài nghiên cứu.

3.1.1 Kiến trúc IoT bốn lớp

Một kiến trúc IoT điển hình bao gồm bốn thành phần chính: phần cứng được kết nối có nhiệm vụ thu thập và xử lý dữ liệu thông qua cảm biến hoặc bộ điều khiển tại thiết bị biên; kết nối truyền thông (có thể gồm nhiều giao thức), đảm bảo việc truyền nhận dữ liệu giữa thiết bị và nền tảng đám mây; hệ thống xử lý dữ liệu – thường tích hợp trong nền tảng điện toán đám mây; và cuối cùng là ứng dụng trực quan hóa dữ liệu thu thập.



Hình 3.1: Mô hình IoT 4 lớp

3.1.1.1 Lớp nhận thức (Perceptual Layer)

Đây là lớp đầu tiên của hệ thống, có nhiệm vụ thu thập dữ liệu từ nhiều nguồn khác nhau. Lớp này bao gồm các cảm biến được đặt trong môi trường nhằm thu thập dữ liệu

về nhiệt độ, độ ẩm, ánh sáng và các dữ liệu vật lý khác. Các chuẩn giao tiếp có dây hoặc không dây thường được sử dụng để kết nối với lớp gateway ở trên.

3.1.1.2 Lớp mạng (Network Layer)

Lớp mạng thường được sử dụng như một gateway nhằm đảm bảo kết nối và truyền thông giữa các thiết bị IoT cũng như với mạng internet. Các công nghệ như Wi-Fi, Bluetooth, Zigbee, hoặc mạng di động (4G, 5G) được sử dụng, cùng với các cổng (gateway) như ESP32 sẽ được dùng để chuyển tiếp dữ liệu. Lớp này thường sử dụng các cơ chế mã hóa dữ liệu để đảm bảo an toàn trong quá trình truyền dữ liệu.

3.1.1.3 Lớp xử lý dữ liệu (Data Processing Layer)

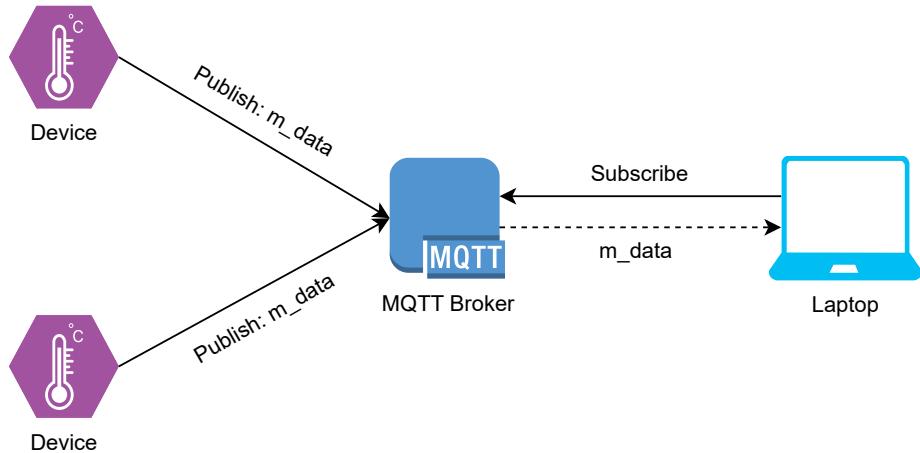
Lớp xử lý dữ liệu thường là các máy chủ (server) trung gian có vai trò thu nhận, phân tích, và xử lý dữ liệu thô từ các thiết bị IoT. Các công cụ như hệ quản trị dữ liệu, nền tảng phân tích, hoặc thuật toán học máy được sử dụng để trích xuất thông tin giá trị. Lớp này cũng có vai trò xác định dữ liệu nhận từ các thiết bị có chính xác hay không, từ đó quyết định xem nên lưu trữ dữ liệu hay từ chối nhận dữ liệu từ bên ngoài.

3.1.1.4 Lớp ứng dụng (Application Layer)

Đây là lớp cao nhất, cung cấp giao diện thân thiện cho người dùng để tương tác với hệ thống IoT. Bao gồm các ứng dụng di động, web, và phần mềm trung gian (middleware) cho phép tích hợp và chia sẻ dữ liệu giữa các thiết bị.

3.1.2 Giao thức MQTT

Trong hệ thống IoT được xây dựng, giao thức được sử dụng để giao tiếp giữa gateway và máy chủ là giao thức MQTT (Message Queuing Telemetry Transport). Đây là một giao thức nhắn tin nhẹ, tối ưu cho các ứng dụng IoT, nơi yêu cầu truyền dữ liệu nhanh, gọn và tiêu tốn ít tài nguyên. Trong kiến trúc MQTT, Broker đóng vai trò là trung tâm phân phối thông điệp giữa các thiết bị, giúp chúng gửi và nhận thông điệp thông qua mô hình publish/subscribe. Khi một thiết bị publish một thông điệp đến một chủ đề (topic), Broker sẽ tiếp nhận và chuyển thông điệp đó đến mọi thiết bị đã subscribe topic tương ứng. Ngoài việc chuyển tiếp dữ liệu, Broker còn chịu trách nhiệm quản lý kết nối của các thiết bị, xử lý cơ chế giữ kết nối, và duy trì trạng thái phiên làm việc của client.



Hình 3.2: Giao thức MQTT

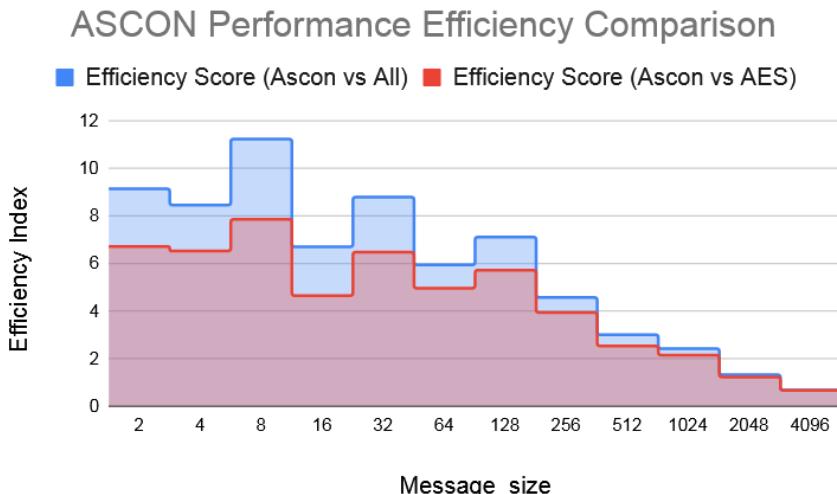
3.1.3 Mật mã hóa nhẹ ASCON-128a

Trong các hệ thống IoT hạn chế tài nguyên, việc sử dụng các thuật toán mã hóa truyền thống như AES hoặc RSA thường gặp nhiều bất cập do yêu cầu cao về năng lượng và khả năng tính toán. Điều này dẫn đến độ trễ lớn và tiêu tốn tài nguyên, ảnh hưởng đến hiệu quả hoạt động của các thiết bị nhúng. Trước thực trạng đó, mã hóa nhẹ (lightweight cryptography) đã được nghiên cứu và phát triển như một giải pháp tiềm năng, nhằm cung cấp mức độ bảo mật phù hợp với chi phí tính toán thấp.

Các công trình gần đây, chẳng hạn như bài báo [1] đã triển khai hệ thống IoT ứng dụng cho y tế đã so sánh 2 thuật toán mã hóa là Ascon-128 và AES-GCM-128 trên Raspberry Pi. Kết quả cho thấy Ascon-128 cho ra hiệu suất tốt hơn hoàn toàn so với AES-GCM-128 về mặt hiệu năng. Bài báo thực hiện so sánh tốc độ mã hóa của từng gói tin với kích thước khác nhau, Ascon cho ra hiệu suất cao với các gói tin có kích thước nhỏ, chẳng hạn với 8 byte dữ liệu, Ascon có hiệu suất gấp 8 lần so với AES và với 1024 byte, Ascon có hiệu suất gấp đôi. Với 2048 byte, Ascon vẫn cho ra hiệu suất tốt hơn dù mức độ chênh lệch không nhiều, hình 3.3 được trích ra từ bài báo thể hiện hiệu suất của Ascon-128 so với phương pháp AES.

Một nghiên cứu khác từ [5] trong bài giới thiệu Ascon v1.2 cũng nhấn mạnh hiệu suất cao của Ascon-128 khi thực hiện mã hóa dữ liệu có kích thước vừa và nhỏ so sánh với các thiết kế AES dùng native AES instruction trên Intel Skylake processors.

Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (NIST – National Institute of Standards and Technology) cũng đã thực hiện đánh giá một số thuật toán mã hóa nhẹ ở vòng chung kết. Kết quả cho thấy Ascon là thuật toán có hiệu suất tốt nhất so với các thuật toán mã hóa nhẹ khác như Elephant, Grain-128aead hoặc Sparkle. Ascon cũng đã được NIST lựa chọn làm tiêu chuẩn mã hóa nhẹ chính thức vào năm 2023 dựa vào độ an toàn cao và hiệu



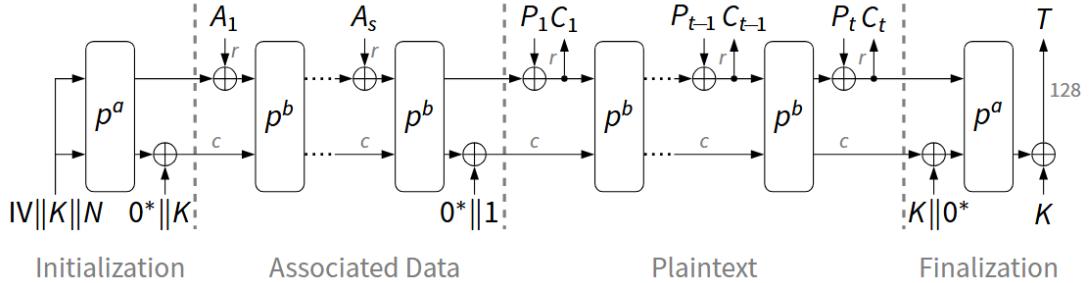
Hình 3.3: So sánh hiệu năng giữa Ascon và AES được trích từ bài báo

suất vượt trội trên cả phần mềm lẫn phần cứng trong môi trường IoT giới hạn tài nguyên. NIST thực hiện đánh giá trên các nền tảng như 8-bit AVR, ARM Cortex-M0+/M3/M4, RISC-V, Xtensa LX6, v.v. Thời gian thực thi của Ascon nhanh hơn đáng kể so với chuẩn AES-GCM trên nRF52840. Trong các thử nghiệm benchmark độc lập, Ascon là một trong số ít thuật toán đạt hiệu suất cao và footprint nhỏ nhất, đặc biệt trên thiết bị nhúng có tài nguyên hạn chế.

3.1.3.1 Thuật toán ASCON-128a

ASCON là một bộ thuật toán mã hóa nhẹ bao gồm tập hợp các thiết kế mã hóa xác thực và một họ các hàm băm cùng hàm sinh đầu ra mở rộng [?]. Ở đề tài này tập trung vào phần mã hóa xác thực trong triển khai hệ thống. Vì vậy, để đảm bảo tính ngắn gọn, chỉ có mã hóa xác thực được thảo luận trong phần này. ASCON sử dụng chế độ hoạt động dựa trên cấu trúc duplex, trong đó phép hoán vị 320-bit của ASCON được lặp lại thông qua một mạng hoán vị - thay thế (Substitution-Permutation Network - SPN) để thực hiện quá trình mã hóa hoặc giải mã dữ liệu. Quá trình này được thực hiện theo kiểu bit-slice, một đặc điểm giúp ASCON có khả năng mở rộng linh hoạt trên các nền tảng xử lý 8, 16, 32 và 64 bit, đồng thời vẫn đảm bảo tính "nhẹ" của thuật toán.

ASCON có hai biến thể chính phù hợp với độ dài khối dữ liệu cần mã hóa khác nhau: ASCON-128 và ASCON-128a. Trong các thuật toán mã hóa khối (block cipher), dữ liệu đầu vào được chia thành các khối có độ dài cố định. Trong đó, ASCON-128 sử dụng khối dữ liệu 64-bit và ASCON-128a sử dụng khối dữ liệu 128-bit. Các tập tham số tương ứng của từng biến thể ASCON được trình bày trong Bảng 3.1. Như thể hiện trong bảng, điểm khác biệt chính giữa các biến thể là kích thước khối dữ liệu và số vòng lặp trong quá trình hoán vị.



Hình 3.4: Quá trình mã hóa của ASCON với dữ liệu liên kết

Quy trình mã hóa của ASCON được mô tả trong Hình 3.4, gồm bốn giai đoạn chính:

1. Khởi tạo (Initialization)
2. Xử lý dữ liệu liên kết (Associated Data Processing)
3. Xử lý plaintext (Plaintext Processing)
4. Kết thúc và tạo thẻ xác thực (Finalization for Tag Generation)

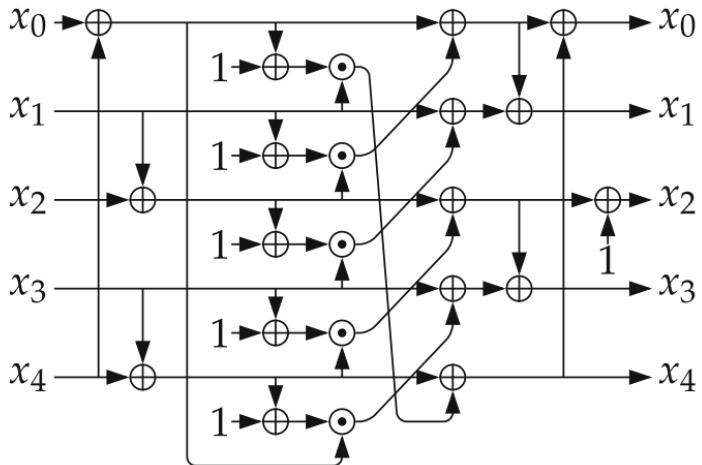
Trong suốt quá trình này, hai phép hoán vị 320-bit ký hiệu là p^a và p^b được sử dụng, trong đó a và b biểu thị số vòng lặp tương ứng. Các phép hoán vị này được biểu diễn theo dạng bit-slice, chia thành năm thanh ghi 64 bit, tạo thành trạng thái nội bộ gồm 5 word mỗi word có 64 bit.

Trong phiên bản ASCON đầy đủ với 12 vòng lặp, các phép hoán vị áp dụng lặp đi lặp lại quá trình biến đổi dựa trên mạng hoán vị - thay thế (SPN), bao gồm các bước: công hằng số vòng lặp (round constants), áp dụng lớp thay thế (substitution layer) và sử dụng lớp tuyến tính (linear layer) để khuếch tán dữ liệu trong trạng thái nội bộ.

Lớp thay thế (Substitution layer) sử dụng một hộp thay thế (S-box) gồm 5 bit có cấu trúc nhỏ gọn và tối ưu cho các hệ thống nhẹ, như minh họa trong Hình 3.5. S-box được áp dụng song song 64 lần, nhằm cập nhật từng bit-slice trong trạng thái nội bộ. S-box 5 bit được thiết kế dựa trên logic Boolean, cho phép triển khai hiệu quả trên cả hai nền tảng phần cứng ASIC và FPGA.

Lớp tuyến tính (Linear Layer) trong ASCON cập nhật mỗi word 64 bit trong trạng thái nội bộ bằng cách xoay (rotate) các thanh ghi với các giá trị dịch bit khác nhau, sau đó cộng theo modulo-2 (XOR) các giá trị đã được dịch.

Quá trình giải mã có một vài điểm khác biệt so với mã hóa. Thay vì xử lý plaintext, giải mã sử dụng hàm xử lý dữ liệu mã hóa (ciphertext processing), trong đó các khối plaintext được khôi phục bằng cách thực hiện phép XOR giữa khối plaintext và trạng thái hiện tại.



Hình 3.5: Khối 5-bit S-box của ASCON

Bảng 3.1: Bảng tham số cho ASCON

Variants	Bit size				Rounds	
	Key	Nonce	Tag	Data Block	p^a	p^b
ASCON-128	128	128	128	64	12	6
ASCON-128a	128	128	128	64	12	8

3.1.4 Thuật toán trao đổi khóa Elliptic Curve Diffie-Hellman

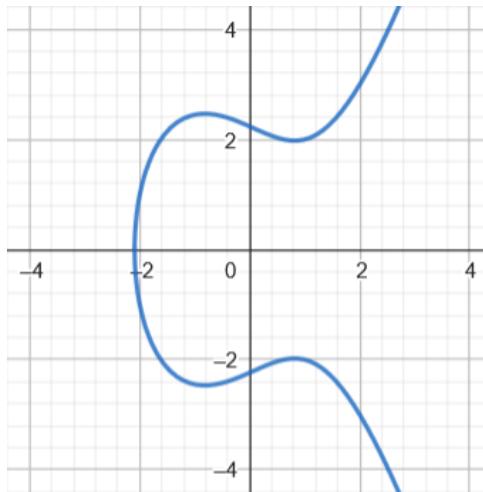
3.1.4.1 Đường cong Elliptic trên trường hữu hạn

Một đường cong elliptic là một đường cong trơn (không có điểm nhọn hay tự cắt) được xác định bởi phương trình dạng

$$y^2 = x^3 + ax + b \pmod{p}$$

trong đó a và b là các hằng số và p là một số nguyên tố đại diện cho trường hữu hạn (finite field). Đường cong này có thể được diễn giải theo hình học và mang cấu trúc nhóm, cụ thể là một nhóm cộng với phép cộng điểm trên đường cong. Hình 3.7 minh họa một đường cong elliptic với $a = -2$ và $b = 5$.

Phép cộng hai điểm P và Q trên đường cong sẽ tạo ra một điểm thứ ba $R = P + Q$. Cách thực hiện là vẽ đường thẳng đi qua hai điểm P và Q , tìm điểm giao thứ ba của đường thẳng đó với đường cong, sau đó phản xạ điểm này qua trực hoành (trục x) để thu được điểm R .



Hình 3.6: Đường cong elliptic $y^2 = x^3 - 2x + 5$

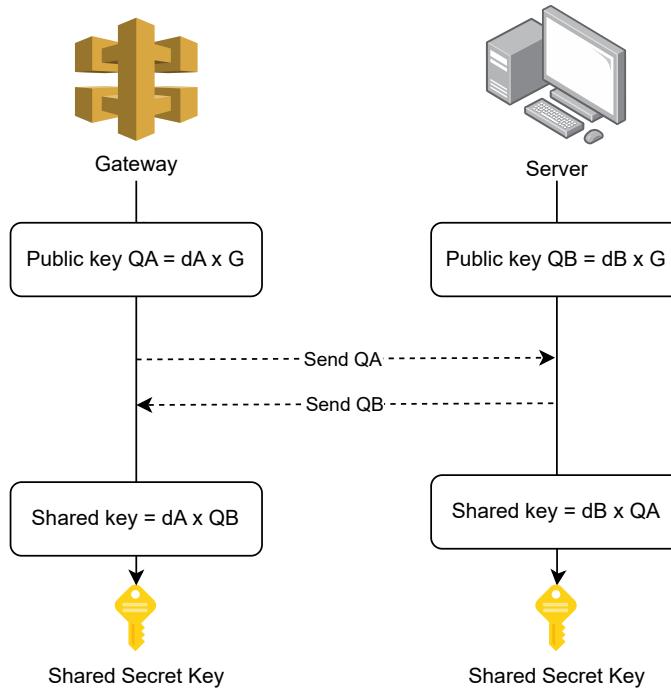
3.1.4.2 Trao đổi khóa Elliptic Curve Diffie-Hellman (ECDH)

Elliptic Curve Diffie-Hellman (ECDH) là một thuật toán trao đổi khóa cho phép hai bên thiết lập một khóa bí mật chung qua kênh truyền thông không an toàn, dựa trên độ phức tạp tính toán của bài toán logarit rời rạc trên đường cong elliptic. Với hiệu suất cao và mức độ an toàn mạnh mẽ, ECDH được ứng dụng rộng rãi trong các giao thức mật mã. Cho G là điểm sinh (generator point) trên đường cong với bậc n . Device và Gateway sẽ là hai thiết bị thực hiện trao đổi khóa với nhau, quá trình thực hiện ECDH được tóm tắt qua các bước sau:

1. Device chọn khóa riêng (private key) $d_A \in \mathbb{Z}_n$, và tính khóa công khai (public key) $Q_A = d_A \cdot G$.
2. Gateway thực hiện quá trình tương tự, chọn khóa riêng $d_B \in \mathbb{Z}_n$ và tính khóa công khai $Q_B = d_B \cdot G$.
3. Device và Gateway thực hiện trao đổi khóa công khai với nhau Q_A và Q_B .
4. Device tính khóa bí mật chung $S_A = d_A \cdot Q_B$.
5. Gateway tính khóa bí mật chung $S_B = d_B \cdot Q_A$.
6. Do tính chất giao hoán của phép nhân điểm trên đường cong elliptic, $S_A = S_B = d_A \cdot d_B \cdot G$, cả hai bên đạt được khóa bí mật chung S .

Để tài sử dụng đường cong *sect163k1* trên trường hữu hạn $GF(2^{163})$, có phương trình sau:

$$f(x) = x^{163} + x^7 + x^6 + x^3 + 1,$$



Hình 3.7: Quá trình thực hiện trao đổi khóa ECDH

được SECG (Standards for Efficient Cryptography Group) đề xuất nhờ khả năng tính toán hiệu quả trong họ đường cong Koblitz, giúp giảm chi phí xử lý tính toán. Với mức độ bảo mật tương đương 80-bit, đường cong này phù hợp cho các thiết bị IoT có tài nguyên hạn chế trong việc trao đổi khóa bảo mật [2]. Các tham số quan trọng bao gồm:

$$a = 1, \quad b = 1, \quad h = 2,$$

$$n = 04\ 00000000\ 00000000\ 00020108\ a2e0cc0d\ 99f8a5ef,$$

Với điểm cơ sở G :

$$G_x = 0000002\ fe13c053\ 7bbc11ac\ aa07d793$$

$$\text{de4e6d5e}\ 5c94eee8$$

$$G_y = 0000002\ 89070fb0\ 5d38ff58\ 321f2e80$$

$$0536d538\ ccdcaa3d9$$

ECDH đã được chứng minh có độ an toàn tương đương với thuật toán Diffie-Hellman truyền thống nhưng sử dụng các khóa có kích thước nhỏ hơn nhiều [?] được trình bày ở Bảng 3.2, điều này đặc biệt phù hợp với các thiết bị IoT có hạn chế về tài nguyên xử lý và bộ nhớ.

Bảng 3.2: Ratio of DH Security to EC Security at Different Security Levels

Security Level (bits)	Ratio (DH : EC)
80	3 : 1
112	6 : 1
128	10 : 1
192	32 : 1
256	64 : 1

Mặc dù ECDH đòi hỏi chi phí tính toán cao hơn so với một số phương pháp đơn giản khác như sử dụng timestamp, bộ đếm (counter) hoặc khóa tĩnh, việc lựa chọn đường cong elliptic phù hợp sẽ giúp cân bằng giữa hiệu suất và bảo mật, từ đó đảm bảo hệ thống duy trì được mức độ an toàn trong thời gian dài.

3.1.5 Hàm băm nhẹ Ascon

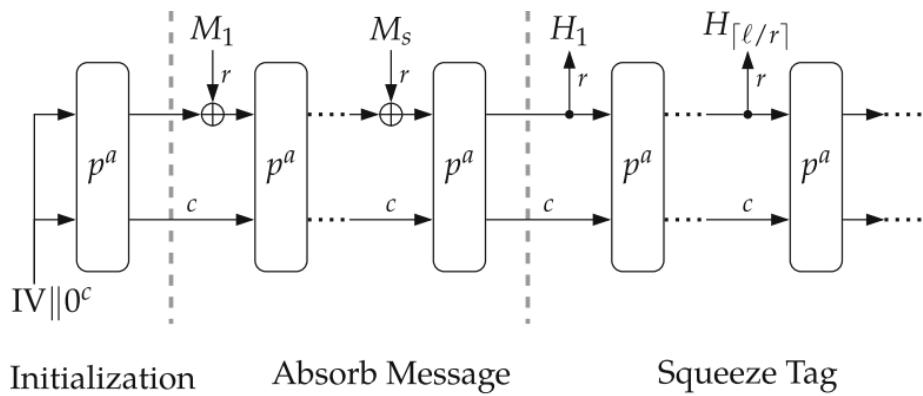
Ascon sử dụng cấu trúc sponge duplex để cung cấp cả chức năng băm mật mã và mã hóa xác thực. Cấu trúc sponge là một phương thức vận hành, ánh xạ dữ liệu đầu vào có độ dài bất kỳ sang đầu ra có độ dài tuỳ chỉnh, thông qua một hàm hoán vị cố định và quy tắc đệm (padding). Cấu trúc này hoạt động trên một trạng thái hữu hạn, lặp lại việc áp dụng hàm hoán vị nội bộ, tổng quát hóa cả mã dòng với đầu vào cố định và hàm băm với đầu ra cố định. Sponge duplex là một biến thể của sponge, cho phép xen kẽ các khối đầu vào và đầu ra với tốc độ tương đương [7].

Ascon-Hash là một hàm băm nhẹ dựa trên cấu trúc sponge, thực hiện hoán vị trên dữ liệu có kích thước bất kỳ để tạo ra giá trị băm 256-bit cố định, tối ưu cho các thiết bị nhúng. Sự tiêu chuẩn hóa của hàm băm Ascon trong NIST SP 800-232 và sự công nhận từ CAESAR làm tăng tính ứng dụng trong các hệ thống hạn chế tài nguyên, nơi hiệu suất và an ninh là ưu tiên hàng đầu. Theo kết quả đánh giá từ [12], Ascon-Hash nhẹ hơn và tối ưu hơn cho phần cứng, đặc biệt trên các thiết bị như Arduino Due với ARM Cortex M3 so với các hàm băm truyền thống như SHA-256. Hàm này sử dụng các thao tác đơn giản như XOR, dịch vòng (rotation), AND, NOT cùng với các hằng số để tối ưu hóa lớp thay thế (substitution layer), giúp tiết kiệm tài nguyên trên phần cứng [7]. Đặc tính nhẹ của Ascon-Hash cho phép triển khai hiệu quả trên các thiết bị IoT tài nguyên thấp, hỗ trợ nhiều bộ xử lý trong hệ thống với chi phí thấp. Dựa trên cấu trúc sponge, Ascon-Hash vận hành qua ba giai đoạn chính:

1. Khởi tạo (Initialization): Trạng thái khởi tạo 320-bit của Ascon-Hash được xác định bởi hằng số IV , trong đó chứa các thông số cấu hình của thuật toán gồm các giá trị như $k = 0$, tốc độ xử lý dữ liệu r , số vòng lặp a và b . Sau đó là độ dài đầu ra tối đa h bit, đối với Ascon-Hash giá trị h được khởi tạo là 0. Cuối cùng là một

dãy 256-bit toàn số 0.

2. Hấp thụ (Absorption): Dữ liệu đầu vào được chia thành các khối 64-bit, mỗi khối được XOR với trạng thái nội bộ 320-bit, sau đó áp dụng hàm hoán vị 12 vòng với các thao tác Addition, Rotation, XOR.
3. Nén (Squeezing): Sau khi hấp thụ, trạng thái được xử lý thêm 12 vòng để tạo giá trị băm cuối cùng.



Hình 3.8: Chế độ băm Ascon-Hash

Ở mỗi phiên, khóa bí mật được tạo ra sau khi thực hiện trao đổi khóa ECDH sẽ được đưa vào hàm Ascon-Hash nhằm tạo ra khóa phiên cho mã hóa và giải mã dữ liệu.

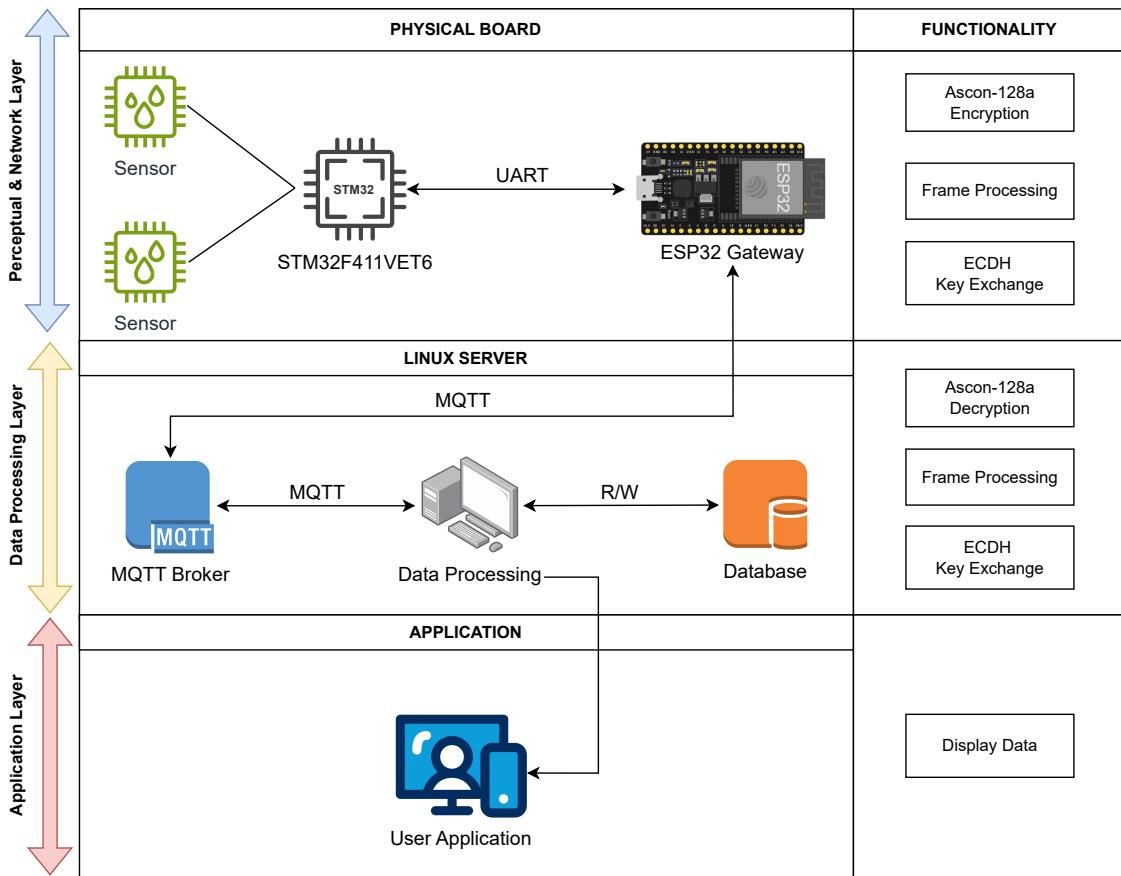
3.2 Phương pháp thực hiện

3.2.1 Tổng quan hệ thống IoT

Mô hình IoT được triển khai dựa trên kiến trúc IoT bốn lớp như trình bày ở phần 3.1.1. Ở lớp nhận thức, vi điều khiển STM32F411VET6 được sử dụng để thu thập dữ liệu từ cảm biến. Dữ liệu sau đó được truyền tới lớp xử lý dữ liệu thông qua ESP32, thiết bị đóng vai trò như một gateway trung gian. Giao tiếp giữa STM32 và ESP32 được thực hiện thông qua giao thức UART. Trên máy chủ Linux, các thành phần được triển khai bao gồm MQTT Broker và Backend chịu trách nhiệm tiếp nhận dữ liệu từ gateway, xử lý và phân giải các gói tin, kiểm tra tính hợp lệ của gói tin. Hình 3.9 trình bày mô hình tổng quan của hệ thống. Phần dưới đây sẽ trình bày từng thành phần trong hệ thống, các khái niệm về khung truyền được đề cập tới sẽ trình bày chi tiết ở các phần 3.2.2 và 3.2.4.

3.2.1.1 STM32F411VET6 MCU

Vi điều khiển STM32F411VET6 được phát triển dựa trên bộ công cụ STM32CubeIDE kết hợp với STM32CubeMX. Trong kiến trúc hệ thống đề xuất, STM32 được sử dụng



Hình 3.9: Mô hình tổng quan hệ thống

ở lớp nhận thức nhằm thu thập dữ liệu xung quanh thông qua các cảm biến được kết nối. Sau khi dữ liệu được thu thập, STM32 sẽ tiến hành mã hóa dữ liệu bằng thuật toán mã hóa nhẹ Ascon-128a. Tiếp theo, STM32 sẽ cấu trúc dữ liệu đã mã hóa thành khung truyền, và gửi đến lớp mạng để tiếp tục xử lý. Trước mỗi lần mã hóa, STM32 sẽ tạo các khóa phiên bằng cách sử dụng hàm băm Ascon Hash, đảm bảo mỗi khối dữ liệu sẽ sử dụng một khóa mã hóa khác nhau nhằm tăng cường tính bảo mật (kỹ thuật tạo khóa phiên sẽ được trình bày ở phần 3.2.3). Thứ tự khóa phiên sẽ được STM32 đồng bộ liên tục cùng với máy chủ để đảm bảo quá trình mã hóa - giải mã được thực hiện chính xác.

3.2.1.2 ESP32 Gateway

ESP32 đóng vai trò như một IoT gateway tại lớp mạng trong mô hình IoT bốn lớp. Đây là thành phần quan trọng và có mức độ phức tạp cao nhất trong quá trình triển khai hệ thống. ESP32 đóng vai trò trung gian, thực hiện chuyển tiếp dữ liệu giữa lớp nhận thức (các thiết bị cảm biến sử dụng vi điều khiển STM32) sử dụng giao thức UART và lớp xử lý dữ liệu (máy chủ) sử dụng giao thức MQTT. Mọi khung truyền được gửi đến gateway sẽ được phân giải và kiểm tra tính toàn vẹn trước khi chuyển tiếp giữa các lớp,

đảm bảo rằng chỉ dữ liệu hợp lệ mới được truyền đi. ESP32 là thiết bị thực hiện việc cấu trúc nhiều bộ khung truyền nhất trong hệ thống với tổng cộng năm bộ khung được định nghĩa và sử dụng tại đây. ESP32 cũng là nơi bắt đầu quá trình trao đổi khóa bí mật chung, khi khóa bí mật chung hết hạn hoặc có dấu hiệu bị lộ, ESP32 sẽ tiến hành yêu cầu quá trình trao đổi khóa bí mật chung với máy chủ. Đặc biệt, cơ chế *safe counter* (trình bày ở phần 3.2.5) cũng được triển khai trong ESP32, đây là một trong những cơ chế quan trọng giúp tăng tính bảo mật của hệ thống. Nhìn chung, đây là thành phần rất quan trọng, không chỉ đảm nhận chức năng giao tiếp mà còn thực hiện nhiều nhiệm vụ bảo mật quan trọng, góp phần đảm bảo sự ổn định và an toàn của toàn bộ kiến trúc IoT.

3.2.1.3 Hệ thống máy chủ

Máy chủ đóng vai trò trung tâm trong hệ thống IoT nằm ở lớp xử lý dữ liệu trong kiến trúc IoT bốn lớp, chịu trách nhiệm tiếp nhận dữ liệu từ gateway, xử lý và phân giải các gói tin, kiểm tra tính hợp lệ của gói tin, sau đó cung cấp thông tin đến người dùng cuối. Trong hệ thống này, thành phần máy chủ được triển khai trên máy tính sử dụng hệ điều hành Linux. Trên máy chủ bao gồm hai thành phần chính là hệ thống Backend và MQTT Broker.

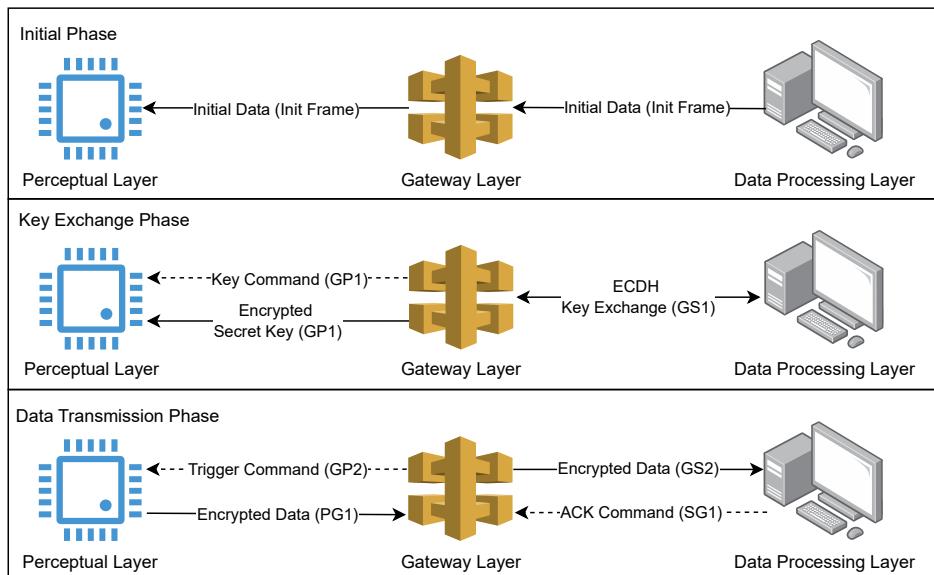
Đầu tiên là thành phần MQTT Broker, được triển khai trên máy chủ sử dụng Aedes. Đây là một MQTT Broker nhẹ được phát triển bằng ngôn ngữ JavaScript, phù hợp để tích hợp với hệ thống Node.js. Việc lựa chọn tự triển khai MQTT Broker trên máy chủ thay vì sử dụng các dịch vụ MQTT Broker public xuất phát từ hai lý do chính. Trước hết, độ trễ khi sử dụng Broker public thường cao hơn nhiều do lượng lớn người dùng chia sẻ cùng một hạ tầng dẫn đến tình trạng tắc nghẽn hoặc chậm phản hồi. Ngược lại, triển khai Broker riêng giúp tối ưu tốc độ giao tiếp nhờ kiểm soát hoàn toàn tài nguyên và môi trường mạng. Thứ hai, tự cấu hình Broker cho phép tích hợp thêm các tính năng bảo mật quan trọng như xác thực người dùng, phân quyền truy cập theo topic, và giới hạn quyền publish/subscribe chỉ dành cho các thiết bị đáng tin cậy. Điều này nâng cao tính bảo mật và khả năng kiểm soát trong hệ thống, đặc biệt quan trọng đối với các ứng dụng IoT yêu cầu tính riêng tư và độ tin cậy cao.

Hệ thống Backend chính là nơi tiếp nhận dữ liệu từ gateway và tiến hành xử lý dữ liệu. Khi nhận được khung truyền từ gateway, Backend sẽ tiến hành phân giải để kiểm tra. Đối với các gói tin không hợp lệ, Backend có khả năng từ chối ngay lập tức để không ảnh hưởng đến người dùng. Trong kiến trúc hệ thống này, Backend được cấu hình để thực hiện một số chức năng quan trọng liên quan đến bảo mật và xử lý dữ liệu. Thuật toán trao đổi khóa ECDH được sử dụng để thiết lập khóa chung bí mật giữa gateway, trong khi thuật toán Ascon-128a đảm nhiệm việc giải mã và xác thực dữ liệu nhận được.

Ngoài ra, máy chủ còn sử dụng hàm băm Ascon Hash nhằm tạo ra khóa phiên cho quá trình giải mã. Cơ chế *safe counter* cũng được triển khai tại đây nhằm tăng tính bảo mật của hệ thống. Hệ thống Backend được triển khai trên nền tảng Node.js, với JavaScript là ngôn ngữ lập trình chính, giúp đảm bảo khả năng mở rộng, xử lý bất đồng bộ hiệu quả và dễ dàng tích hợp với các thành phần khác trong hệ thống.

3.2.2 Mô hình giao tiếp của hệ thống

Hệ thống IoT được chia thành ba quá trình giao tiếp: khởi tạo, trao đổi khóa và truyền dữ liệu. Tất cả dữ liệu trên đường truyền được đảm bảo mã hóa và xác thực. Ngoài ra, tất cả quá trình truyền sẽ được đóng gói thành các khung truyền được cấu hình độc nhất cho mỗi gói tin trình bày ở phần 3.1.1. Hình 3.10 minh họa mô hình giao tiếp của hệ thống.



Hình 3.10: Mô hình giao tiếp của hệ thống

Mỗi khi thiết bị khởi động hoặc reboot, phiên khởi tạo sẽ được kích hoạt. Trong phiên này, máy chủ sẽ truyền các thông tin đồng bộ, bao gồm *safe counter* và *key index* đến thiết bị. Những dữ liệu này đóng vai trò thiết yếu trong việc đảm bảo đồng bộ hóa, khôi phục trạng thái hệ thống, và duy trì tính toàn vẹn của quá trình truyền dữ liệu. Phần 3.2.5 sẽ trình bày ý nghĩa của *safe counter* và phần 3.2.3 sẽ trình bày chi tiết về cách hoạt động của *key index*.

Trong quá trình trao đổi khóa, gateway và máy chủ sẽ trao đổi khóa với nhau bằng thuật toán ECDH để có thể tạo ra hai khóa bí mật sử dụng cho mã hóa và giải mã. Dữ liệu sẽ được mã hóa ngay tại lớp nhận thức, vì vậy sau khi sinh thành công khóa bí mật, gateway sẽ chuyển khóa này đến lớp giao thức để tiến hành mã hóa dữ liệu. Để tránh

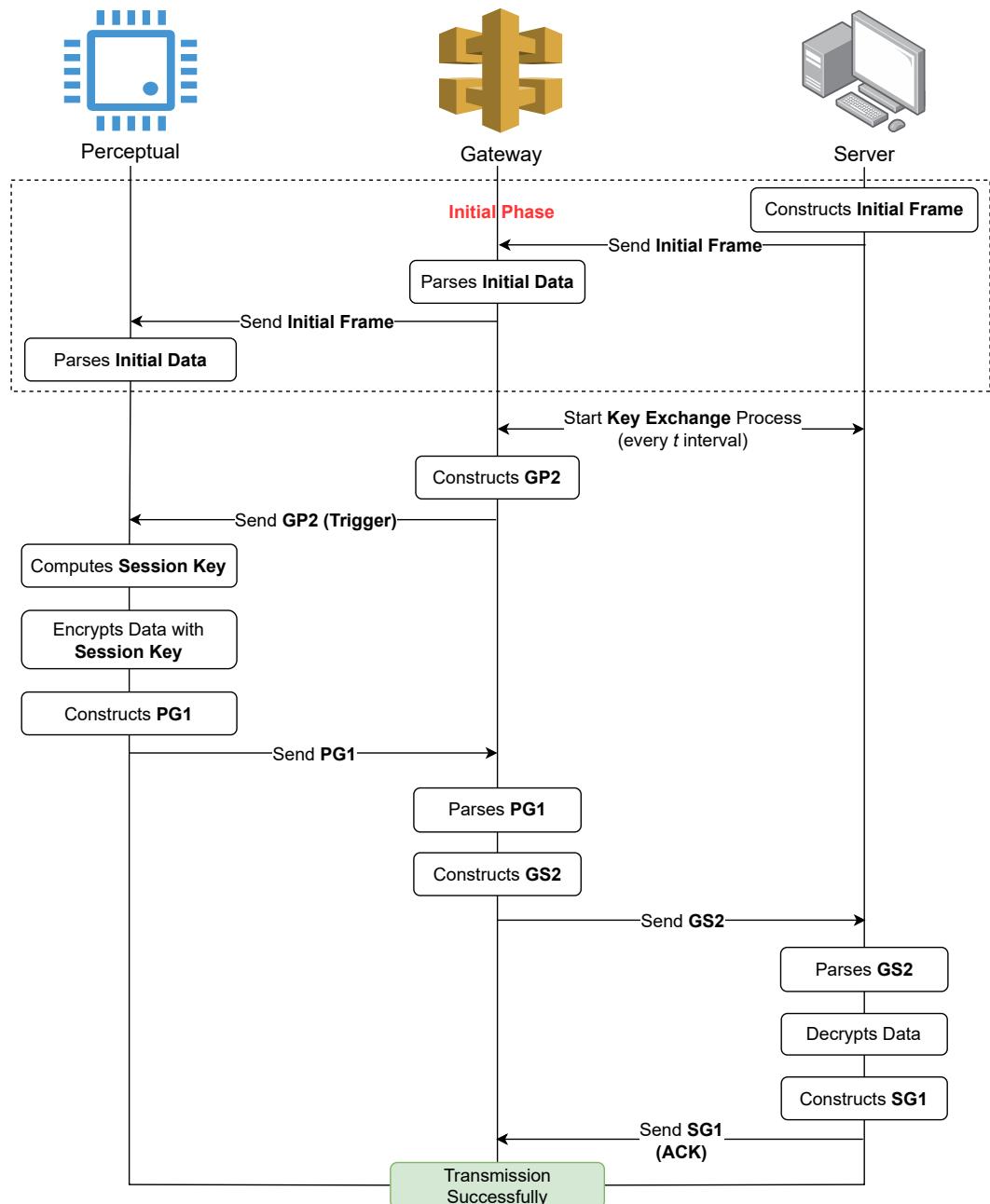
lộ khóa bí mật, gateway tiến hành mã hóa khóa này bằng một khóa chia sẻ trước (pre-shared key) trước khi truyền xuống lớp nhận thức. Tại lớp nhận thức, khóa được giải mã và sử dụng cho quá trình mã hóa dữ liệu. Cơ chế này giúp đảm bảo khóa bí mật không bị lộ hoặc đánh chặn trong quá trình trao đổi khóa, qua đó tăng cường tính an toàn cho hệ thống.

Phần lớn thời gian STM32 tại lớp nhận thức sẽ ở trong chế độ ngủ (Sleep Mode) để có thể tiết kiệm năng lượng, vì vậy để bắt đầu quá trình truyền dữ liệu, gateway trước tiên sẽ gửi một tín hiệu kích hoạt (trigger signal) cho lớp nhận thức. Khi nhận được tín hiệu, lớp nhận thức sẽ thoát khỏi sleep mode để tiến hành thu thập và mã hóa dữ liệu thu được bằng các khóa phiên được băm từ khóa bí mật (như trình bày ở phần 3.1.5) và gửi lên cho gateway. Gateway sẽ tiếp tục chuyển tiếp dữ liệu này lên cho máy chủ để xử lý. Nếu máy chủ có thể giải mã dữ liệu thành công, máy chủ sẽ gửi lại cho gateway một tín hiệu ACK nhằm thông báo phiên truyền dữ liệu thành công và tiếp tục bắt đầu phiên truyền mới. Trong trường hợp gateway không nhận được tín hiệu ACK từ máy chủ sau một khoảng thời gian nhất định (timeout), điều này có nghĩa dữ liệu đã không được truyền thành công hoặc giải mã không thành công, gateway sẽ gửi một gói tin thông báo cho STM32 ở lớp nhận thức để STM32 đồng bộ lại dữ liệu và bắt đầu lại phiên truyền. Hình 3.11 mô tả chi tiết toàn bộ quá trình giao tiếp của hệ thống bao gồm các khung truyền sẽ được trình bày chi tiết trong phần 3.2.4.

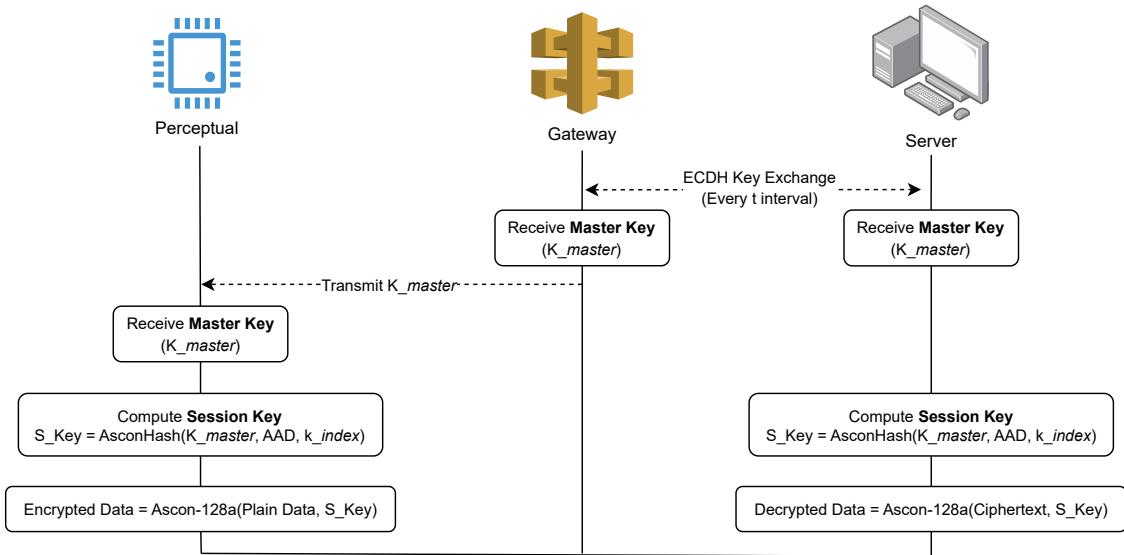
3.2.3 Phương pháp tạo khóa phiên từ khóa bí mật sử dụng trong mã hóa và giải mã dữ liệu

Để đảm bảo an toàn cho hệ thống IoT, việc thiết kế một phương thức trao đổi khóa bí mật an toàn là vô cùng cần thiết. Khóa bí mật này đóng vai trò quan trọng trong việc mã hóa và giải mã dữ liệu, góp phần bảo vệ tính toàn vẹn và bảo mật thông tin truyền tải. Nếu hệ thống IoT sử dụng phương pháp trao đổi khóa, tạo khóa kém an toàn hoặc dựa trên khóa tĩnh được cài đặt một lần duy nhất, nguy cơ lộ khóa hoặc bị tấn công brute-force với khóa có độ dài bit bảo mật thấp sẽ tăng lên đáng kể.

Như đã trình bày trong phần 3.1.4.2, nhược điểm lớn của ECDH chính là chi phí tính toán lớn, điều này có thể ảnh hưởng đến độ trễ của hệ thống. Vì vậy, để đảm bảo khóa luôn được làm mới và hạn chế quá trình trao đổi khóa liên tục, đề tài đề xuất một phương pháp tạo khóa phiên dựa trên khóa bí mật để giải quyết vấn đề trên. Trong đề tài này, thuật toán trao đổi khóa Elliptic Curve Diffie-Hellman (ECDH) được áp dụng nhằm thiết lập khóa bí mật trên kênh truyền không an toàn, đồng thời kết hợp với hàm băm nhẹ ASCON để sinh các khóa phiên (session key) phục vụ cho quá trình mã hóa, giải mã dữ liệu. Hình 3.12 thể hiện quá trình tạo khóa phiên đề xuất.



Hình 3.11: Luồng hoạt động của hệ thống



Hình 3.12: Quy trình tạo khóa phiên

Sau khi hoàn thành trao đổi khóa như đã trình bày ở phần 3.1.4.2, gateway và server sẽ tạo ra hai khóa bí mật, ở phần này sẽ gọi là Master Key (K_{master}). Thời gian giữa mỗi lần trao đổi K_{master} được gọi là t , được cấu hình tùy vào yêu cầu bảo mật của hệ thống, t càng nhỏ thì quá trình trao đổi K_{master} sẽ diễn ra với tần suất nhiều hơn nhưng bù lại sẽ tốn nhiều tài nguyên và thời gian cho việc trao đổi khóa. Và như đã trình bày ở phần 3.2.2, K_{master} sẽ được truyền xuống lớp nhận thức để phục vụ quá trình mã hóa. Trước khi lớp nhận thức mã hóa dữ liệu, khóa phiên sẽ được tạo ra bằng hàm băm ASCON. Đầu vào của hàm băm ASCON là sự kết hợp của các thành phần sau: K_{master} , AAD và k_{index} . Trong đó AAD là Associated Data được sử dụng trong mã hóa xác thực Ascon-128a và k_{index} chính là thứ tự của khóa hiện tại và tăng dần sau mỗi phiên.

$$input = K_{master} || AAD || k_{index}$$

Kết quả của hàm băm với đầu vào $input$ sẽ cho ra khóa phiên và sử dụng cho việc mã hóa dữ liệu. Phía server cũng sẽ sử dụng quá trình tương tự để tạo khóa phiên cho việc giải mã. Với k_{index} thay đổi liên tục, khóa phiên sinh ra cũng sẽ thay đổi liên tục, đảm bảo việc dò khóa sẽ rất phức tạp.

$$K_{session} = \text{AsconHash}(K_{master} || AAD || k_{index})$$

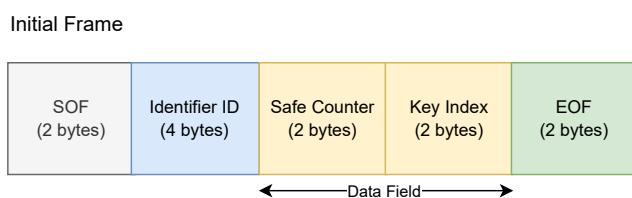
3.2.4 Kiến trúc khung truyền giao tiếp giữa các lớp trong hệ thống IoT

Phần này sẽ trình bày chi tiết về kiến trúc khung truyền được sử dụng trong hệ thống, được sử dụng trong giao tiếp giữa các lớp như minh họa ở hình 3.10. Tất cả các khung đều có định dạng chung được gọi là trường chuẩn, khởi đầu với 2-byte *Start of Frame* (SOF) để một phiên truyền. Tiếp theo đó sẽ là 4-byte *Identifier ID*, định danh thiết bị truyền dữ liệu, *ID* này sẽ được cấu hình tĩnh bên trong thiết bị ở lớp nhận thức. Khung truyền sẽ kết thúc bằng 2-byte *End of Frame* (EOF). Trường chứa dữ liệu (*Data field*) của khung truyền sẽ thay đổi tùy theo mục đích của quá trình truyền, ví dụ trường dữ liệu của khung truyền trao đổi khóa sẽ khác với trường dữ liệu của khung truyền dữ liệu mã hóa. Ngoài các trường chuẩn ở trên, mỗi khung truyền sẽ có thêm một số trường bổ sung để tăng cường độ bảo mật và tin cậy của hệ thống.

Để dễ hiểu, mỗi khung truyền sẽ được đặt tên dựa trên lớp nguồn và lớp đích, ngoại trừ khung truyền khởi tạo; ví dụ, một khung truyền từ gateway đến máy chủ (server) sẽ được ký hiệu là GS (Gateway-Server). Đối với dữ liệu liên quan (Associated Data - AAD), ở phần này sẽ sử dụng hai AAD bao gồm AAD_{data} và AAD_{key} . Trong đó, AAD_{data} được tính toán mỗi phiên truyền dữ liệu và AAD_{key} được cấu hình sẵn và sử dụng trong các phiên trao đổi khóa giữa gateway và máy chủ. Phần này sẽ chia các khung truyền thành ba nhóm chính: khung truyền khởi tạo, khung truyền trong giao tiếp giữa lớp nhận thức và gateway, khung truyền trong giao tiếp giữa gateway và máy chủ.

3.2.4.1 Khung truyền khởi tạo

Khung truyền khởi tạo này có kiến trúc rất đơn giản, chỉ bao gồm các trường chuẩn như trình bày ở phần trên. Khung truyền này chứa các dữ liệu cần thiết giúp máy chủ đồng bộ giao tiếp với thiết bị. Trường dữ liệu của khung khởi tạo bao gồm 2-byte *safe counter* và 2-byte *key index* hỗ trợ thiết lập trạng thái ban đầu và đảm bảo tính toàn vẹn hệ thống. Trong suốt vòng đời hoạt động, khung khởi tạo chỉ được kích hoạt một lần duy nhất ngay sau khi thiết bị khởi động lại.



Hình 3.13: Khung truyền khởi tạo

3.2.4.2 Khung truyền trong giao tiếp giữa lớp nhận thức và lớp mạng

Phần này mô tả chi tiết cấu trúc khung truyền trong quá trình giao tiếp giữa lớp nhận thức và lớp mạng. Hình 3.14 minh họa khung truyền được sử dụng khi dữ liệu được gửi từ lớp mạng xuống lớp nhận thức, ký hiệu là GP. Trong khung truyền này, một trường 1 byte có tên là *packet type* được thêm vào để xác định loại dữ liệu đang được truyền. Hai loại khung chính được định nghĩa: một dành cho việc truyền khóa bí mật (GP1), và một dành cho truyền tín hiệu kích hoạt (GP2).

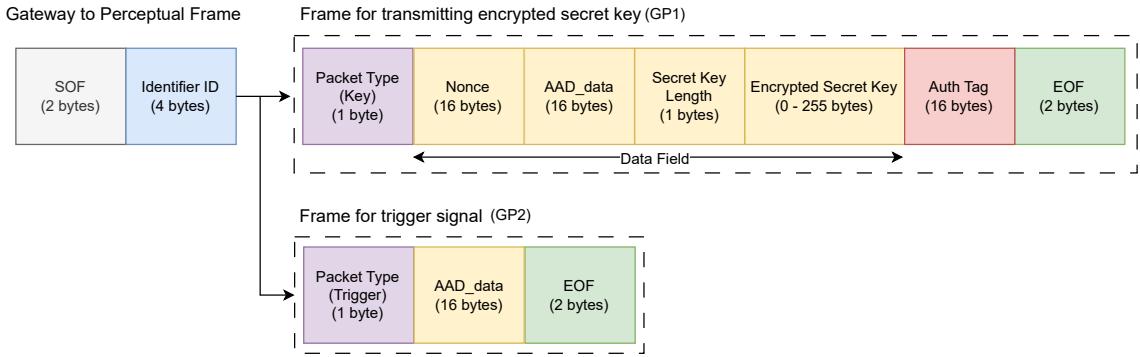
Với khung truyền GP2, trường dữ liệu của khung sẽ chỉ chứa 16-byte AAD_{data} dùng cho mã hóa dữ liệu, tức là, lớp nhận thức sẽ sử dụng 16-byte AAD_{data} này cho việc mã hóa dữ liệu trước khi truyền đi. Các trường chuẩn của GP2 vẫn được giữ nguyên.

Khung truyền GP1 sẽ có sự phức tạp hơn do đây là khung đảm nhận vai trò truyền khóa bí mật xuống lớp nhận thức, vì đây là dữ liệu nhạy cảm nên một số trường bổ sung được thêm vào. Trường dữ liệu của GP1 sẽ bao gồm 16-byte *Nonce* và AAD_{data} , AAD_{data} vẫn sẽ được sử dụng trong mã hóa dữ liệu trước khi truyền. *Nonce* và AAD_{key} là hai dữ liệu quan trọng để lớp nhận thức có thể giải mã khóa bí mật (khóa bí mật sẽ được mã hóa bởi gateway trước khi truyền xuống lớp nhận thức như trình bày ở phần 3.2.2) với AAD_{key} đã được cấu hình tĩnh trước cho cả lớp giao thức và gateway. Tiếp theo đó, sẽ là 1-byte *secret key length* và phần tải chứa khóa bí mật đã được mã hóa, trong đó 1-byte *secret key length* biểu thị độ dài của khóa bí mật. Một trường quan trọng được bổ sung vào khung GP1 đó chính là 16-byte *authentication tag* (Auth Tag), được sinh ra trong quá trình mã hóa khóa bí mật với Ascon-128a. Phần 2.2.2 đã trình bày việc tấn công Man-in-the-Middle có thể đánh tráo khóa trên đường truyền, vì vậy Auth Tag được thêm vào để đảm bảo dữ liệu được xác thực, nếu việc tính toán Auth Tag sai và không trùng khớp, lớp nhận thức sẽ ngay lập tức từ chối khóa bí mật này, tránh việc sử dụng khóa giả mạo cho việc mã hóa và giải mã.

Về cách tính AAD_{data} trong GP1 và GP2, AAD_{data} được ghép từ một số trường dữ liệu trong GS2 (trình bày ở phần 3.2.4.3). Vì thế, AAD_{data} sẽ được tính như sau:

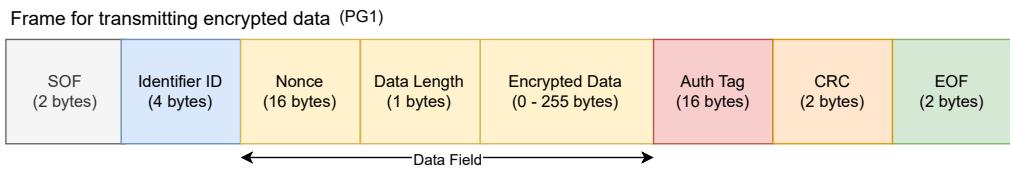
$$AAD_{data} = \text{SOF}_2 || \text{Identifier ID}_4 || \text{Sequence Number}_4 || \text{Nonce}_4 || \text{EOF}_2$$

Khi nhận tín hiệu kích hoạt (khung GP2) từ gateway, lớp nhận thức sẽ tạo khung PG1 chứa dữ liệu mã hóa và gửi cho gateway (minh họa hình 3.15). Trường dữ liệu của khung PG1 chứa 16-byte *Nonce*, đây là *Nonce* được dùng trong mã hóa dữ liệu và phía máy chủ sẽ dùng *Nonce* này cho giải mã. Tiếp theo, 16-byte Auth Tag sẽ được gửi nhằm xác thực dữ liệu mã hóa, máy chủ sau khi giải mã sẽ so sánh Auth Tag, nếu có sự khác biệt về Auth Tag, máy chủ sẽ từ chối gói tin đó. Ngoài ra, 2-byte *Cyclic Redundancy Check*



Hình 3.14: Khung truyền trao truyền dữ liệu từ gateway xuống lớp nhận thức

(CRC) giúp phát hiện có sự sai lệch hay mất mát dữ liệu nào trong quá trình truyền.



Hình 3.15: Khung truyền trong truyền dữ liệu từ lớp nhận thức lên gateway

3.2.4.3 Khung truyền trong giao tiếp giữa gateway và server

Nhóm các khung truyền trong quá trình giao tiếp giữa gateway và máy chủ được minh họa trong Hình 3.17. Tổng cộng có bốn loại khung truyền chính, bao gồm: GS1, GS2, SG1 và một khung truyền đặc biệt dùng để gửi thông tin trạng thái hiện tại của hệ thống lên máy chủ.

Khung truyền GS1 là khung được sử dụng để truyền thông tin về khóa công khai, phục vụ cho quá trình trao đổi khóa giữa gateway và máy chủ. Các trường chuẩn trong cấu trúc khung được giữ nguyên. Trường dữ liệu của khung chứa một *Nonce* có độ dài 16 byte, dùng trong quá trình tính toán thẻ xác thực, tiếp đó là 1-byte *public key length* và phần tải chứa khóa công khai. Do trường dữ liệu chứa thông tin quan trọng, việc bổ sung 16 byte *Auth Tag* giúp tăng cường bảo mật và hạn chế nguy cơ giả mạo khóa trong quá trình trao đổi. *Auth Tag* này được tạo ra bằng thuật toán Ascon-128a, sử dụng khóa chia sẻ trước cùng với giá trị *AAD_{key}* đã được cấu hình sẵn giữa gateway và máy chủ. Để tối ưu hiệu suất tính toán, phần bản rõ (plaintext) được bỏ trống vì chỉ cần tạo *Auth Tag*. Điều này giúp giảm thời gian xử lý cho cả gateway và máy chủ, đồng thời vẫn đảm bảo tính toàn vẹn của khóa công khai. Sau khi nhận được khung GS1 từ gateway, máy chủ sẽ tiến hành kiểm tra các trường trong khung. Tiếp theo, máy chủ sử dụng thuật toán giải mã Ascon-128a với bản mã (ciphertext) rỗng, cùng với *Nonce* và giá trị *AAD_{key}* để tính

lại Auth Tag. Auth Tag này sau đó được so sánh với Auth Tag nhận được nhằm xác minh tính toàn vẹn của thông tin khóa công khai. Hình 3.16 minh họa việc tính toán Auth Tag trong quá trình trao đổi khóa.

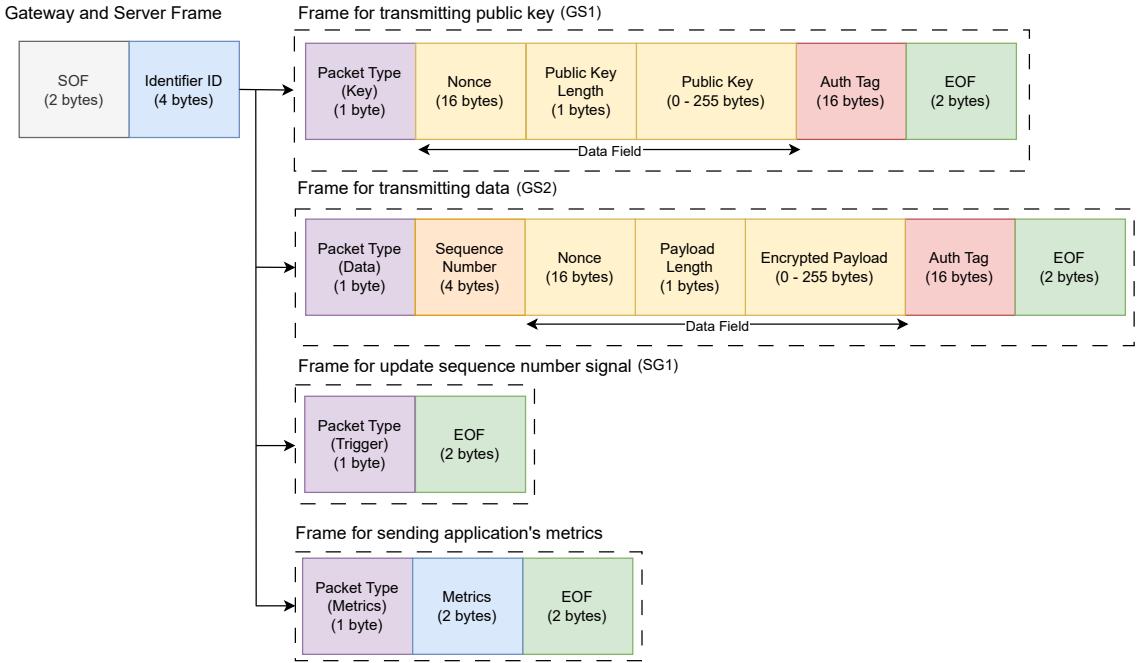


Hình 3.16: Tính toán thẻ xác thực (Auth Tag) trong trao đổi khóa

Trong truyền dữ liệu mã hóa từ gateway lên máy chủ, khung truyền GS2 sẽ được sử dụng. Khung truyền này giữ lại các trường tương tự như GS1, tuy nhiên, GS2 được cấu hình thêm một trường 4-byte *sequence number*. Khung truyền GS2 đóng vai trò chính trong quá trình truyền dữ liệu và được sử dụng thường xuyên trong suốt thời gian hệ thống hoạt động. Đây là loại khung được sử dụng phổ biến nhất, do đảm nhiệm chức năng truyền tải dữ liệu giữa gateway và máy chủ một cách liên tục và ổn định. Việc bổ sung 4 byte *sequence number* vào khung truyền giúp tăng cường bảo mật trong quá trình truyền dữ liệu, nhằm ngăn chặn các cuộc tấn công như tấn công phát lại (replay attack) hoặc tấn công xen giữa (Man-in-the-Middle). Khi giải mã khung truyền, máy chủ sẽ kiểm tra *sequence number* hiện tại của gói tin. Nếu phát hiện bất kỳ sai lệch nào về giá trị này, gói tin sẽ bị từ chối ngay lập tức. Cơ chế này làm giảm nguy cơ tấn công phát lại, vì ngay cả khi kẻ tấn công thu được một gói tin cũ và cố gắng gửi lại, *sequence number* bên trong gói tin sẽ không còn phù hợp với trạng thái hiện tại, do giá trị này được tăng tuần tự qua từng phiên truyền. Trường dữ liệu của khung GS2 bao gồm một *Nonce* có độ dài 16 byte, một trường *payload length* dài 1 byte, phần tải chứa dữ liệu đã được mã hóa, và cuối cùng là Auth Tag dài 16 byte được tạo ra trong quá trình mã hóa dữ liệu. Các thành phần như *Nonce*, dữ liệu mã hóa và Auth Tag đều được trích xuất từ khung PG1 do lớp nhận thức gửi lên gateway. Khi máy chủ nhận được khung GS2, nó sẽ tách các trường cần thiết để tái tạo giá trị AAD_{data} , phục vụ cho quá trình giải mã và xác thực dữ liệu.

Khung SG1 được sử dụng như một tín hiệu điều khiển gửi từ máy chủ về gateway. Hiện tại, SG1 được cấu hình để phục vụ hai loại tín hiệu chính: tín hiệu xác nhận (ACK) nhằm thông báo rằng phiên truyền dữ liệu đã thành công, và tín hiệu cập nhật *sequence number* mới. Trong một số tình huống, khi máy chủ xác định cần cập nhật giá trị *sequence number*, nó sẽ sử dụng khung SG1 để gửi tín hiệu tương ứng đến gateway nhằm đảm bảo sự đồng bộ giữa hai bên.

Cuối cùng là một loại khung truyền đặc biệt, dùng để gửi các thông tin về tình trạng hoạt động của hệ thống, bao gồm các chỉ số như tỷ lệ chuyển gói thành công (PDR – Packet Delivery Ratio), độ trễ, số lượng gói tin đã truyền, và các thông số liên quan khác.

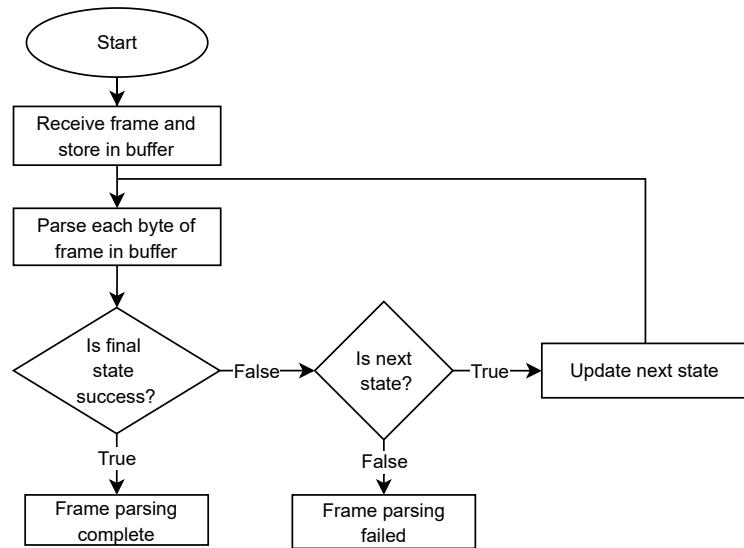


Hình 3.17: Bộ khung truyền trong giao tiếp giữa gateway và máy chủ

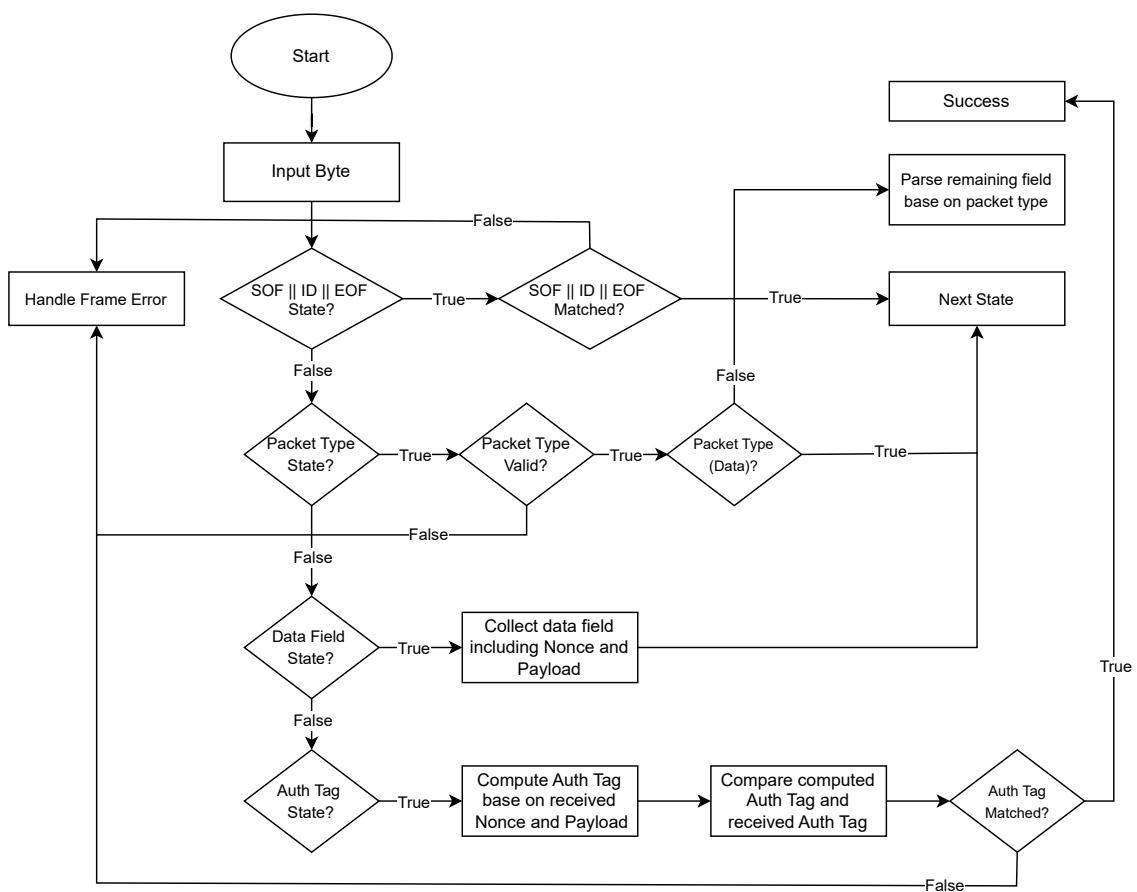
Khung truyền này được gửi định kỳ lên máy chủ, thường mỗi vài phút một lần, nhằm cung cấp dữ liệu giám sát giúp người dùng theo dõi và đánh giá hiệu suất của hệ thống.

3.2.4.4 Quá trình phân giải khung truyền

Quá trình phân giải khung truyền được trình bày trong hình 3.19 và 3.18. Tất cả các khung truyền được sử dụng trong hệ thống đều áp dụng chung quá trình phân giải này. Vì vậy, để đảm bảo tính ngắn gọn cũng như dễ hiểu trong báo cáo, khung truyền GS2 sẽ được minh họa trong hình để trình bày về quá trình phân giải khung truyền. Việc phân giải bắt đầu từ việc nhận dữ liệu từ thiết bị đích và lưu trữ vào trong một bộ đệm nội bộ (internal buffer), để chuẩn bị cho quá trình phân giải. Mỗi byte trong bộ đệm sẽ được luân phiên đưa vào hàm phân giải khung truyền (Hình 3.18). Trong mỗi hàm phân giải, các điều kiện ở mỗi trường trong khung truyền được cấu hình sẵn. Từ đó, hàm phân giải sẽ kiểm tra số byte tương ứng trong bộ đệm nội bộ với điều kiện; nếu điều kiện đúng, hàm phân giải sẽ tiến hành kiểm tra trường tiếp theo; ngược lại, sẽ ngừng quá trình phân giải và trả về mã lỗi. Nếu hàm phân giải có thể phân giải được tất cả các trường trong bộ đệm, quá trình phân giải sẽ hoàn tất. Việc mã hóa dữ liệu sẽ được diễn ra trong quá trình phân giải này, cụ thể là trong bước tính thẻ xác thực. Ở bước này, dữ liệu sẽ được mã hóa, từ đó sinh ra thẻ xác thực, sau đó so sánh với thẻ xác thực nhận được để kiểm tra tính toàn vẹn của dữ liệu.



Hình 3.18: Quá trình phân giải khung truyền



Hình 3.19: Hàm phân giải khung truyền GS2

3.2.5 Cơ chế safe counter

Như đã đề cập ở phần 3.2.4.3, khung truyền GS2 được cấu hình thêm 4-byte *sequence number* để hạn chế khả năng tấn công phát lại (replay attack) và tấn công xen giữa (Man-in-the-Middle). Tuy nhiên, một số trường hợp kẻ tấn công có thể gửi rất nhiều gói tin với các *sequence number* khác nhau nhằm cố gắng brute-force thành công *sequence number* hiện tại của hệ thống. Có thể thấy rằng với 4 byte *sequence number*, sau khi toàn bộ không gian 2^{32} giá trị được thử, xác suất brute-force thành công sẽ đạt 100%. Vấn đề đặt ra ở đây là xác suất thành công của brute-force tăng dần theo số lần thử — tức là bài toán mang tính xác suất tích lũy [?]. Để khắc phục điều này, cơ chế safe counter được triển khai nhằm chuyển bài toán từ dạng có xác suất tăng dần thành một quá trình với xác suất thành công cố định ở mỗi lần thử, từ đó làm cho brute-force trở nên kém hiệu quả hơn. *Safe counter* thực chất là một biến đếm 32-bit đồng bộ giữa gateway và máy chủ, biến đếm này sẽ được cập nhật liên tục trong quá trình truyền dữ liệu. Trong trường hợp bị brute-force, máy chủ sẽ từ chối các gói tin sai về *sequence number* trong quá trình phân giải, số gói tin mà máy chủ từ chối sẽ được gọi là R . Ở máy chủ, một ngưỡng được đặt ra gọi là T_{reject} , nếu máy chủ từ chối một số lượng gói tin chạm ngưỡng T_{reject} , hay nói cách khác nếu $R \geq T_{reject}$, máy chủ sẽ ngay lập tức gửi một tín hiệu tới gateway bằng khung truyền SG1 và thực hiện việc cập nhật *sequence number* mới dựa trên *safe counter* hiện tại của hệ thống. Vì *safe counter* được đồng bộ giữa gateway và máy chủ, việc tính toán lại *sequence number* sẽ cho ra kết quả giống nhau, đảm bảo việc truyền dữ liệu giữa gateway và máy chủ vẫn diễn ra bình thường với *sequence number* mới. Thuật toán 1 trình bày chi tiết cách tính *sequence number* mới dựa trên *safe counter*.

Algorithm 1 Cơ chế safe counter

Require: 32-bit Safe Counter SC , Secret Key K , Threshold $T_{rejected}$, Rejected Packet Count R , Current Sequence Number SN

- 1: $SC \leftarrow (SC + ((SC \ll 3) \oplus (SC \gg 2) \oplus 7)) \bmod 2^{32}$
- 2: **if** $R \geq T_{rejected}$ **then**
- 3: $SN \leftarrow (SC \oplus K) \bmod 2^{32}$
- 4: $R \leftarrow 0$
- 5: **return** SN

Với cơ chế này, kẻ tấn công sẽ có T_{reject} cơ hội để đoán trước khi *sequence number* được cập nhật lại một giá trị mới ngẫu nhiên. Tỉ lệ đoán thành công *sequence number* sau T_{reject} lần thử có công thức là:

$$P_{\text{guess}} \approx 1 - \left(1 - \frac{1}{2^{32}}\right)^{T_{\text{rejected}}}.$$

Trong hệ thống thử nghiệm, T_{reject} được đặt mặc định là 10, vì thế tỉ lệ cố định để có thể brute-force thành công *sequence number* là:

$$P_{\text{guess}} \approx 1 - \left(1 - \frac{1}{2^{32}}\right)^{10} \approx 0.00000023\%$$

Thử nghiệm trên 70,000 gói tin giả mạo với các *sequence number* khác nhau được gửi vào hệ thống, kết quả cho thấy máy chủ từ chối hoàn toàn 70,000 gói tin, đạt tỉ lệ chính xác 100%, cho thấy được độ tin cậy cao của cơ chế này.

Chương 4: Kết quả triển khai và phân tích

4.1 Mô hình thực nghiệm

Hệ thống IoT được xây dựng dựa trên kiến trúc IoT bốn lớp được đề xuất. Ở lớp nhận thức, vi điều khiển STM32F411VET6 được sử dụng, hoạt động với tần số 16 MHz. Ở lớp mạng, sử dụng module ESP32 với tần số 240 MHz, đóng vai trò là gateway, tổng hợp và luân chuyển dữ liệu lên máy chủ với cấu hình của máy chủ được trình bày trong bảng 4.1. Đôi với quá trình trao đổi khóa giữa máy chủ và gateway, mô hình được cấu hình sẵn thời gian giữa các lần trao đổi khóa là một phút, nhằm dễ đánh giá việc chi phí tính toán của thuật toán ECDH ảnh hưởng đến hệ thống như thế nào. Tuy nhiên, thời gian trao đổi khóa có thể được cấu hình khác nhau trong thực tế, tùy vào khả năng tính toán của hệ thống, năng lượng tiêu thụ và yêu cầu về bảo mật. Thời gian giữa mỗi lần truyền dữ liệu được đặt là 1 ms.

Bảng 4.1: Cấu hình máy chủ

Thành phần	Cấu hình
OS	Ubuntu 18.04.6 LTS
RAM	7.8 GiB
Storage	97 GiB
CPU	Intel(R) Xeon(R) CPU E5504 2.00 GHz

4.2 Kết quả, phân tích và đánh giá

Phần này nêu ra các kết quả thực hiện và hiệu suất của hệ thống. Đồng thời, đánh giá một số ưu nhược điểm của hệ thống cũng như khả năng chống lại một số cuộc tấn công phổ biến.

4.2.1 Kết quả thực hiện

Trên MCU STM32F411VET6, các kỹ thuật bao gồm kiến trúc khung truyền, kỹ thuật tạo khóa phiên và mã hóa dữ liệu sử dụng thuật toán mã hóa nhẹ Ascon-128a đã được triển khai thành công. Hình 4.1 trình bày kết quả thực hiện mã hóa và tạo khóa phiên bằng Debugger trên công cụ CubeIDE. Đôi với mỗi quá trình tạo khóa và mã hóa thành công, các biến kiểm tra sẽ được đặt thành 1 để báo quá trình thành công. Giá trị *session_key_index* thể hiện thứ tự phiên hiện tại trong quá trình giao tiếp, là tham số được sử dụng trong hàm tạo khóa phiên.

Expression	Type	Value
0x= is_encrypt_success	uint8_t	1 '\001'
0x= is_session_key_complete	uint8_t	1 '\001'
0x= time_encrypt_us	float	332.1875
0x= time_frame_construct_us	float	642
0x= time_encrypt_aes_us	float	936.9375
0x= time_parsing_frame_us	float	43.25
0x= time_frame_transmit_us	float	5580.4375
0x= session_key_index	uint16_t	1248
0x= mocker6	uint8_t	171 '\u00a6'
0x= mocker7	uint8_t	4 '\004'
0x= mocker8	uint8_t	118 'v'
0x= mocker9	uint8_t	234 '\u00e'
0x= mocker10	uint8_t	10 '\n'

Hình 4.1: Kết quả thực hiện trên MCU STM32F411VET6

Khi hệ thống được reboot, quá trình khởi tạo ngay lập tức hoạt động. ESP32 gửi một tín hiệu yêu cầu gói tin khởi tạo từ máy chủ. Sau đó, máy chủ sẽ xác minh thiết bị yêu cầu và gửi về dữ liệu khởi tạo. Dữ liệu khởi tạo này sau đó sẽ được ESP32 tiếp tục chuyển tiếp xuống STM32 ở lớp nhận thức. Kết quả thực hiện quá trình này được trình bày trong hình 4.2.

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TAILSCALE
TERMINAL
Attempting MQTT connection...
MQTT Connected
Subscribed to topics: handshake-send/ecdh
-- Init Session
-- Requesting initial data from server...
-- Publish succeeded
-- Waiting for initial data from server...
-- Received initial data from server
-- Fetching safe counter...
-- Fetching initial data from server...
-- Received safe counter: 163
-- Safe Counter: 163
-- Transmitting derivation index to STM32...
-- Init Completed. Wait for system to be stable...
[1/5] Checking key...
-- Key Expired! Renewing...
-- Constructed public key frame in 0.43 ms
-- Publish succeeded
-- Sent public key frame to server
-- Received public key frame from server
-- Shared secret computed successfully 52 8E E4 D0 83 FD D0 43 7D EC
4E 91 01 65 A2 C4
-- Handshake for key exchanging completed in 826.79 ms
-- Construct frame key to STM32...
-- Frame constructed in 0.67 ms
-- Transmitting key to STM32...
-- Transmit key to STM32 completed in 9.98 ms
[2/5] Transmitting trigger signal to STM32...
-- Constructing frame for transmitting trigger signal to STM32...
-- Command: 1
[1/3] Parse data frame completed
-- Received ECDH Handshake message
[2/3] Parsed Server Data Frame:
(index) Field Value
0 'Preamble' '\x0aa55'
1 'Identifier ID' '\x8118910'
2 'Packet Type' '\x01'
3 'Sequence Number' 5
4 'Nonce' '\x98395d94d42ae7410a9ca28bf7e191a3'
5 'Payload Length' 20
6 'Encrypted Payload' '\xf1567a8e8767dbfa5971922d7c2337b3883879b'
7 'MAC Tag' '\xe767db0fa5971922d7c2337b3883879b'
8 'End Marker' '\xaabb'

[2/3] Parsed sensor data completed
Safe counter current: 47
[1/3] Published handshake/ecdh completed
[3/3] Publish ACK to handshake-send/ecdh completed
[NICE] Everything is done
--> Processed handshake/ecdh in 70ms
-- Received ECDH Handshake message
[INITIAL SESSION] Retrieved initial session data from device: 135334160
[INITIAL SESSION] Published safe counter to handshake-send/ecdh completed
-- Received ECDH Handshake message
-- Parsed Handshake Frame:
(index) Field Value
0 'Preamble' '\x0aa55'
1 'Identifier ID' '\x8118910'
2 'Packet Type' '\x01'
3 'Sequence Number' 1
4 'Nonce' '\x100f9dc2506b4c17...'
5 'Shared Secret' '\xaabb'

-- Received client public key: \x100f9dc2506b4c17...
-- Successfully published server public key
-- Generated secret key: 528ee4d083fdd043...
--> Processed handshake/ecdh in 87ms
-- Received ECDH Handshake message
Start Parsing Data Frame

```

Hình 4.2: Quá trình khởi tạo khi reboot

Kỹ thuật trao đổi khóa sử dụng thuật toán ECDH được triển khai thành công trên ESP32 IoT Gateway và hệ thống Backend. Thứ tự thực hiện quá trình trao đổi khóa được thực hiện chính xác bao gồm gửi khóa chung, nhận khóa chung và tính toán khóa bí mật. Các khóa chung khi publish đều được cấu trúc bằng khung truyền. Hình 4.3 trình bày kết quả thực hiện trên ESP32 Gateway và máy chủ.

Quá trình truyền dữ liệu được triển khai thành công theo đúng trình tự được trình bày trong hình 3.10 và 3.11. Trên ESP32, quá trình truyền được chia thành năm giai

```

[1/5] Checking key...
-- Key Expired! Reusing...
-- Constructing public key frame in 0.38 ms
-- Publish succeeded
-- Sent public key frame to server
-- Received public key from server
-- Shared secret computed successfully 43 C5 C2 72 C4 96 29 1E 8F 65 EC 4D 78 4A D4 36
-- Handshake for key exchanging completed in 824.76 ms
-- Construct frame key to STM32...
-- Transmit key to STM32 completed in 9.96 ms
[2/5] Transmitting trigger signal to STM32...
-- Constructing frame for transmitting trigger signal to STM32...
-- Command: 4
-- Transmitting trigger signal to STM32
-- Transmit trigger signal to STM32 completed in 9.93 ms
[3/5] Frame processing...
-- Receiving data from STM32
-- Frame parsing completed in 0.35 ms
-- Frame parsing completed in 32.59 ms
[4/5] Construct and send data to server...
Safe counter current: 148
-- Constructing server data frame
-- Sequence number: 1
Server data frame constructed in 0.0990 ms
-- Publish succeeded
-- Construct frame key to server in 4.8140 ms
-- Send data to server completed in 16.00 ms
[5/5] Waiting for ACK package...
-- Received ACK package from server in 224.48 ms
[SPECIAL PACKET PER MINUTE] SENDING METRICS: PDR: 12.50%, AVG LATENCY: 1152.84 ms
-- Publish succeeded
Metrics sent - PDR: 12.50, Latency: 1152.84, Packets: 1
[NICE] Transmission completed!
[1/5] Checking key...
-- Key is still valid
[2/5] Transmitting trigger signal to STM32...
-- Constructing frame for transmitting trigger signal to STM32...
-- Command: 1
-- Transmitting trigger signal to STM32
-- Transmit trigger signal to STM32 completed in 9.95 ms

```

(index)	Field	Value
0	'Preamble'	'0xa55'
1	'Identifier ID'	'0x8110910'
2	'Packet Type'	'0x3'
3	'Public Key'	'5df8f72d5bb2a84d...'
4	'End Marker'	'0xaabb'

```

-- Received client public key: 5df8f72d5bb2a84d...
-- Successfully published server public key
-- Generated secret key: 43c5c272c496291e...
-- Published session key in 124ms
-- Received ECDH Handshake message
Start Parsing Data Frame
Derived key successfully: 0b16b0eaad39fd31...
-- Session key generated: 0b16b0eaad39fd31...
-- Derived session key: 0b16b0eaad39fd31...
[1/3] Parse data frame completed
-- Parsed Server Data Frame:

```

```

npm start
[1/3] [TOPIC_SUBSCRIBED] ESP32Client + handshake-send/ecdh
[1/3] [CLIENT_DISCONNECTED] ESP32Client
[1/3] [CLIENT_CONNECTED] ESP32Client
[1/3] [TOPIC_SUBSCRIBED] mqtts_cu<23801 + handshake-send/ecdh
[1/3] [CLIENT_DISCONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] ESP32Client
[1/3] [TOPIC_SUBSCRIBED] ESP32Client + handshake-send/ecdh
[1/3] [CLIENT_DISCONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] ESP32Client
[1/3] [TOPIC_SUBSCRIBED] ESP32Client + handshake-send/ecdh
[1/3] [CLIENT_CONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] mqtts_cu<23801 + sensors/data
[1/3] [TOPIC_SUBSCRIBED] mqtts_cu<23801 + encrypt/dexchange
[1/3] [TOPIC_CONNECTED] mqtts_cu<23801 + metrics/data
[1/3] [TOPIC_CONNECTED] mqtts_cu<23801 + init/session
[1/3] [CLIENT_DISCONNECTED] ESP32Client
[1/3] [CLIENT_CONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] ESP32Client + handshake-send/ecdh
[1/3] [TOPIC_CONNECTED] mqtts_cu<23801 + sensors/data
[1/3] [TOPIC_CONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] ESP32Client + handshake-send/ecdh
[1/3] [TOPIC_CONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] ESP32Client + metrics/data
[1/3] [TOPIC_CONNECTED] ESP32Client + init/session
[1/3] [CLIENT_DISCONNECTED] ESP32Client
[1/3] [CLIENT_CONNECTED] ESP32Client
[1/3] [TOPIC_CONNECTED] ESP32Client + handshake-send/ecdh

```

Hình 4.3: Quá trình trao đổi khóa giữa gateway và máy chủ

đoạn chính bắt đầu bằng việc kiểm tra khóa, gửi tín hiệu kích hoạt đến cho STM32 ở lớp nhận thức, nhận khung dữ liệu từ STM32 và tiến hành phân giải, tạo khung truyền dữ liệu mới để gửi lên máy chủ, cuối cùng chờ tín hiệu ACK từ máy chủ. Về phần máy chủ, sau khi nhận khung truyền đã phân giải và mã hóa dữ liệu thành công sẽ gửi về tín hiệu ACK để kết thúc phiên. Các kết quả giao tiếp trên được trình bày trong hình 4.4.

```

[1/5] Checking key...
-- Received ACK package from server in 170.28 ms
[NICE] Transmission completed!
[2/5] Transmitting trigger signal to STM32...
-- Constructing frame for transmitting trigger signal to STM32...
-- Command: 1
-- Transmitting trigger signal to STM32
-- Transmit trigger signal to STM32 completed in 9.95 ms
[3/5] Frame processing...
-- Receiving data from STM32
-- Frame parsing completed in 0.33 ms
-- Frame parsing completed in 32.71 ms
[4/5] Construct and send data to server...
Safe counter current: 201
-- Constructing server data frame
-- Sequence number: 8
-- Server data frame constructed in 0.0950 ms
-- Publish succeeded
-- Sent server data frame to server in 5.7810 ms
-- Received ACK package from server in 6.58 ms
[NICE] Transmission completed!
[1/5] Checking key...
-- Key is still valid
[2/5] Transmitting trigger signal to STM32...
-- Constructing frame for transmitting trigger signal to STM32...
-- Command: 1
-- Transmitting trigger signal to STM32
-- Transmit trigger signal to STM32 completed in 9.95 ms
[3/5] Frame processing...
-- Receiving data from STM32
-- Frame parsing completed in 0.33 ms
-- Frame parsing completed in 32.71 ms
[4/5] Construct and send data to server...
Safe counter current: 78
-- Constructing server data frame
-- Sequence number: 9
-- Server data frame constructed in 0.0940 ms
-- Publish succeeded
-- Sent server data frame to server in 4.7570 ms

```

(index)	Field	Value
0	'Preamble'	'0xa55'
1	'Identifier ID'	'0x8110910'
2	'Packet Type'	'0x1'
4	'Sequence Number'	'0'
5	'Nonce'	'ffe7e0f22c73cf665421ff0010c4a14'
6	'Payload Length'	'20'
7	'Encrypted Payload'	'579e367f9df20d8d09911fd3ee3af1342d15eaa'
8	'MAC Tag'	'79df20d8d09911fd3ee3af1342d15eaa'
9	'End Marker'	'0xaabb'

```

[2/3] Parsed sensor data completed
Safe counter current: 164
-- data1: _data2: _data3: _data4: _data5: _data6: _data7: _data8: _data9: _data10: _data11: _data12: _data13: _data14: _data15: _data16: _data17: _data18: _data19: _data20: _data21: _data22: _data23: _data24: _data25: _data26: _data27: _data28: _data29: _data30: _data31: _data32: _data33: _data34: _data35: _data36: _data37: _data38: _data39: _data40: _data41: _data42: _data43: _data44: _data45: _data46: _data47: _data48: _data49: _data50: _data51: _data52: _data53: _data54: _data55: _data56: _data57: _data58: _data59: _data60: _data61: _data62: _data63: _data64: _data65: _data66: _data67: _data68: _data69: _data70: _data71: _data72: _data73: _data74: _data75: _data76: _data77: _data78: _data79: _data80: _data81: _data82: _data83: _data84: _data85: _data86: _data87: _data88: _data89: _data90: _data91: _data92: _data93: _data94: _data95: _data96: _data97: _data98: _data99: _data100: _data101: _data102: _data103: _data104: _data105: _data106: _data107: _data108: _data109: _data110: _data111: _data112: _data113: _data114: _data115: _data116: _data117: _data118: _data119: _data120: _data121: _data122: _data123: _data124: _data125: _data126: _data127: _data128: _data129: _data130: _data131: _data132: _data133: _data134: _data135: _data136: _data137: _data138: _data139: _data140: _data141: _data142: _data143: _data144: _data145: _data146: _data147: _data148: _data149: _data150: _data151: _data152: _data153: _data154: _data155: _data156: _data157: _data158: _data159: _data160: _data161: _data162: _data163: _data164: _data165: _data166: _data167: _data168: _data169: _data170: _data171: _data172: _data173: _data174: _data175: _data176: _data177: _data178: _data179: _data180: _data181: _data182: _data183: _data184: _data185: _data186: _data187: _data188: _data189: _data190: _data191: _data192: _data193: _data194: _data195: _data196: _data197: _data198: _data199: _data200: _data201: _data202: _data203: _data204: _data205: _data206: _data207: _data208: _data209: _data210: _data211: _data212: _data213: _data214: _data215: _data216: _data217: _data218: _data219: _data220: _data221: _data222: _data223: _data224: _data225: _data226: _data227: _data228: _data229: _data230: _data231: _data232: _data233: _data234: _data235: _data236: _data237: _data238: _data239: _data240: _data241: _data242: _data243: _data244: _data245: _data246: _data247: _data248: _data249: _data250: _data251: _data252: _data253: _data254: _data255: _data256: _data257: _data258: _data259: _data260: _data261: _data262: _data263: _data264: _data265: _data266: _data267: _data268: _data269: _data270: _data271: _data272: _data273: _data274: _data275: _data276: _data277: _data278: _data279: _data280: _data281: _data282: _data283: _data284: _data285: _data286: _data287: _data288: _data289: _data290: _data291: _data292: _data293: _data294: _data295: _data296: _data297: _data298: _data299: _data300: _data301: _data302: _data303: _data304: _data305: _data306: _data307: _data308: _data309: _data310: _data311: _data312: _data313: _data314: _data315: _data316: _data317: _data318: _data319: _data320: _data321: _data322: _data323: _data324: _data325: _data326: _data327: _data328: _data329: _data330: _data331: _data332: _data333: _data334: _data335: _data336: _data337: _data338: _data339: _data340: _data341: _data342: _data343: _data344: _data345: _data346: _data347: _data348: _data349: _data350: _data351: _data352: _data353: _data354: _data355: _data356: _data357: _data358: _data359: _data360: _data361: _data362: _data363: _data364: _data365: _data366: _data367: _data368: _data369: _data370: _data371: _data372: _data373: _data374: _data375: _data376: _data377: _data378: _data379: _data380: _data381: _data382: _data383: _data384: _data385: _data386: _data387: _data388: _data389: _data390: _data391: _data392: _data393: _data394: _data395: _data396: _data397: _data398: _data399: _data400: _data401: _data402: _data403: _data404: _data405: _data406: _data407: _data408: _data409: _data410: _data411: _data412: _data413: _data414: _data415: _data416: _data417: _data418: _data419: _data420: _data421: _data422: _data423: _data424: _data425: _data426: _data427: _data428: _data429: _data430: _data431: _data432: _data433: _data434: _data435: _data436: _data437: _data438: _data439: _data440: _data441: _data442: _data443: _data444: _data445: _data446: _data447: _data448: _data449: _data450: _data451: _data452: _data453: _data454: _data455: _data456: _data457: _data458: _data459: _data460: _data461: _data462: _data463: _data464: _data465: _data466: _data467: _data468: _data469: _data470: _data471: _data472: _data473: _data474: _data475: _data476: _data477: _data478: _data479: _data480: _data481: _data482: _data483: _data484: _data485: _data486: _data487: _data488: _data489: _data490: _data491: _data492: _data493: _data494: _data495: _data496: _data497: _data498: _data499: _data500: _data501: _data502: _data503: _data504: _data505: _data506: _data507: _data508: _data509: _data510: _data511: _data512: _data513: _data514: _data515: _data516: _data517: _data518: _data519: _data520: _data521: _data522: _data523: _data524: _data525: _data526: _data527: _data528: _data529: _data530: _data531: _data532: _data533: _data534: _data535: _data536: _data537: _data538: _data539: _data540: _data541: _data542: _data543: _data544: _data545: _data546: _data547: _data548: _data549: _data550: _data551: _data552: _data553: _data554: _data555: _data556: _data557: _data558: _data559: _data560: _data561: _data562: _data563: _data564: _data565: _data566: _data567: _data568: _data569: _data570: _data571: _data572: _data573: _data574: _data575: _data576: _data577: _data578: _data579: _data580: _data581: _data582: _data583: _data584: _data585: _data586: _data587: _data588: _data589: _data590: _data591: _data592: _data593: _data594: _data595: _data596: _data597: _data598: _data599: _data600: _data601: _data602: _data603: _data604: _data605: _data606: _data607: _data608: _data609: _data610: _data611: _data612: _data613: _data614: _data615: _data616: _data617: _data618: _data619: _data620: _data621: _data622: _data623: _data624: _data625: _data626: _data627: _data628: _data629: _data630: _data631: _data632: _data633: _data634: _data635: _data636: _data637: _data638: _data639: _data640: _data641: _data642: _data643: _data644: _data645: _data646: _data647: _data648: _data649: _data650: _data651: _data652: _data653: _data654: _data655: _data656: _data657: _data658: _data659: _data660: _data661: _data662: _data663: _data664: _data665: _data666: _data667: _data668: _data669: _data670: _data671: _data672: _data673: _data674: _data675: _data676: _data677: _data678: _data679: _data680: _data681: _data682: _data683: _data684: _data685: _data686: _data687: _data688: _data689: _data690: _data691: _data692: _data693: _data694: _data695: _data696: _data697: _data698: _data699: _data700: _data701: _data702: _data703: _data704: _data705: _data706: _data707: _data708: _data709: _data710: _data711: _data712: _data713: _data714: _data715: _data716: _data717: _data718: _data719: _data720: _data721: _data722: _data723: _data724: _data725: _data726: _data727: _data728: _data729: _data730: _data731: _data732: _data733: _data734: _data735: _data736: _data737: _data738: _data739: _data740: _data741: _data742: _data743: _data744: _data745: _data746: _data747: _data748: _data749: _data750: _data751: _data752: _data753: _data754: _data755: _data756: _data757: _data758: _data759: _data760: _data761: _data762: _data763: _data764: _data765: _data766: _data767: _data768: _data769: _data770: _data771: _data772: _data773: _data774: _data775: _data776: _data777: _data778: _data779: _data780: _data781: _data782: _data783: _data784: _data785: _data786: _data787: _data788: _data789: _data790: _data791: _data792: _data793: _data794: _data795: _data796: _data797: _data798: _data799: _data800: _data801: _data802: _data803: _data804: _data805: _data806: _data807: _data808: _data809: _data810: _data811: _data812: _data813: _data814: _data815: _data816: _data817: _data818: _data819: _data820: _data821: _data822: _data823: _data824: _data825: _data826: _data827: _data828: _data829: _data830: _data831: _data832: _data833: _data834: _data835: _data836: _data837: _data838: _data839: _data840: _data841: _data842: _data843: _data844: _data845: _data846: _data847: _data848: _data849: _data850: _data851: _data852: _data853: _data854: _data855: _data856: _data857: _data858: _data859: _data860: _data861: _data862: _data863: _data864: _data865: _data866: _data867: _data868: _data869: _data870: _data871: _data872: _data873: _data874: _data875: _data876: _data877: _data878: _data879: _data880: _data881: _data882: _data883: _data884: _data885: _data886: _data887: _data888: _data889: _data890: _data891: _data892: _data893: _data894: _data895: _data896: _data897: _data898: _data899: _data900: _data901: _data902: _data903: _data904: _data905: _data906: _data907: _data908: _data909: _data910: _data911: _data912: _data913: _data914: _data915: _data916: _data917: _data918: _data919: _data920: _data921: _data922: _data923: _data924: _data925: _data926: _data927: _data928: _data929: _data930: _data931: _data932: _data933: _data934: _data935: _data936: _data937: _data938: _data939: _data940: _data941: _data942: _data943: _data944: _data945: _data946: _data947: _data948: _data949: _data950: _data951: _data952: _data953: _data954: _data955: _data956: _data957: _data958: _data959: _data960: _data961: _data962: _data963: _data964: _data965: _data966: _data967: _data968: _data969: _data970: _data971: _data972: _data973: _data974: _data975: _data976: _data977: _data978: _data979: _data980: _data981: _data982: _data983: _data984: _data985: _data986: _data987: _data988: _data989: _data990: _data991: _data992: _data993: _data994: _data995: _data996: _data997: _data998: _data999: _data1000: _data1001: _data1002: _data1003: _data1004: _data1005: _data1006: _data1007: _data1008: _data1009: _data1010: _data1011: _data1012: _data1013: _data1014: _data1015: _data1016: _data1017: _data1018: _data1019: _data1020: _data1021: _data1022: _data1023: _data1024: _data1025: _data1026: _data1027: _data1028: _data1029: _data1030: _data1031: _data1032: _data1033: _data1034: _data1035: _data1036: _data1037: _data1038: _data1039: _data1040: _data1041: _data1042: _data1043: _data1044: _data1045: _data1046: _data1047: _data1048: _data1049: _data1050: _data1051: _data1052: _data1053: _data1054: _data1055: _data1056: _data1057: _data1058: _data1059: _data1060: _data1061: _data1062: _data1063: _data1064: _data1065: _data1066: _data1067: _data1068: _data1069: _data1070: _data1071: _data1072: _data1073: _data1074: _data1075: _data1076: _data1077: _data1078: _data1079: _data1080: _data1081: _data1082: _data1083: _data1084: _data1085: _data1086: _data1087: _data1088: _data1089: _data1090: _data1091: _data1092: _data1093: _data1094: _data1095: _data1096: _data1097: _data1098: _data1099: _data1100: _data1101: _data1102: _data1103: _data1104: _data1105: _data1106: _data1107: _data1108: _data1109: _data1110: _data1111: _data1112: _data1113: _data1114: _data1115: _data1116: _data1117: _data1118: _data1119: _data1120: _data1121: _data1122: _data1123: _data1124: _data1125: _data1126: _data1127: _data1128: _data1129: _data1130: _data1131: _data1132: _data1133: _data1134: _data1135: _data1136: _data1137: _data1138: _data1139: _data1140: _data1141: _data1142: _data1143: _data1144: _data1145: _data1146: _data1147: _data1148: _data1149: _data1150: _data1151: _data1152: _data1153: _data1154: _data1155: _data1156: _data1157: _data1158: _data1159: _data1160: _data1161: _data1162: _data1163: _data1164: _data1165: _data1166: _data1167: _data1168: _data1169: _data1170: _data1171: _data1172: _data1173: _data1174: _data1175: _data1176: _data1177: _data1178: _data1179: _data1180: _data1181: _data1182: _data1183: _data1184: _data1185: _data1186: _data1187: _data1188: _data1189: _data1190: _data1191: _data1192: _data1193: _data1194: _data1195: _data1196: _data1197: _data1198: _data1199: _data1200: _data1201: _data1202: _data1203: _data1204: _data1205: _data1206: _data1207: _data1208: _data1209: _data1210: _data1211: _data1212: _data1213: _data1214: _data1215: _data1216: _data1217: _data1218: _data1219: _data1220: _data1221: _data1222: _data1223: _data1224: _data1225: _data1226: _data1227: _data1228: _data1229: _data1230: _data1231: _data1232: _data1233: _data1234: _data1235: _data1236: _data1237: _data1238: _data1239: _data1240: _data1241: _data1242: _data1243: _data1244: _data1245: _data1246: _data1247: _data1248: _data1249: _data1250: _data1251: _data1252: _data1253: _data1254: _data1255: _data1256: _data1257: _data1258: _data1259: _data1260: _data1261: _data1262: _data1263: _data1264: _data1265: _data1266: _data1267: _data1268: _data1269: _data1270: _data1271: _data1272: _data1273: _data1274: _data1275: _data1276: _data1277: _data1278: _data1279: _data1280: _data1281: _data1282: _data1283: _data1284: _data1285: _data1286: _data1287: _data1288: _data1289: _data1290: _data1291: _data1292: _data1293: _data1294: _data1295: _data1296: _data1297: _data1298: _data1299: _data1300: _data1301: _data1302: _data1303: _data1304: _data1305: _data1306: _data1307: _data1308: _data1309: _data1310: _data1311: _data1312: _data1313: _data1314: _data1315: _data1316: _data1317: _data1318: _data1319: _data1320: _data1321: _data1322: _data1323: _data1324: _data1325: _data1326: _data1327: _data1328: _data1329: _data1330: _data1331: _data1332: _data1333: _data1334: _data1335: _data1336: _data1337: _data1338: _data1339: _data1340: _data1341: _data1342: _data1343: _data1344: _data1345: _data1346: _data1347: _data1348: _data1349: _data1350: _data1351: _data1352: _data1353: _data1354: _data1355: _data1356: _data1357: _data1358: _data1359: _data1360: _data1361: _data1362: _data1363: _data1364: _data1365: _data1366: _data1367: _data1368: _data1369: _data1370: _data1371: _data1372: _data1373: _data1374: _data1375: _data1376: _data1377: _data1
```

nhiều gói tin trong khoảng thời gian ngắn, máy chủ sẽ gửi một yêu cầu tạo lại *sequence number* mới thông qua *quasafe counter*. Hình 4.5 trình bày quá trình cập nhật và đồng bộ *sequence number* mới giữa gateway và máy chủ.

```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TAILSCALE
TERMINAL CONSOLE GATEWAY
-- Sequence number: 0
-- Server data frame constructed in 0.0900 ms
-- Publish succeeded
-- Sent server data frame to server in 4.9380 ms
-- Send data to server completed in 16.18 ms
[5/5] Waiting for ACK package...
-- Attack detected! Received signal from server to update sequence number
-- Updating sequence number 10.33 ms
-- Safe counter used for updating sequence: 61
-- Sequence number updated to 3
[1/5] Checking key...
-- Key is still valid
[2/5] Transmitting trigger signal to STM32...
-- Constructing frame for transmitting trigger signal to STM32...
-- Command: 4
-- Transmitting trigger signal to STM32
-- Transmit trigger signal to STM32 completed in 9.97 ms
[3/5] Frame processing...
-- Receiving data from STM32
-- Frame parsing completed in 0.34 ms
-- Frame parsing completed in 10.66 ms
[4/5] Construct and send data to server...
Safe counter current: 61
-- Constructing server data frame
-- Sequence number: 3
-- Server data frame constructed in 0.0990 ms
-- Publish succeeded
-- Sent server data frame to server in 1.6740 ms
-- Send data to server completed in 12.71 ms
[5/5] Waiting for ACK package...
-- Received ACK package from server in 386.15 ms
[NICE] Transmission completed!
Cập nhật sequence number mới

CONSOLE SERVER
Start Parsing Data Frame
-- Data frame has been rejected: Invalid sequence number: got 2083395783, expected 0
-- Processed handshake/ecdh in 4ms
-- Received ECDH Handshake message
Start Parsing Data Frame
-- Data frame has been rejected: Invalid sequence number: got 1417887245, expected 0
-- Processed handshake/ecdh in 4ms
-- Received ECDH Handshake message
Start Parsing Data Frame
-- Data frame has been rejected: Invalid sequence number: got 3606730241, expected 0
-- Processed handshake/ecdh in 2ms
-- Received ECDH Handshake message
Start Parsing Data Frame
-- Data frame has been rejected: Invalid sequence number: got 1297008668, expected 0
-- Sequence number failed 10 times. Resetting counter...
Server Secret Key type: string
-- Safe Counter: 61
-- Resetting sequence number to 3
Cập nhật sequence number mới
(*) Publish Signal to handshake-send/ecdh completed
-- Received ECDH Handshake message
Start Parsing Data Frame
-- Data frame has been rejected: Invalid sequence number: got 356330853, expected 3
-- Processed handshake/ecdh in 1ms
-- Received ECDH Handshake message
Start Parsing Data Frame
CONSOLE MQTT BROKER
npm start
[TOPIC_SUBSCRIBED] ESP32Client → handshake-send/ecdh
[X] [CLIENT_DISCONNECTED] ESP32Client
[✓] [CLIENT_CONNECTED] ESP32Client
[TOPIC_SUBSCRIBED] ESP32Client → handshake-send/ecdh
[X] [CLIENT_DISCONNECTED] ESP32Client
[✓] [CLIENT_CONNECTED] ESP32Client
[TOPIC_SUBSCRIBED] ESP32Client → handshake-send/ecdh
[X] [CLIENT_DISCONNECTED] ESP32Client
[✓] [CLIENT_CONNECTED] ESP32Client
[TOPIC_SUBSCRIBED] ESP32Client → handshake-send/ecdh
[X] [CLIENT_DISCONNECTED] ESP32Client
[✓] [CLIENT_CONNECTED] ESP32Client
[TOPIC_SUBSCRIBED] ESP32Client → handshake-send/ecdh
[X] [CLIENT_DISCONNECTED] ESP32Client
[✓] [CLIENT_CONNECTED] ESP32Client
[TOPIC_SUBSCRIBED] ESP32Client → handshake-send/ecdh
[X] [CLIENT_DISCONNECTED] ESP32Client
[✓] [CLIENT_CONNECTED] GatewayClient

```

Hình 4.5: Quá trình thực hiện cơ chế *safe counter*

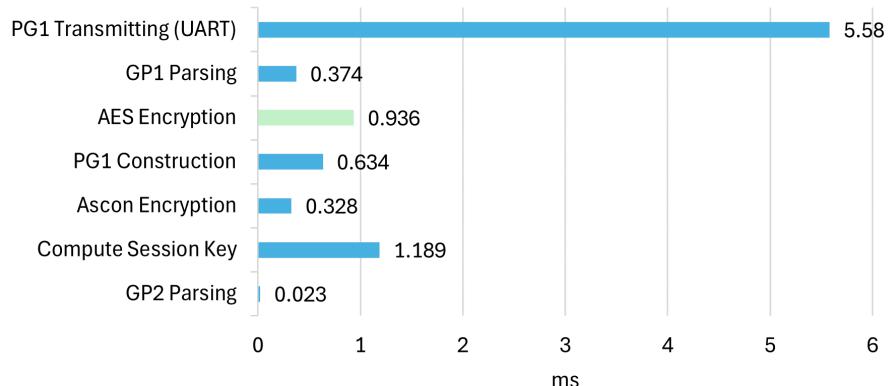
4.2.2 Đánh giá

Hệ thống được đánh giá dựa trên quá trình trao đổi dữ liệu liên tục trong 120 phút, với các kết quả được trình bày trong hình 4.6, 4.7 và hình 4.8. Kết quả này tập trung vào hiệu suất của hai thiết bị công suất thấp ở lớp nhận thức và lớp mạng. Đầu tiên là đánh giá với STM32F411VET6 ở lớp nhận thức. STM32 thực hiện việc cấu hình khung truyền PG1 với thời gian trung bình là 0.634 ms. Truyền khung PG1 bằng UART lên gateway có thời gian trung bình là 5.58 ms. Thuật toán Ascon-128a có thời gian mã hóa trung bình là 0.328 ms, cho thấy hiệu năng vượt trội so với thuật toán AES, vốn được triển khai tương tự bằng thư viện X-CUBE-CRYPTOLIB và có thời gian mã hóa trung bình là 0.936 ms trên lõi ARM Cortex-M4. Quá trình phân giải khung truyền GP1 từ gateway có thời gian trung bình là 0.374 ms; khung truyền GP2 có thời gian trung bình là 0.023 ms. Sự khác biệt lớn về thời gian trong quá trình phân giải phần lớn là do việc giải mã khóa và tính toán thẻ xác thực ở khung truyền GP1.

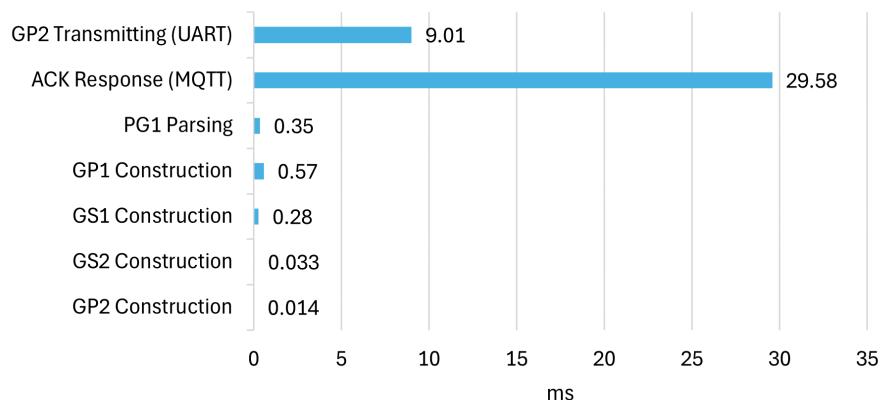
Ở lớp mạng, ESP32 thực hiện việc phân giải PG1 từ STM32 trong thời gian trung bình 0.35 ms. Cấu hình khung truyền GP1 có thời gian trung bình là 0.57 ms, trong khi đó thời gian cấu hình GP2 có thời gian trung bình là 0.014 ms. Thời gian truyền khung truyền GP2 dùng UART xuống lớp nhận thức có thời gian trung bình là 9.01 ms. Quá trình trao đổi khóa bao gồm việc tạo cặp khóa cá nhân (private key)/khóa công khai

(public key) và tính toán khóa bí mật có thời gian trung bình lần lượt là 335.08 ms và 335.46 ms. Tuy nhiên, việc trao đổi khóa chỉ được thực hiện sau một khoảng thời gian cố định, vì vậy nếu xét về tổng thể sẽ không ảnh hưởng nhiều đến toàn bộ hiệu suất của hệ thống. Thời gian trung bình cấu hình khung truyền GS1 và GS2 lần lượt là 0.28 ms và 0.033 ms. Mặc dù GS2 có nhiều trường cần cấu hình hơn GS1, nhưng GS2 chỉ thực hiện việc tách trường dữ liệu và thẻ xác thực từ khung PG1 mà không thực hiện tính toán nào thêm. Vì thế, việc cấu hình GS2 nhanh hơn nhiều so với GS1, khi mà GS1 cần phải tính toán thẻ xác thực trong quá trình cấu hình. Thời gian tính toán khóa phiên ở mỗi phiên truyền dữ liệu có thời gian trung bình là 1.189 ms.

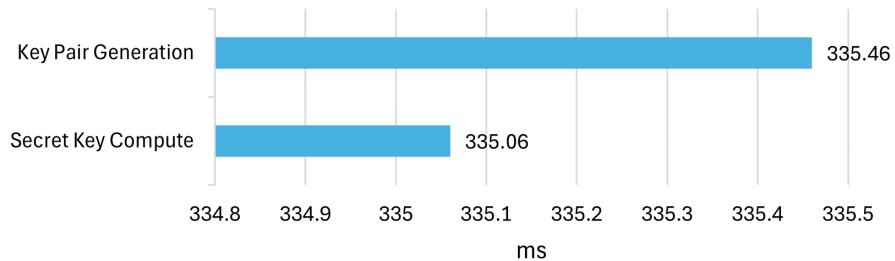
Lệnh ACK phản hồi từ máy chủ, báo hiệu phiên truyền thành công, có độ trễ trung bình là 29.58 ms, tính từ lúc khung truyền GS tới máy chủ và gateway nhận được ACK. Tuy nhiên, độ trễ này là không cố định do phụ thuộc phần lớn vào tốc độ mạng cũng như hiệu suất của máy chủ; vì vậy, nếu triển khai ở một số hệ thống khác thì độ trễ sẽ có thể khác. Tổng thời gian truyền được tính từ lúc STM32 nhận được tín hiệu kích hoạt từ gateway cho đến khi gateway nhận được ACK là 51.09 ms.



Hình 4.6: Đánh giá hiệu suất thực thi của STM32F411VET6



Hình 4.7: Đánh giá hiệu suất thực thi của ESP32



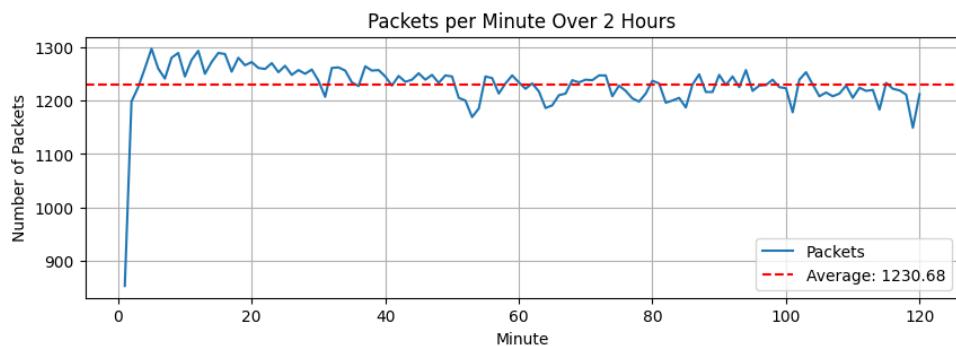
Hình 4.8: Thời gian thực thi quá trình trao đổi khóa trên ESP32

Trong khoảng thời gian 120 phút thực nghiệm, gateway đã thực hiện truyền tổng cộng 147,682 gói tin. Một gói tin truyền thành công được định nghĩa là gói tin được mã hóa đúng cách và máy chủ có thể giải mã đúng gói tin đó. Tỷ lệ truyền gói tin thành công (Packet Delivery Ratio - PDR) là một thông số quan trọng trong việc đánh giá mức độ toàn vẹn của hệ thống được tính toán sau mỗi phút truyền dữ liệu, được tính toán với công thức sau:

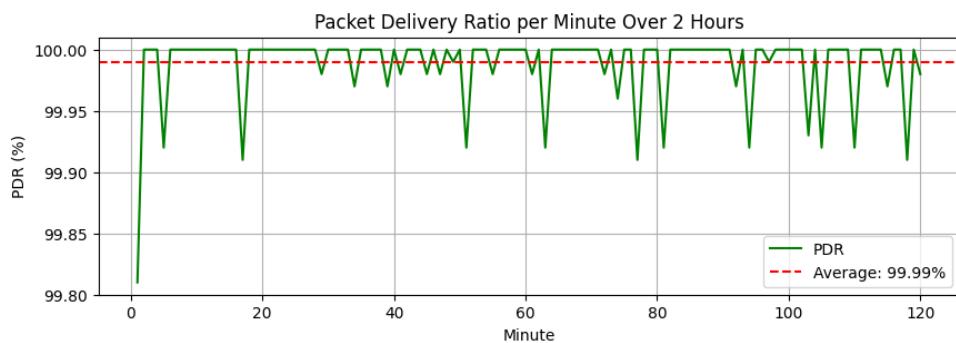
$$PDR = \frac{N_{\text{successful}}}{N_{\text{total}}} \times 100\%$$

Trong đó: $N_{\text{successful}}$ là số gói tin được nhận thành công tại đích (gói/phút), N_{total} là tổng số gói tin được gửi từ nguồn (gói/phút).

Biểu đồ trong Hình 4.9 minh họa số lượng gói tin được truyền trong mỗi phút trong suốt 120 phút hoạt động liên tục, với số lượng trung bình đạt 1231 gói/phút. Biểu đồ trong hình 4.10 cho thấy tỷ lệ truyền gói tin mỗi phút cũng trong tổng thời gian 120 phút. Kết quả cho thấy tỷ lệ 100% được duy trì trong phần lớn thời gian truyền, trong các khoảng thời gian còn lại, tỷ lệ được duy trì ổn định trong khoảng từ 99.8% cho đến 99.975%. Tỷ lệ truyền gói tin trung bình là 99.99%, cho thấy mức độ ổn định và toàn vẹn của hệ thống trong suốt quá trình thực nghiệm. Bảng 4.2 tóm tắt các chỉ số hiệu năng, bao gồm tỷ lệ truyền thành công trung bình, số gói tin trên mỗi khoảng thời gian, và thời gian truyền cho mỗi gói tin.



Hình 4.9: Số lượng gói tin truyền mỗi phút



Hình 4.10: Tỷ lệ truyền gói tin mỗi phút

Bảng 4.2: Bảng đánh giá hiệu năng trong 120 phút

Thông số	Giá trị
Tổng số gói tin truyền	147,682 gói tin
Thời gian truyền trung bình	51.09 ms
Số gói tin truyền trung bình mỗi phút	1231 gói tin
Tỷ lệ truyền thành công (PDR)	99.99%

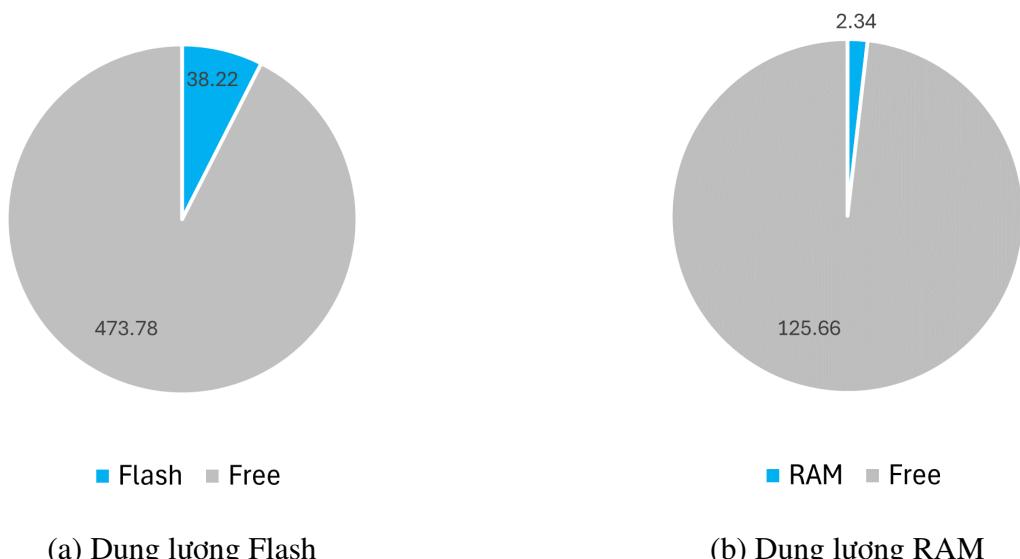
Dung lượng bộ nhớ của hệ thống IoT trên ESP32 và STM32 được cung cấp bởi công cụ lập trình PlatformIO và Cube IDE, hiển thị tổng số byte RAM và FLASH được sử dụng trong quá trình nạp firmware. Chi tiết được trình bày trong bảng 4.3.

Bảng 4.3: Dung lượng bộ nhớ RAM và FLASH

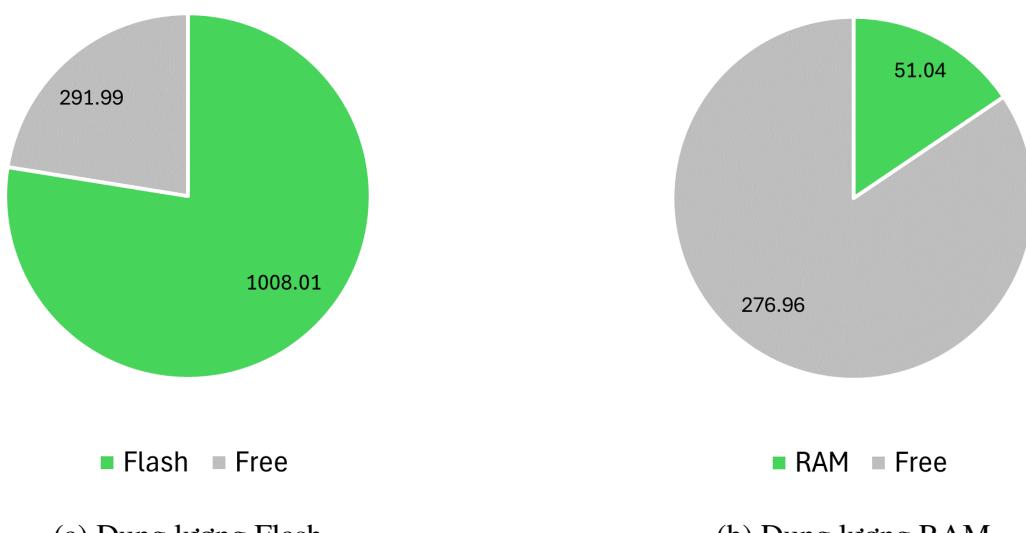
Thiết bị	RAM (kB)	Flash (kB)
STM32F411VET6	2.34	38.22
ESP32-WROOM-32	51.04	1008.01

Trên STM32F411VET6, nhiệm vụ chính là thu thập và mã hóa dữ liệu, với bộ nhớ chủ yếu dành cho thuật toán Ascon-128a, nhưng tổng thể chiếm dụng không đáng kể. Ngược lại, gateway ESP32 thực hiện nhiều chức năng hệ thống như cấu hình khung

truyền và quản lý mạng, thực hiện quá trình trao đổi và quản lý khóa, dẫn đến dung lượng FLASH chiếm nhiều do yêu cầu lập trình đa tác vụ. Hình 4.11 và 4.12 trình bày mức sử dụng bộ nhớ trên hai nền tảng. Cụ thể, trên STM32, bộ nhớ FLASH còn trống 473.78 kB (92.53%) và RAM còn trống 125.66 kB (98.17%), cho thấy dung lượng sử dụng là rất thấp. Trong khi đó, trên ESP32, bộ nhớ FLASH chỉ còn trống 291.99 kB (22.46%) và RAM còn trống 276.96 kB (84.43%). Như vậy, có thể thấy kiến trúc hệ thống này chủ yếu yêu cầu bộ nhớ FLASH lớn trên gateway ESP32, do cần tích hợp nhiều chức năng hệ thống, trong khi thiết bị cảm biến STM32 hoạt động nhẹ, chủ yếu tập trung vào việc mã hóa dữ liệu với mức tiêu thụ bộ nhớ tối thiểu.



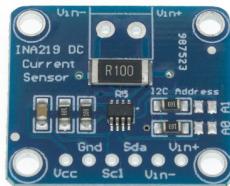
Hình 4.11: Dung lượng tiêu thụ của STM32F411VET6



Hình 4.12: Dung lượng tiêu thụ của ESP32-WROOM-32

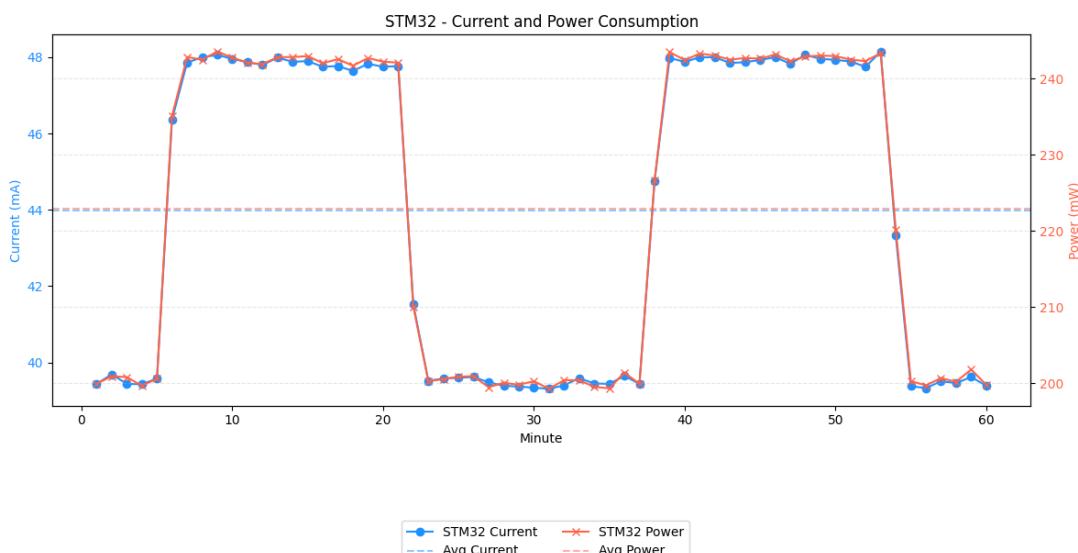
4.2.3 Phân tích công suất tiêu thụ

Quá trình đo công suất tiêu thụ được thực hiện trên hệ thống phần cứng bao gồm vi điều khiển STM32F411VET6 ở lớp nhận thức và module ESP32 ở lớp mạng. Cảm biến INA219 được sử dụng để phục vụ quá trình đo đặc của hệ thống. Đây là một cảm biến chuyên dụng sử dụng giao thức I2C cho phép đo dòng điện, điện áp và tính toán công suất tiêu thụ với độ chính xác cao. Cả hệ thống được cấp nguồn chung là 5V và đo liên tục trong 60 phút. Quá trình trao đổi khóa cũng được cập nhật lại là trao đổi khóa sau mỗi 10 gói tin thành công. Với cấu hình thử nghiệm như vậy sẽ cho ra cái nhìn khách quan hơn về mức tiêu thụ năng lượng của hệ thống. Trong suốt quá trình đo, các thông số về dòng điện (mA) và công suất tiêu thụ (mW) được ghi lại mỗi phút. Các kết quả thu được phản ánh trực tiếp mức độ tiêu thụ năng lượng của từng thành phần cũng như toàn hệ thống trong suốt thời gian hoạt động.



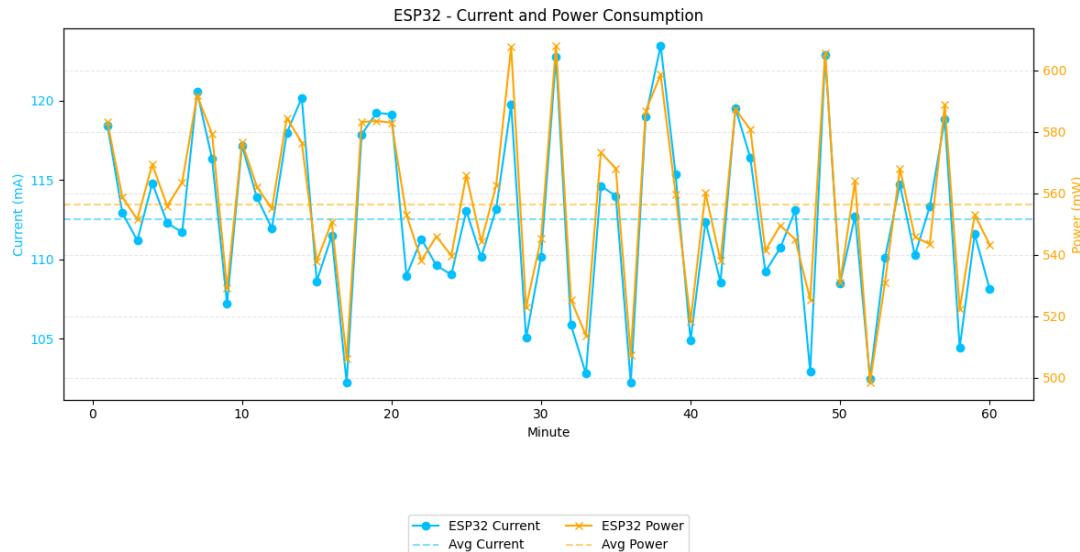
Hình 4.13: Cảm biến dòng INA219

Hình 4.14 minh họa dòng điện và công suất tiêu thụ của STM32 trong suốt 60 phút. Dòng điện dao động trong khoảng từ 40 đến 48 mA, với giá trị trung bình đạt 43.99 mA. Công suất tiêu thụ của STM32 dao động từ 200 đến 240 mW, với mức trung bình trong một giờ là 222.97 mW.



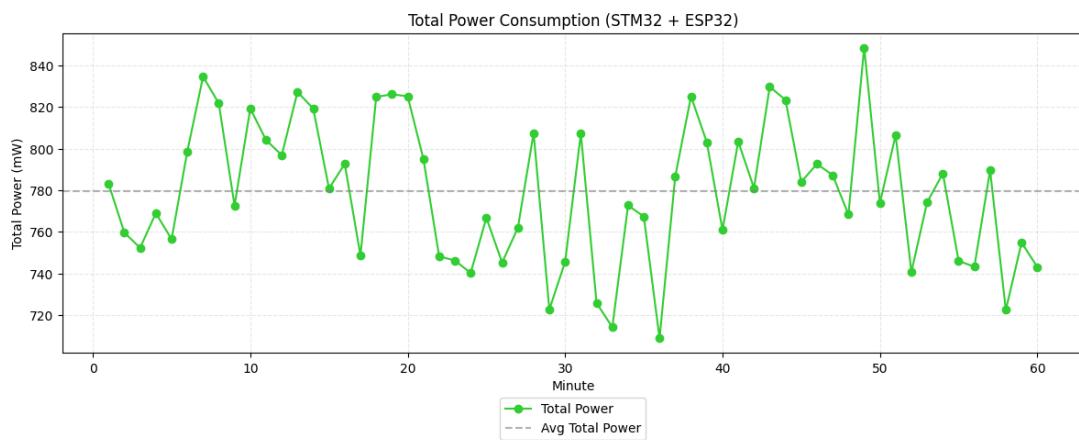
Hình 4.14: Mức tiêu thụ năng lượng của STM32F411VET6

Tương tự, hình 4.15 thể hiện đặc tính tiêu thụ năng lượng của ESP32 IoT Gateway. ESP32 có dòng điện dao động từ 100 đến 125 mA, với giá trị trung bình là 112.52 mA. Công suất tiêu thụ tương ứng dao động trong khoảng 500 đến 600 mW, với giá trị trung bình đạt 556.46 mW. Có thể nhận thấy ESP32 tiêu thụ năng lượng cao hơn đáng kể so với STM32, do phải xử lý các thuật toán phức tạp hơn, duy trì kết nối Wi-Fi và truyền dữ liệu liên tục tới máy chủ qua giao thức MQTT.



Hình 4.15: Mức tiêu thụ năng lượng của ESP32 Gateway

Cuối cùng, hình 4.16 trình bày công suất tiêu thụ tổng của toàn hệ thống. Kết quả ghi nhận mức tiêu thụ trung bình là 779.43 mW.



Hình 4.16: Mức tiêu thụ năng lượng của toàn bộ hệ thống

4.2.4 Phân tích thông lượng và hiệu suất tải dữ liệu

Thông lượng và hiệu suất tải dữ liệu là hai thông số quan trọng trong việc đánh giá hệ thống. Việc sử dụng kiến trúc khung truyền với các trường dữ liệu bổ sung sẽ tồn

thêm chi phí truyền tải, kích thước mỗi gói tin sẽ tăng so với các phương pháp đơn giản hơn. Phần này sẽ tập trung vào phân tích thông lượng và hiệu suất tải dữ liệu để đánh giá một số hạn chế của hệ thống. Đồng thời, đề xuất một số phương pháp có thể giúp cải thiện hệ thống trong tương lai.

Khung truyền GS2 sẽ được sử dụng trong việc đánh giá thông lượng cũng như hiệu suất tải dữ liệu của hệ thống. Đây là khung truyền được sử dụng với tần suất lớn nhất trong suốt quá trình truyền tải dữ liệu. Khung truyền GS2 là khung truyền đảm nhận việc truyền dữ liệu từ lớp nhận thức đến máy chủ bằng giao thức MQTT. Thông qua việc phân tích khung truyền GS2, có thể xác định chính xác nhất thông lượng và hiệu suất tải dữ liệu của hệ thống. Tổng kích thước của khung truyền GS2 là tổng kích thước của tất cả các trường dữ liệu được tính toán như sau:

$$\begin{aligned} \text{Size}_{GS2} &= \text{SOF (2)} + \text{ID (4)} + \text{Packet Type (1)} + \text{Seq. Number (4)} \\ &\quad + \text{Nonce (16)} + \text{Payload Length (1)} + \text{Encrypted Payload (3)} \\ &\quad + \text{Auth Tag (16)} + \text{EOF (2)} \\ &= 49 \text{ bytes} \end{aligned}$$

Trong hệ thống đã triển khai, lớp nhận thức thu thập và truyền tải 3 byte dữ liệu, từ đó tham số *Encrypted Payload* cũng có kích thước là 3 byte. Một điều chú ý đó là tổng kích thước của khung truyền GS2 sẽ có sự thay đổi dựa trên kích thước dữ liệu mà lớp nhận thức muốn truyền tải. Từ kết quả đó được ở phần 4.2.2, trong 120 phút, hệ thống đã truyền tải tổng cộng 147,682 gói tin. Tuy nhiên, vì phần này chỉ đánh giá thông lượng của khung truyền GS2, 120 khung truyền GS1 trong việc trao đổi khóa sẽ bị loại bỏ trong tính toán. Đến đây tổng số gói tin sử dụng khung truyền GS2 sẽ được tính là:

$$\text{Total}_{GS2} = 147,682 - 120 = 147,562 \text{ (gói tin)}$$

Tổng kích thước mà hệ thống đã truyền tải sẽ được tính như sau:

$$\text{Total}_{size} = 147,562 \times 49 = 7,230,538 \text{ (byte)}$$

Từ đó, ta có thể tính toán thông lượng của hệ thống bằng công thức:

$$Throughput = \left(\frac{\text{Data size}}{\text{Transmission Time}} \right) \times 8 = \left(\frac{7,230,538}{120 \times 60} \right) \times 8 \approx 8,033.93 \text{ bps}$$

Trong đánh giá này, mỗi khung truyền GS2 có tổng kích thước là 49 byte, tuy nhiên chỉ có 3 byte dữ liệu cần thiết, phần còn lại chủ yếu là các trường bổ sung nhằm đảm bảo

tính bảo mật của hệ thống, dẫn đến một tham số được đưa ra đó là hiệu suất tải (*payload efficiency*) khá nhỏ là $3/49 \approx 6.12\%$. Con số này nói lên rằng chỉ có 6.12% dữ liệu được truyền là dữ liệu hữu ích, trong khi phần còn lại là chi phí truyền dẫn khác không có ý nghĩa đối với người dùng cuối. Vì vậy, mặc dù thông lượng của hệ thống khá cao nhưng thực tế thì thông lượng hữu ích (goodput) lại thấp hơn nhiều:

$$Goodput = \left(\frac{147,562 \times 3}{120 \times 60} \right) \times 8 \approx 491.87 \text{ bps}$$

Từ những phân tích trên, có thể thấy chi phí truyền tải khi áp dụng kiến trúc khung truyền là không nhỏ. Với hiệu suất tải thấp sẽ dẫn đến việc tiêu tốn băng thông, băng thông sẽ không được sử dụng một cách tối ưu. Tuy nhiên, hiệu suất tải sẽ được cải thiện đáng kể nếu hệ thống truyền nhiều dữ liệu, tức là kích thước tải lớn hơn. Bảng 4.4 trình bày hiệu suất tải với các kích thước tải khác nhau trên cùng một khung truyền dữ liệu, có thể thấy thay vì truyền 3 byte dữ liệu như đề tài đã thực hiện, nếu ứng dụng kiến trúc này vào các hệ thống cần truyền nhiều dữ liệu đồng thời, việc sử dụng băng thông sẽ được tối ưu hơn rất nhiều.

Bảng 4.4: Hiệu suất tải với các kích thước tải khác nhau

Payload Length	Total Frame Size	Efficiency
3 bytes	49 bytes	6.12%
10 bytes	56 bytes	17.85%
20 bytes	66 bytes	30.30%
50 bytes	96 bytes	52.08%

4.2.5 Phân tích khả năng bảo mật của hệ thống

Phần này trình bày đánh giá chi tiết về mức độ an toàn của hệ thống IoT được đề xuất, thông qua việc phân tích và nhận xét khả năng kháng cự của nó trước một số dạng tấn công phổ biến hiện nay.

Nhận xét 1: Hệ thống có khả năng chống lại các cuộc tấn công từ bên ngoài như Man-in-the-Middle

- Trong kiến trúc đề xuất, tất cả các lớp trong mô hình IoT bốn lớp đều được mã hóa bằng thuật toán mã hóa xác thực Ascon-128a. Nhờ đó, dù kẻ tấn công có xâm nhập và đánh cắp dữ liệu tại bất kỳ lớp nào trong quá trình truyền giữa, dữ liệu vẫn không thể bị giải mã hay sử dụng trái phép. Cơ chế này giúp đảm bảo tính bảo mật và toàn vẹn của thông tin người dùng trong toàn bộ hệ thống.

Nhận xét 2: Cơ chế khung truyền được thiết kế để ngăn chặn các thiết bị không hợp

lệ giao tiếp với hệ thống.

- Với thiết kế khung truyền dữ liệu, đảm bảo rằng chỉ có những thiết bị cùng một hệ sinh thái mới có thể giao tiếp và truyền dữ liệu cho nhau. Các thiết bị khác ở bên ngoài sẽ không có khả năng gửi dữ liệu vì không khớp ở các trường như SOF, Identifier ID và EOF, đây vốn là những trường độc nhất được cấu hình riêng cho mỗi thiết bị.

Nhận xét 3: Cơ chế khung truyền đảm bảo tính toàn vẹn của dữ liệu.

- Trước khi truyền, dữ liệu luôn được cấu trúc lại thành khung truyền bao gồm các trường xác thực. Khi một lớp trong hệ thống tiếp nhận khung truyền, dữ liệu sẽ được kiểm tra thông qua hàm phân giải nhằm xác thực tính hợp lệ. Cơ chế này giúp phát hiện và loại bỏ các sai sót trong quá trình truyền, đảm bảo rằng dữ liệu đến đích luôn chính xác. Điều này được thể hiện thông qua chỉ số PDR đạt 99.99% trong quá trình thử nghiệm.

Nhận xét 4: Quá trình trao đổi khóa được đảm bảo an toàn nhờ tích hợp thẻ xác thực (Auth Tag) trong khung truyền.

- Để bảo vệ quá trình trao đổi khóa, khóa công khai được truyền với khung truyền được tích hợp 16 byte Auth Tag. Với sự kết hợp của Auth Tag, đảm bảo rằng kể cả khi có kẻ tấn công từ bên ngoài muốn gửi khóa công khai giả nhằm phá hoại việc tính toán khóa bí mật, hệ thống vẫn có thể phát hiện và từ chối các khóa giả ngay lập tức.

Nhận xét 5: Thuật toán trao đổi khóa ECDH đảm bảo mức độ bảo mật theo khuyến nghị của NIST.

- Quá trình trao đổi khóa dùng thuật toán ECDH sử dụng đường cong *sect163k1* với mức bảo mật là 80-bit. Mức bảo mật này đáp ứng được khuyến nghị của NIST về mức độ bảo mật [11]. Hơn nữa, khóa bảo mật sẽ được cập nhật định kỳ, khiến các cuộc tấn brute-force vào khóa bí mật trở nên không khả thi.

Nhận xét 6: Hệ thống có khả năng phát hiện và ngăn chặn các cuộc tấn công phát lại (replay attack).

- Với 4 byte *sequence number* được tích hợp trong khung truyền GS2, đảm bảo rằng máy chủ có thể phát hiện và từ chối bất kỳ gói tin nào bị gửi lại do không trùng khớp *sequence number*. Nhờ đó, các cuộc tấn công phát lại sẽ không thể làm gián đoạn hệ thống.

Nhận xét 7: Cơ chế safe counter giúp chống lại tấn công brute-force vào sequence number của hệ thống.

- Một kịch bản có thể xảy ra đó là kẻ tấn công cố gắng brute-force toàn bộ 4 byte *sequence number* để có thể thành công gửi gói tin vào máy chủ. Vì vậy, cơ chế *safe counter* được thiết lập để có thể các cuộc tấn công trên. Nếu máy chủ từ chối một số lượng gói tin nhất định, cả gateway và máy chủ sẽ sử dụng *safe counter* để cập nhật lại *sequence number* mới một cách đồng bộ. Với cơ chế này, tỉ lệ brute-force thành công gần như là không thể xảy ra.

Nhận xét 8: Hệ thống đã được kiểm nghiệm thực tế với lượng lớn dữ liệu giả mạo và cho thấy khả năng phòng thủ cao.

- Hệ thống đã được kiểm tra bằng cách gửi 70,000 gói tin giả mạo. Rõ ràng rằng nếu kẻ tấn công không biết được kiến trúc của khung truyền dữ liệu, họ sẽ không thể gửi dữ liệu tới hệ thống. Vì vậy, trong thử nghiệm, giả định rằng kẻ tấn công biết trước khung truyền dữ liệu cho từng giao tiếp. Kết quả vẫn cho thấy máy chủ đã từ chối toàn bộ 70,000 gói tin giả mạo, đạt tỉ lệ 100%, chứng minh khả năng bảo vệ của hệ thống trước các cuộc tấn công trên.

Nhận xét 9: Kiến trúc đề xuất triển khai phù hợp với các thiết bị IoT hiện nay.

- Đề tài sử dụng các phần cứng tầm trung như STM32F411VET6 và ESP32, xung nhịp của STM32 cũng được giới hạn ở mức mặc định là 16MHz. Tuy nhiên, với các phân tích về sử dụng bộ nhớ cũng như thời gian xử lý dữ liệu ở phần 4.2.2, có thể thấy kiến trúc IoT đề tài đề xuất hoàn toàn phù hợp để triển khai trên các thiết bị IoT có khả năng tính toán thấp mà vẫn cung cấp tính năng bảo mật tốt. Các dòng vi điều khiển cao cấp hơn như STM32F7 hoặc STM32H7, kết hợp với Raspberry Pi như một IoT Gateway thì hiệu suất thậm chí sẽ vượt trội hơn nhưng các phần cứng tầm trung là hoàn toàn đủ để triển khai tốt kiến trúc này.

Chương 5: Kết luận

Danh mục công trình của tác giả

1. Được chấp nhận trình bày tại Hội nghị Quốc tế IAAA 2025 (International Conference on Intelligent Aerial Access and Applications), tổ chức tại Hà Nội từ ngày 16 đến 18 tháng 7.

Tài liệu tham khảo

Tiếng Anh

- [1] Ahmad, Ijaz. "Adaptive Lightweight Security for Performance Efficiency in Critical Healthcare Monitoring". In: *18th International Symposium on Medical Information and Communication Technology (ISMICT)* (2024).
- [2] Brown, Daniel R. L. "Standards for Efficient Cryptography SEC 2: Recommended Elliptic Curve Domain Parameters. Version 2.0". In: (2010).
- [3] Coutinho, Rodolfo W. L. and Boukerche, Azzedine. "Modeling and Analysis of a Shared Edge Caching System for Connected Cars and Industrial IoT-Based Applications". In: *IEEE Transactions on Industrial Informatics* (2003).
- [4] Delvai, M., Eisenmann, U., and Elmenreich, W. "Intelligent UART Module for Real-Time Applications". In: *WISES Conference Proceedings*. 2003.
- [5] Dobraunig, C., Eichlseder, M., and Mendel, F. "ASCON v1.2: Lightweight Authenticated Encryption and Hashing". In: *Journal of Cryptology* (2022).
- [6] Ekwueme, C., Adam, I., and Dwivedi, A. "EAI Endorsed Transactions on Internet of Things". In: *Proceedings of IEEE ICIINFS Conference*. 2024.
- [7] El-Hadey, Mohamed. "RECO-ASCON: Reconfigurable ASCON Hash Functions for IoT Applications". In: *Integration* 93 (2023).
- [8] Hameed, Sufian, Khan, Faraz Idris, and Hameed, Bilal. "Understanding Security Requirements and Challenges in Internet of Things (IoT): A Review". In: *Journal of Computer Networks and Communications* (2022).
- [9] Hou, Jia-Li and Yeh, Kuo-Hui. "A secure data transmission framework for IoT enabled healthcare". In: *Heliyon* 10.16 (2024).
- [10] Hou, Jia-Li and Yeh, Kuo-Hui. "Novel Authentication Schemes for IoT Based Healthcare Systems". In: *International Journal of Distributed Sensor Networks* (2015).
- [11] Mahto and Yadav. "RSA and ECC: A Comparative Analysis". In: *International Journal of Applied Engineering Research* 10.12 (2017), pp. 9053–9061.
- [12] Online. *ASCON Hashing and XOF (eXtendable-Output Functions)*. URL: <https://asecuritysite.com/ascon/ascon02> (visited on 01/06/2025).
- [13] Online. *Cryptanalysis*. URL: <https://en.wikipedia.org/wiki/Cryptanalysis> (visited on 02/06/2025).

- [14] Online. *LaTeX/Floats, Figures and Captions*. URL: http://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions (visited on 06/06/2015).
- [15] Online. *LaTeX/Source Code Listings*. URL: http://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings (visited on 06/06/2015).
- [16] Online. *LaTeX/Tables*. URL: <http://en.wikibooks.org/wiki/LaTeX/Tables> (visited on 06/06/2015).
- [17] Online. *Man-in-the-Middle*. URL: <https://viblo.asia/p/man-in-the-middle-attack-mitm-aWj53LMbK6m> (visited on 02/06/2025).
- [18] Online. *Man in the Middle attack in Diffie-Hellman Key Exchange*. URL: <https://www.geeksforgeeks.org/man-in-the-middle-attack-in-diffie-hellman-key-exchange/> (visited on 02/06/2025).
- [19] Online. *Password cracking*. URL: <https://www.distributed.net/RC5> (visited on 01/06/2025).
- [20] Online. *Power Side-channel Attack on software implemented AES using chip-whisperer NANO*. URL: <https://labs.dese.iisc.ac.in/embeddedlab/power-side-channel-attack-on-software-implemented-aes/> (visited on 02/06/2025).
- [21] Online. *Project RC5*. URL: https://en.wikipedia.org/wiki/Password_cracking (visited on 01/06/2025).
- [22] Online. *Related-key Attack*. URL: https://en.wikipedia.org/wiki/Related-key_attack (visited on 02/06/2025).
- [23] Online. *Replay Attack*. URL: https://vi.wikipedia.org/wiki/T%E1%BA%A5n_c%C3%B4ng_ph%C3%A1t_1%E1%BA%A1i (visited on 02/06/2025).
- [24] Online. *Side-channel attack*. URL: https://en.wikipedia.org/wiki/Side-channel_attack (visited on 02/06/2025).
- [25] Online. *What is a brute force attack?* URL: <https://fieldeffect.com/blog/what-is-a-brute-force-attack> (visited on 01/06/2025).
- [26] Peretti, Giulio, Lakkundit, Vishwas, and Zorzi, Michele. “BlinkToSCoAP: An End-to-End Security Framework for the Internet of Things”. In: *Future Information Security Workshop, COMSNETS*. 2015.
- [27] Rajmohan, Tanusan, Nguyen, Phu H., and Ferry, Nicolas. “A decade of research on patterns and architectures for IoT security”. In: *Cybersecurity* (2022).

- [28] Satapathy, U. et al. “An ECC-Based Lightweight Authentication Protocol for Mobile Phones in Smart Homes”. In: *Proceedings of IEEE ICIINFS Conference*. 2018.
- [29] Shashi, Rekha et al. “Study of security issues and solutions in Internet of Things (IoT)”. In: *Materials Today: Proceedings* (2021).
- [30] Yousuf, T. et al. “Internet of Things (IoT) Security: Current Status, Challenges and Countermeasures”. In: *International Journal for Information Security Research* (2015).