# MỞ ĐẦU

**Cơ sở khoa học và thực tiễn của đề tài:**

Ngày nay, chúng ta có thể dễ dàng bắt gặp thuật ngữ IoT ở bất cứ nơi nào. Thuật ngữ này là viết tắt của cụm từ “***Internet of Things***”, để chỉ những vật được kết nối internet và có khả năng trao đổi dữ liệu. IoT trong những năm gần đây rất phổ biến, trong năm 2017 nó đã được thêm vào từ điển Oxford [11] dưới dạng một danh từ. Đặc điểm chung của những thiết bị IoT là kích thước nhỏ gọn và năng lượng tiêu thụ thấp. Ví dụ: cảm biến môi trường, cảm biến y tế … Phần lớn các thiết bị IoT đang gặp phải vấn đề về bảo mật.

Trong năm 2004, thuật ngữ “***Lightweight cryptography***” được đưa ra thảo luận tại nhiều hội nghị. Thuật ngữ này dùng để chỉ những hệ mật “*nhẹ*” có khả năng cài đặt trên các thiết bị bị giới hạn bởi năng lượng tiêu thụ và khả năng lưu trữ. Như vậy, mật mã nhẹ rất phù hợp để áp dụng vào bảo mật cho các thiết bị IoT. Do đó, việc phát triển nhanh và mạnh của *internet of things* cũng chính là động lực để thúc đẩy mật mã nhẹ phát triển. Bên cạnh IoT, có rất nhiều thiết bị bị giới hạn cũng cần đến sự bảo mật ví dụ như các thiết bị trong lĩnh vực quân sự, y tế, nông nghiệp … Nên tiềm năng của mật mã nhẹ là rất lớn.

Trong luận văn này của mình, tác giả trình bày những nghiên cứu về mật mã nhẹ và đi sâu vào một nhánh con là hàm băm nhẹ. Ngoài ra, tác giả sẽ trình bày những tìm hiểu của mình về một hệ mật cụ thể là PRESENT và hàm băm tương ứng của nó theo công thức Davies Mayer và cấu trúc Merkle Damgard.

**Nội dung đề tài và những vấn đề cần giải quyết:**

***Hướng nghiên cứu***

* Nghiên cứu một cách tổng quan nhất về mật mã nhẹ.
* Đi sâu nghiên cứu một nhánh nhỏ trong mật mã nhẹ là hàm băm.
* Nghiên cứu một hệ mật cụ thể là PRESENT và hàm băm tương ứng của nó theo công thức Davies Mayer.
* Xây dựng ứng dụng xác thực mật khẩu trong chương trình trên điện thoại thông minh.

***Nội dung nghiên cứu***

Ngoài phần mở đầu và kết luận, luận văn được trình bày theo 3 chương với các nội dung như sau:

Chương 1: Tổng quan về mật mã nhẹ

Trong chương này, tác giả giới thiệu các khái niệm cơ bản về mật mã nhẹ, động lực thúc đẩy mật mã nhẹ phát triển và một vài chiến lược trong thiết kế mật mã nhẹ.

Chương 2: Hàm băm nhẹ

Chương này là chương trọng tâm của luận văn, phần đầu tiên của chương, tác giả trình bày các khái niệm cơ bản về hàm băm nhẹ, các yêu cầu cần đảm bảo đối với một hàm băm. Phần kế, tác giả trình bày một số hàm băm nhẹ, thách thức mà những hàm băm này đang gặp phải. Phần cuối cùng của chương, tác giả trình bày những tìm hiểu của mình về hệ mật PRESENT và hàm băm của hệ mật này theo công thức Davies Mayer và cấu trúc Merkle Damgard.

Chương 3: Thực nghiệm

Xây dựng chương trình xác thực mật khẩu cho ứng dụng trên điện thoại thông minh nền tảng Android.

***Kết quả đạt được***

Sau thời gian tìm hiểu và nghiên cứu, luận văn đã đạt được một số kết quả ban đầu. Đó là việc nghiên cứu một cách tổng quan nhất về mật mã nhẹ và hàm băm nhẹ. Sau đó là việc tập trung nghiên cứu hệ mật PRESENT, hàm băm của PRESENT theo công thức Davies Mayer và cấu trúc Merkle Damgard. Với kết quả hiện tại, tác giả đã làm chủ được mã nguồn để có thể áp dụng vào các ứng dụng của mình.

# Chương 1: TỔNG QUAN VỀ MẬT MÃ NHẸ

* 1. **Mật mã nhẹ**

Như tác giả đã trình bày ở phần mở đầu, mật mã nhẹ ra đời hướng tới các thiết bị bị giới hạn bởi năng lượng tiêu thụ và không gian lưu trữ do đó mục tiêu của các hệ mật nhẹ là vừa “***bảo mật***”, vừa “***chi phí thấp***”, vừa “***hiệu suất cao***”. Ta có thể dễ thấy, ba yếu tố này không thể cùng đi lên nên việc duy nhất có thể làm là cân bằng ba yếu tố này trong trường hợp áp dụng cụ thể.

* + 1. **Khái niệm mật mã nhẹ**

Không có một ranh giới rõ ràng nào để phân biệt sự nhẹ của một hệ mật với các hệ mật thông thường [8]. Mật mã nhẹ là một nhánh nghiên cứu con của mật mã hướng tới việc tối ưu sự tinh gọn của hệ mật để có thể cài đặt và chạy hiệu quả trên các thiết bị vô cùng nhỏ bé và bị giới hạn bởi năng lượng tiêu thụ và khả năng lưu trữ. Ví dụ như các thẻ chip, thẻ từ dùng gắn trên các bao bì sản phẩm hay có thể gắn vào bất cứ vật nào chúng ta muốn theo dõi.

Hiện nay, các hệ thống đánh giá mật mã nhẹ vẫn đang trong quá trình hoàn thiện. Năm 2015 viện tiêu chuẩn và công nghệ quốc gia Hoa Kỳ cũng đã bắt tay vào đánh giá và chuẩn hóa kỹ thuật mã hóa nhẹ. Đến thời điểm hiện tại, chưa có một thông báo chính thức nào để đánh giá một hệ mật là “*mật mã nhẹ*”.

* + 1. **Đặc điểm của mật mã nhẹ**

Tuy không có một khái niệm rõ ràng về mật mã nhẹ nhưng ta có thể nhận dạng nó thông qua một vài thông số như kích thước khối, kích thước khóa, số vòng mã hóa, và pha tính toán khóa của hệ mật.

Kích thước khối nhỏ: Để tiết kiệm bộ nhớ, mã khối nhẹ thông thường sử dụng khối nhỏ, chẳng hạn như 64 bit hoặc 80 bit [6].

Kích thước khóa nhỏ: Một vài mã khối nhẹ sử dụng khóa nhỏ, kích thước nhỏ hơn 96 bit. Tuy nhiên nó vẫn đảm bảo tính hiệu quả trong việc mã hóa [6]. Ví dụ như PRESENT 80 bit khóa.

Các vòng mã hóa đơn giản: Nhìn vào sơ đồ mã hóa của mã nhẹ, ta có thể dễ thấy các công thức tính toán tương đối đơn giản.

Tính toán khóa đơn giản: Pha tính toán khóa nếu sử dụng một công thức thức tạp sẽ dẫn đến việc tăng chi phí về lưu trữ, tăng độ trễ và năng lượng tính toán [6]. Như vậy, nhìn vào sơ đồ tính toán khóa của một mã nhẹ không thể nào lại là một công thức rối ren, phức tạp được.

* 1. **Động lực thúc đẩy mật mã nhẹ phát triển**

Hiện nay, mọi người có thể kết nối mạng ở bất cứ nơi đâu khi sử dụng thiết bị điện thoại thông minh, máy tính bảng hay laptop cá nhân của mình. Có thể chia sẻ hình ảnh, chia sẻ thông điệp hay gửi tin nhắn … bất cứ lúc nào mong muốn. Điều ta nên đặt ra câu hỏi là việc gửi và nhận như vậy có đảm bảo an toàn? Manh nha ở đâu đó, chúng ta có thể nghe những thuật ngữ như “*nhà thông minh*”, thẻ chíp, thẻ từ … Và khi bắt tay vào tìm hiểu, chúng ta sẽ đặt ra câu hỏi, làm sao để một ngôi nhà được gọi là thông minh trước mối nguy hại phá hoại và tấn công từ nhiều phía? Làm sao để những thẻ chíp, thẻ từ có thể sống sót trước sự nhòm ngó của kẻ thù?

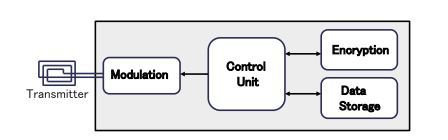
* + 1. **Internet kết nối vạn vật**

Những năm gần đây, thuật ngữ “Internet of things” [4] (IoT) rất phổ biến mà ai trong chúng ta đều có thể gặp ở bất cứ nơi đâu. Thuật ngữ này bắt đầu xuất hiện từ cuối những năm 90 của thế kỷ trước. Nhưng đến năm 1999 khi Keven Ashton đưa ra thì cụm từ này mới thực sự được xác nhận tồn tại [13].

IoT là một thuật ngữ đại diện cho một mạng lưới các vật tham gia kết nối internet. Ở đó, các vật có thể thu thập thông tin và truyền tải dữ liệu. Đây cũng chính là một điểm sáng rất lớn, là kỳ vọng cho lĩnh vực tự động hóa trong hầu hết các ngành nghề.

* + 1. **Công nghệ nhận dạng tần số sóng vô tuyến (RFID)**

Như đã nhắc ở phần 1.2.1, ở phần này tôi sẽ trình bày kỹ hơn một chút về công nghệ RFID [10]. Công nghệ nhận dạng (hay còn gọi là nhận diện dùng để đọc dữ liệu từ chip, thẻ hoặc là thu lấy hình ảnh của đối tượng để mang về máy tính xử lý) không tiếp xúc, sử dụng tần số sóng vô tuyến.

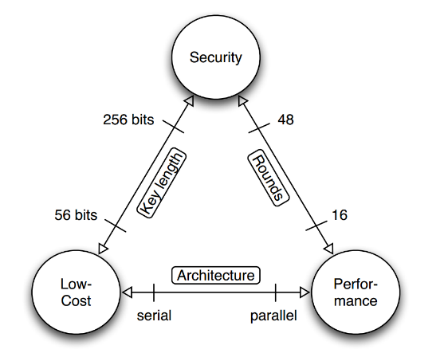


Hình 1.1: Cấu trúc của một thiết bị RFID

Hiện nay, RFID đang được nghiên cứu rất nhiều để áp dụng trong lĩnh vực tự động hóa. Ví dụ như ô tô tự động hóa, tự động hóa trả phí đường bộ …

* 1. **Chiến lược thiết kế cho mật mã nhẹ**

Mỗi một chiến lược thiết kế mật mã nhẹ đều phải đối phó với sự đánh đổi giữa bảo mật, chi phí và hiệu suất. Đối với mã khối thì chiều dài khóa là sự đánh đổi lẫn nhau giữa độ bảo mật và chi phí. Trong khi số lượng vòng là sự hoán đổi lẫn nhau giữa độ bảo mật và hiệu suất. Và kiến trúc phần cứng là sự hoán đổi giữa chi phí và hiệu suất [3]. Ta hãy xem hình 1.2 bên dưới:



Hình 1.2: Thiết kế sự hoán đổi các yếu tố trong mật mã nhẹ

Luôn luôn, chúng ta chỉ có thể đạt được hai trong số ba chiến lược trong khi thiết kế. Lựa chọn bảo mật tốt và chi phí thấp nhưng như vậy thì hiệu suất lại thấp. Lựa chọn bảo mật tốt và hiệu suất cao thì lúc này chi phí của ta buộc phải cao. Hay cuối cùng ta chọn chi phí thấp và hiệu suất cao thì sự bảo mật lại lỏng lẻo.

Như vậy, chúng ta có ba hướng tiếp cận để tối ưu hóa một hệ mật khi xây dựng ứng dụng [3]:

- (1) Tối ưu hóa chi phí cài đặt trên phần cứng theo chuẩn và thuật toán tin tưởng.

- (2) Sử đổi một chút theo một nghiên cứu tốt và mã tin tưởng.

- (3) Thiết kế các mã mới để đạt được chi phí cài đặt phần cứng thấp theo yêu cầu thiết kế.

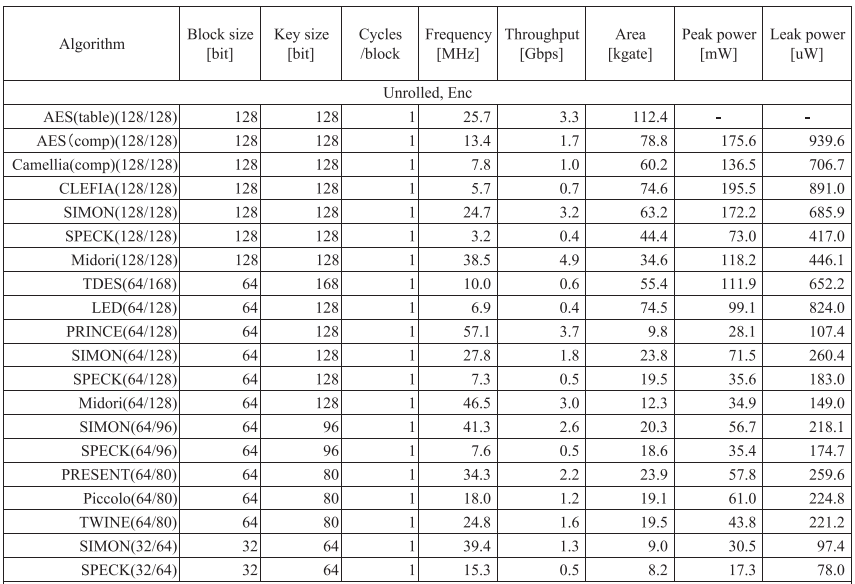
* 1. **Một số mật mã nhẹ**

Các thông số thống kê tham khảo từ bảng 1 của tài liệu [2] và bảng 2.8 của tài liệu [9]. Qua đó ta có thể thấy, các mã “nặng” truyền thống yêu cầu phần cứng lớn hơn rất nhiều so với các mã nhẹ. Ngay từ đơn vị của dùng để tính đã là Gbps và kGE còn mã nhẹ đơn vị sử dụng là Kbps và GE. Ví dụ Keccack-1600 là thuật toán dùng để cài đặt SHA3 có thông lượng yêu cầu 22 Gbps và bề mặt là 48 kGE, PRSENT-80 yêu cầu thông lượng là 11.4 Kbps và bề mặt là 1075 GE. Như vậy, yêu cầu phần cứng của mã nhẹ thấp hơn rất nhiều so với các mã nặng.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Mã nhẹ** | | | **Mã nặng truyền thống** | | |
| **Tên** | **Thông lượng (Kbps)** | **Bề mặt (GE)** | **Tên** | **Thông lượng (Gbps)** | **Bề mặt (kGE)** |
| **PRESENT – 80** | 11.4 | 1075 | Keccak-1600 | 22 | 48 |
| **DES** | 44.5 | 2309 | BLAKE-512 | 18.8 | 79 |
| **mCrypton** | 492.3 | 2681 | Skein-512 | 58 | 61 |
| **TEA** | 100 | 2355 |  |  |  |
| **Grain** | 100 | 1294 |  |  |  |
| **Trivium** | 100 | 2599 |  |  |  |

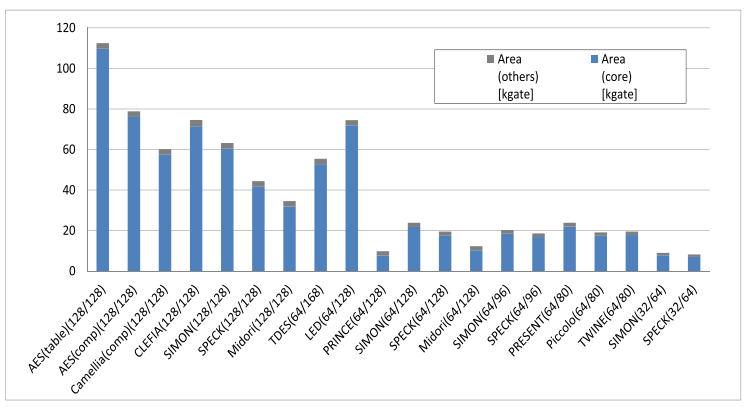
Bảng 1.1: Một số hệ mật nhẹ và một số hệ mật “nặng” truyền thống

Để biết rõ hơn thông tin về năng lượng tiêu thụ và các chi phí về phần cứng của các hệ mật nhẹ, ta có thể theo dõi bảng 1.2 [7]. Nhìn vào những thông tin được liệt kê trong bảng, ta có thể nắm được phần nào về yêu cầu phần cứng. Từ đó, có thể dùng những thông tin này để đưa ra quyết định lựa chọn một hệ mật sao cho phù hợp với thiết bị nhẹ của mình.

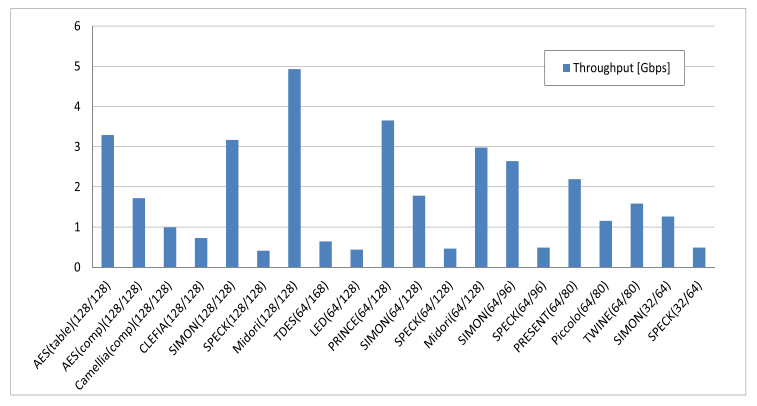


Bảng 1.2: Thông tin về yêu cầu phần cứng của một vài hệ mật nhẹ

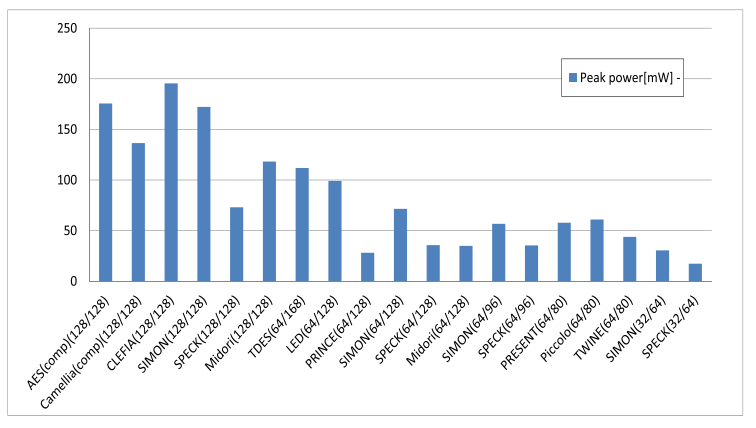
Để có một cái nhìn trực quan hơn về các thông số thống kê được, tác giả xin được liệt kê một vài đồ thị tham khảo từ tài liệu [7].



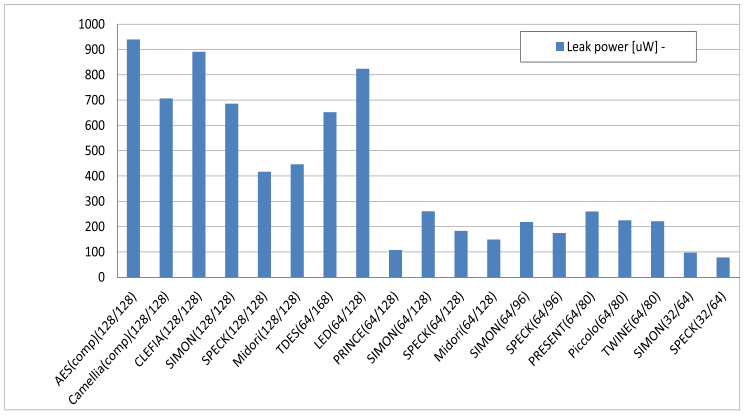
Hình 1.3: Đồ thị so sánh theo thông số bề mặt của một số hàm băm nhẹ



Hình 1.4: Đồ thị so sánh theo thông số thông lượng của một số hàm băm nhẹ



Hình 1.5: Đồ thị so sánh năng lượng sử dụng ở mức cao của một số hàm băm nhẹ



Hình 1.6: Đồ thị so sánh năng lượng sử dụng ở mức thấp của một số hàm băm nhẹ

# Chương 2: HÀM BĂM NHẸ

* 1. **Khái niệm**

Hàm băm nhẹ là một phần nhỏ trong nhánh nghiên cứu của mật mã nhẹ nên nó cũng không có một ranh giới rõ ràng nào để phân biệt nhẹ hay không nhẹ [8]. Mục tiêu của hàm băm nhẹ hướng tới sự nhỏ gọn trong cài đặt để phù hợp với các thiết bị bị giới hạn bởi dung lượng lưu trữ và năng lượng tiêu thụ.

* + 1. **Các yêu cầu cơ bản của hàm băm nhẹ**

Tuy không có một khái niệm rõ ràng nào về hàm băm nhẹ, nhưng nó vẫn phải tuẩn thủ những nguyên tắc cơ bản của một hàm băm thông thường. Ba yếu tố dưới đây phải đảm bảo: *Preimage resistant, Second preimage* *resistant, Collision resistant*

* + 1. **Động lực phát triển của hàm băm nhẹ**

Hàm băm nhẹ là một nhánh nghiên cứu con trong mật mã nhẹ và là một trong những nhánh quan trọng bậc nhất và cấp thiết nhất cho tới thời điểm hiện tại bởi tính hữu dụng của nó.

Khi IoT phát triển, xung quanh ta có rất nhiều thiết bị muốn kết nối internet. Từ những thiết bị gia dụng cho tới những phương tiện tham gia giao thông hay thẻ chip, thẻ từ. Để ý thấy, các thiết bị này đều là những thiết bị mang tính riêng tư, như vậy ta cần bảo mật để không một ai khác ngoài chúng ta có thể kiểm soát. Mật khẩu là một trong những cơ chế tốt nhất để bảo vệ tính cho tới thời điểm hiện tại. Mà mật khẩu chính là một ứng dụng của hàm băm. Đặc điểm của phần lớn các thiết bị IoT là bị giới hạn rất nhiều về nặng lượng tiêu thụ và khả năng tính toán, do đó đây cũng chính là điểm mạnh của hàm băm nhẹ, là động lực thúc đẩy mật mã nhẹ nói chung và hàm băm nhẹ nói riêng phát triển.

* 1. **Ứng dụng của hàm băm nhẹ**

Như tác giả đã trình bày ở phần triển vọng của hàm băm nhẹ, do hàm băm là nhánh con của hàm băm nhẹ nên ứng dụng của nó cũng không ngoài ứng dụng của hàm băm. Như vậy, ta có thể điểm qua một vài ứng dụng của hàm băm nhẹ như sau: Xác thực, bảo vệ thông điệp

* 1. **Thách thức của hàm băm nhẹ**

Thách thức lớn nhất đối với hàm băm nhẹ nói riêng và mật mã nhẹ nói chung là việc làm sao để đảm bảo chương trình nhẹ nhàng để cài đặt trên các thiết bị bị giới hạn bởi năng lượng tiêu thụ và khả năng lưu trữ, đồng thời phải đảm bảo yếu tố bảo mật và yếu tố hiệu suất.

Như vậy, thách thức đối với mật mã nhẹ nói chung và hàm băm nhẹ nói riêng là rất lớn, làm sao để tối đa bảo mật, tối đa hiệu suất nhưng lại giảm chi phí. Thách thức lớn đồng thời cũng là một mỏ vàng để các nhà mật mã học khai thác.

* 1. **Một số hàm băm nhẹ**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hàm băm** | **Kích thước khối** | **Preimage** | **Second preimage** | **Collisions** | **Công nghệ sử dụng** | **Bề mặt (GE)** | **Thông lượng (Kb/s @ 100kHz)** | **Năng lượng tiêu thụ**  **(**µW**)** |
| **ARMADILLO** | 80 | 280 | 280 | 240 | 0.18 µm | 4030/2923 | 109/27 |  |
| 128 | 2128 | 2128 | 264 | 6025/4353 | 1000/250 |  |
| **PHOTON** | 80 | 264 | 240 | 240 | 0.18 µm | 85/1168 | 2.82/15.15 |  |
| **QUARK** | 136 | 2128 | 264 | 264 | 0.18 µm | 1379/2392 | 1.47/11.76 | 2.44/4.07 |
| **SPONGENT** | 80 | 280 | 240 | 240 | 0.13 µm | 738/1127 | 0.81/17.78 | 1.57/2.31 |

Bảng 2.1: Một số hàm băm nhẹ

Thông qua bảng 2.1 (*các thông sô tham khảo từ tài liệu* [10]) ta có thể thấy được một số hàm băm nhẹ và các thông số đi kèm của nó. Dễ thấy, những hàm băm này có độ bảo mật rất khá thể hiện qua các thông số “cryptographic properties”.

* 1. **Hàm băm của hệ mật PRESENT**
     1. **Hệ mật PRESENT**

Trong phần này, tác giả sẽ trình bày hiểu biết của mình về hệ mật PRESENT [1] đã được công bố trong bài báo “***Present: An Ultra-Lightweight Block Cipher***” của A. Bogdanov và các cộng sự.

Tác giả sẽ trình bày từ kế hoạch thiết kế cho tới việc thiết kế chi tiết hệ mật này. Đây cũng là cách tiếp cận thứ 3 đã được đề cập ở mục 1.3 (*chiến lược thiết kế cho mật mã nhẹ*) là thiết kế một hệ mật mới phù hợp với yêu cầu bảo mật của các thiết bị bị giới hạn.

* + - 1. **Ý tưởng thiết kế**

Mục tiêu khi Bogdanov và các cộng sự thiết kế PRESENT là muốn xây dựng một hệ mật thật đơn giản nhưng hiệu quả.

- Mã hóa sẽ được thực hiện trong phần cứng.

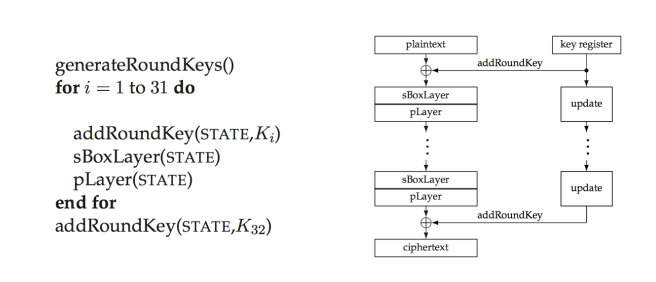
- Các ứng dụng sẽ chỉ yêu cầu mức bảo mật vừa phải.

- Các ứng dụng sẽ không yêu cầu mã hóa một lượng lớn dữ liệu.

Từ những phác thảo về quyết định thiết kế, Bogdanov và các cộng sự đã đề xuất hệ mật PRESENT.

* + - 1. **Quá trình mã hóa**

Mỗi vòng trong số 31 vòng bao gồm một thao tác XOR để đưa ra một khóa tròn Ki sao cho 1 ≤ i ≤ 32, trong đó K32 được sử dụng cho post-whitening, hoán vị bitwise tuyến tính và một lớp thay thế không tuyến tính. Lớp phi tuyến tính sử dụng S-Box 4 bit đơn, được áp dụng song song 16 lần trong mỗi vòng. Các vòng mã hóa được mô tả trong hình 2.3 bao gồm các pha sinh khóa, addRoundKey, S-Box layer, pLayer. Hoạt động của từng pha sẽ được trình bày ở các phần ngay bên dưới đây.



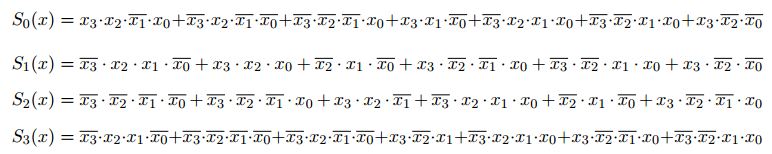
Hình 2.3: Quy trình mã hóa của PRESENT

***Hàm addRoundKey***

Giả sử ta có tập khóa Ki = …. sao cho 1 <= i <= 32 và STATE hiện tại là b63…b0, hàm addRoundKey bao gồm vòng lặp từ 0 <= j <= 63 thỏa mãn: bj 🡪 bj 

***Hàm sBoxLayer***

Bằng cách sử dụng công cụ tối ưu hóa boolean *espresso*, ta thu được bốn bit đầu ra kiểu boolean cho S-Box hiện tại:

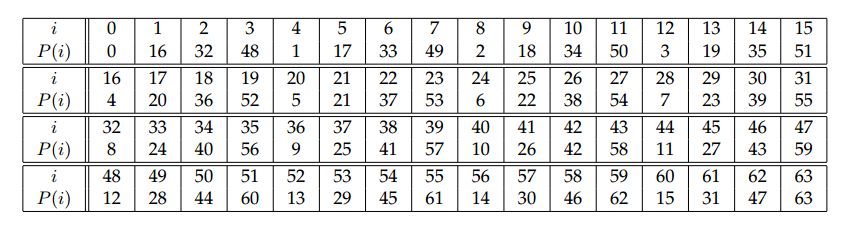


Trong đó “·” biểu thị phép logic AND, “+” biểu thị phép logic OR. Bảng 2.2 là S-Box trong hệ thập lục phân:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| S[x] | C | 5 | 6 | B | 9 | 0 | A | D | 3 | E | F | 8 | 4 | 7 | 1 | 2 |

Bảng 2.2: Hộp S-Box 4 bit của hệ mật PRESENT trong hệ thập lục phân.

***Hàm pLayer***



Bảng 2.3: Hoán vị bit sử dụng trong PRESENT

Bit i của STATE được chuyển sang vị trí bit P (i).

Cũng có thể viết P-layer theo cách sau:

P(i) =

* + - 1. **Quá trình giải mã**

***Hàm addRoundKey***

Giả sử ta có tập khóa Ki = …. sao cho 1 <= i <= 32 và STATE hiện tại b63…b0, hàm addRoundKey bao gồm vòng lặp từ 0 <= j <= 63 thỏa mãn: bj 🡪 bj 

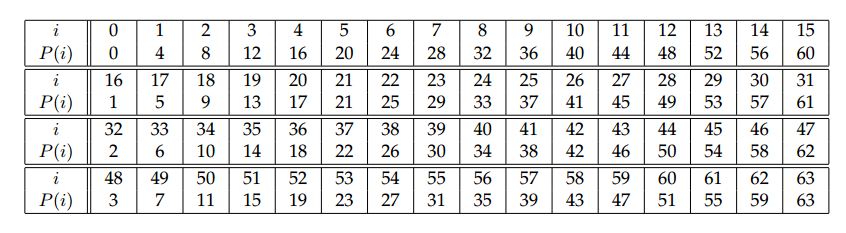
***Hàm invSBoxlayer***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| S[x] | 5 | E | F | 8 | C | 1 | 2 | D | B | 4 | 6 | 3 | 0 | 7 | 9 | A |

Bảng 2.4: Hộp S-Box 4 bit nghịch đảo của hệ mật PRESENT trong hệ thập lục phân

***invPLayer***

Các hoán vị bit được sử dụng trong giải mã của PRESENT được cho trong bảng 2.5 phía dưới đây. Bit i của STATE được chuyển sang vị trí bit P(i).



Bảng 2.5: Nghịch đảo việc hoán vị bit trong hệ mật PRESENT

* + - 1. **Tính toán khóa**

***Tính toán khóa cho PRESENT-80***

Khóa do người dùng cung cấp được lưu trữ lại trong thiết bị, ký hiệu là K và được biểu diễn là k79k78 … k0. Tại vòng i, khóa tròn 64 bit Ki = κ63κ62 … κ0 bao gồm 64 bit trái là nội dung hiện tại của K. Như vậy tại vòng i ta có:

Ki = κ63κ62 … κ0 = k79k78 … k16

Sau khi tách vòng khóa Ki, khóa K = k79k78 … k0 được cập nhật như sau:

1. [k79k78 … k1 k0] = [k18k17 … k20k19]

2. [k79k78k77k76] = S [k79k78k77k76]

3. [k19k18k17k16k15] = [k19k18k17k16k15]  round\_counter

***Tính toán khóa cho PRESENT-128***

Tính toán khóa cho 128 bit cũng tương tự như khóa 80 bit. Ban đầu, khóa do người dùng cung cấp sẽ được lưu trữ lại, ký hiệu là K và và được biểu diễn dưới dạng k127k126 … k0. Tại vòng thứ i thì vòng khóa 64 bit Ki = κ63κ62 … κ0 bao gồm 64 bit trái của khóa lưu trữ. Vì thế, tại vòng thứ i ta có:

Ki = κ63κ62 … κ0 = k127k126 … k64.

Sau khi tách vòng khóa Ki, khóa đăng ký K = k127k126 … k0 được cập nhật như sau:

1. [k127k126 … k1k0] = [k66k65 … k68k67]

2. [k127k126k125k124] = S[k127k126k125k124]

3. [k123k122k121k120] = S[k123k122k121k120]

4. [k66k65k64 k63k62] = [k66k65k64 k63k62]  round\_counter

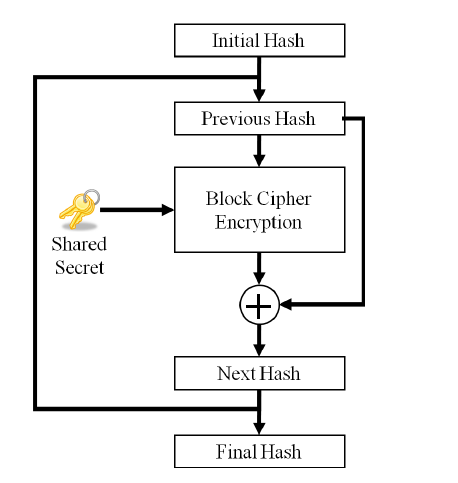
* + 1. **Hàm Băm của hệ mật PRESENT**

Có rất nhiều lựa chọn để xây dựng một hàm băm 64 bit từ mã khối 64 bit. Trong phần này, chúng ta tìm hiểu dựa trên công thức Davies-Mayer. Trong một vài tài liệu sẽ lấy tiền tố DM- đặt trước hàm PRESENT và gọi là DM-PRESENT ám chỉ hàm băm của hệ mật PRESENT sử dụng Davies-Mayer.

Công thức tính toán của chúng ta là:

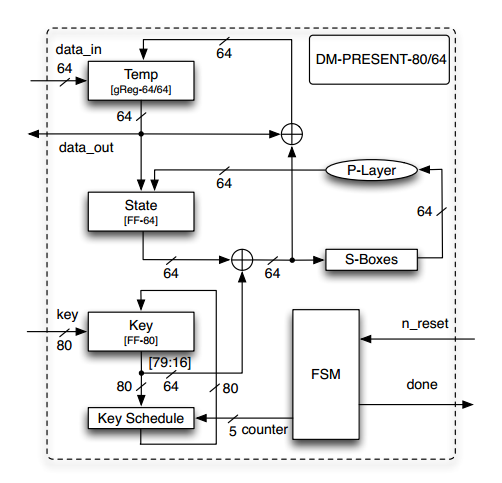
= E (Hi, M)  Hi

Trong đó là chuỗi băm đầu ra, E là hàm mã hóa, Hi là chuỗi đầu vào, là khóa.



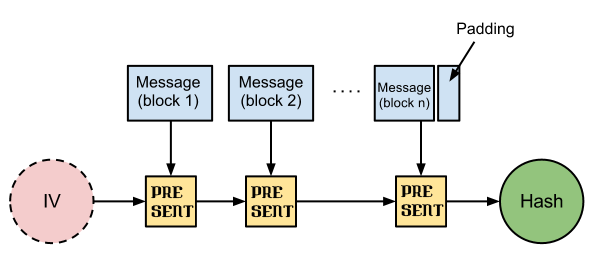
Hình 2.6: Cấu trúc băm sử dụng công thức Davies-Mayer

Để có cái nhìn trực quan hơn về kiến trúc băm của hàm băm của PRESENT, ta có thể theo dõi hình 2.7: *Sơ đồ tuần tự hàm băm của hệ mật PRESENT theo công thức DaviesMayer và cấu trúc Merkle Damgard* và hình 2.8: *Kiến trúc của hàm băm PRESENT theo cấu trúc Davies Mayer với đầu vào 64 bit và khóa 80 bit.*



Hình 2.8: Kiến trúc của hàm băm PRESENT theo cấu trúc Davies Mayer với đầu vào 64 bit và khóa 80 bit

Trong trường hợp đầu vào có kích thước lớn hơn 64 bit, ta phải sử dụng thêm một cấu trúc rất phổ biến là Merkle Damgard để tóm tắt thông điệp.



Hình 2.9: Cấu trúc Merkle Damgard

Qua hình 2.9 ta có thể thấy, thông điệp đầu vào sẽ được chia thành các thông điệp nhỏ hơn có chiều dài 64 bit. Nếu thông điệp cuối cùng có chiều dài nhỏ hơn 64 bit thì nó sẽ được gắn thêm một chuỗi bit vào đằng sau như trường hợp đầu tiên phía bên trên. IV chính là vector khởi tạo, là khóa do người dùng cung cấp. Sau vòng mã hóa đầu tiên, IV sẽ được cập nhật là 64 bit đầu ra gắn liền phía sau là 16 bit cuối cùng của khóa. Cứ như vậy thực hiện băm đến hết chiều dài của chuỗi đầu vào, ta sẽ thu được chuỗi băm đầu ra có kích thước 64 bit.

Với hàm PRESENT 80 bit khóa, khối 64 bit thì số vòng trên mỗi khối là 32. Cài đặt trên phần cứng tính toán được là: thông lượng: 200 Kbps tại 100KHz, sử dụng công nghệ 0.18 micromet và số vòng đạt được 1570 [1]. Như vậy hàm PRESENT rất thuận lợi để cài đặt trên phần cứng của các thiết bị IoT bị giới hạn nhiều bởi lượng tiêu thụ và khả năng lưu trữ. Với độ bảo mật hiện tại đại diện là preimage và second preimage của hàm nén PRESENT là 264 thì việc áp dụng hàm này vào thực tế rất khả thi.

# Chương 3: THỰC NGHIỆM

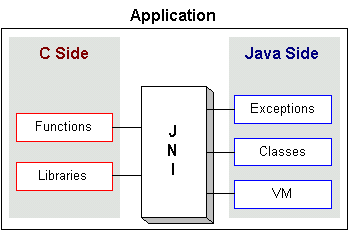
Ở chương 4, tác giả tiến hành cài đặt thuật toán băm PRESENT theo cấu trúc Merkle Damgard sau đó ứng dụng vào xác thực mật khẩu trên các app di động nền tảng Android.

* 1. **Mục đích thực nghiệm**

Cài đặt và ứng dụng chương trình băm PRESENT vào các app di động. Tương lai có thể áp dụng vào các thiết bị bị giới hạn bởi năng lượng tiêu thụ và khả năng lưu trữ như Arduino.

* 1. **Tiến hành thực nghiệm**
     1. **Tích hợp PRESENT vào app di động**

Build hàm băm PRESENT theo dạng thư viện tĩnh sau đó thông qua Jni để sử dụng như một thư viện của Java/Android. IDE sử dụng Android Studio ver 3.0. Cấu trúc sử dụng như hình 3.1 bên dưới:

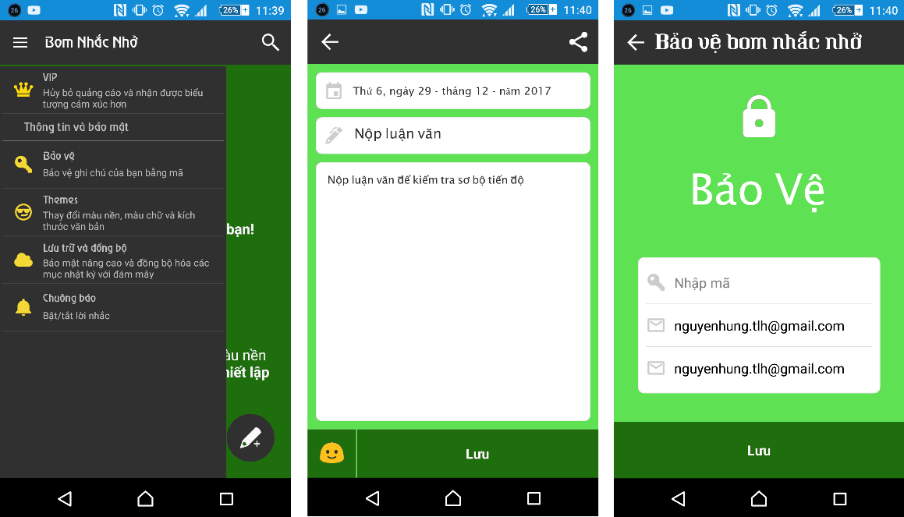


Hình 3.1: Sử dụng Jni làm cầu nối để gọi qua lại giữa Java và C/C++

* 1. **Kết quả thực nghiệm**

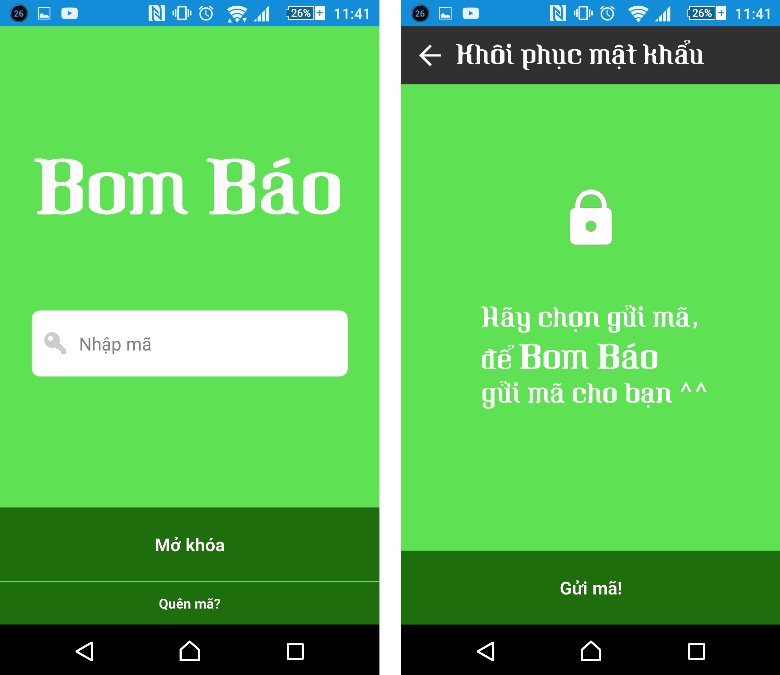
Xây dựng chương trình Bom Báo. Chương trình này cho phép người dùng lưu lại thời gian sự kiện cần nhắc nhở. Sau đó đến thời điểm đã được đặt sẵn, chương trình sẽ thông báo cho người dùng. Chương trình băm PRESENT đảm bảo chức năng bảo mật cho mã bảo vệ, giúp thông tin của người dùng không bị xâm phạm.

* Đầu tiên, người dùng sẽ tạo một ghi chú, đến thời điểm đã được cài đặt, chương trình sẽ thông báo cho người dùng để nhắc nhở. Biểu diễn như *hình 3.2: Ứng dụng Bom Báo và chức năng bảo vệ*
* Nội dung thông báo có thể là những thông tin mà người dùng không muốn chia sẻ với người khác. Do đó việc cài đặt xác thực trên những ghi chú đó là điều nên làm. Trong chương trình Bom Báo, người dùng lựa chọn chức năng “Bảo vệ” như màn 1, hình 3.2. Kế tiếp, người dùng cần chọn ghi chú muốn bảo vệ, sau đó màn hình 3, hình 3.2 sẽ hiện ra yêu cầu người dùng nhập mã và địa chỉ email.



Hình 3.2: Ứng dụng Bom Báo và chức năng bảo vệ

* Những lần sau, khi truy cập vào ghi chú đã được bảo vệ, người dùng phải nhập mã đã cài đặt thì mới truy cập được. Như màn 1, hình 3.3, người dùng cần nhập mã sau đó nhấn “Mở khóa” để truy cập.
* Trong trường hợp người dùng quên mã đã cài đặt, có thể lựa chọn chức năng “Quên mã?” như màn 1, hình 3.3, sau đó màn 2, hình 3.3 sẽ hiện ra. Khi màn 2, hình 3.3 hiện ra, người dùng cần lựa chọn “Gửi mã” để hệ thống gửi một mã reset mật khẩu đến email đã đăng ký. Lúc này, người dùng cần truy cập email để lấy mã cài đặt lại mật khẩu cho ghi chú.



Hình 3.3: Chức năng khóa trong ứng dụng Bom Báo

***Giai đoạn cài đặt mật khẩu:***

(1) Đầu tiên, người dùng sẽ yêu cầu bảo mật cho ghi chú (*Tương ứng với việc người dùng lựa chọn chức năng bảo mật trong màn 1 hình 3.2*).

(2) Chương trình sẽ yêu cầu nhập mật khẩu từ phía người dùng (*Tương ứng với màn 3 hình 3.2*).

(3) Người dùng nhập mật khẩu để bảo vệ ghi chú.

(4) Chương trình kiểm tra tính hợp lệ của mật khẩu về độ dài, về tính hợp lệ của ký tự.

(5) Nếu mật khẩu là hợp lệ, chương trình sẽ lưu mật khẩu vào cơ sở dữ liệu.

***Gia đoạn truy cập ghi chú:***

(9) Người dùng yêu cầu truy cập ghi chú

(13) Chương trình kiểm tra xem ghi chú có được bảo vệ bằng mật khẩu hay không

(14) Trường hợp ghi chú không được bảo vệ, người dùng có thể truy cập được tài liệu ngay lập tức.

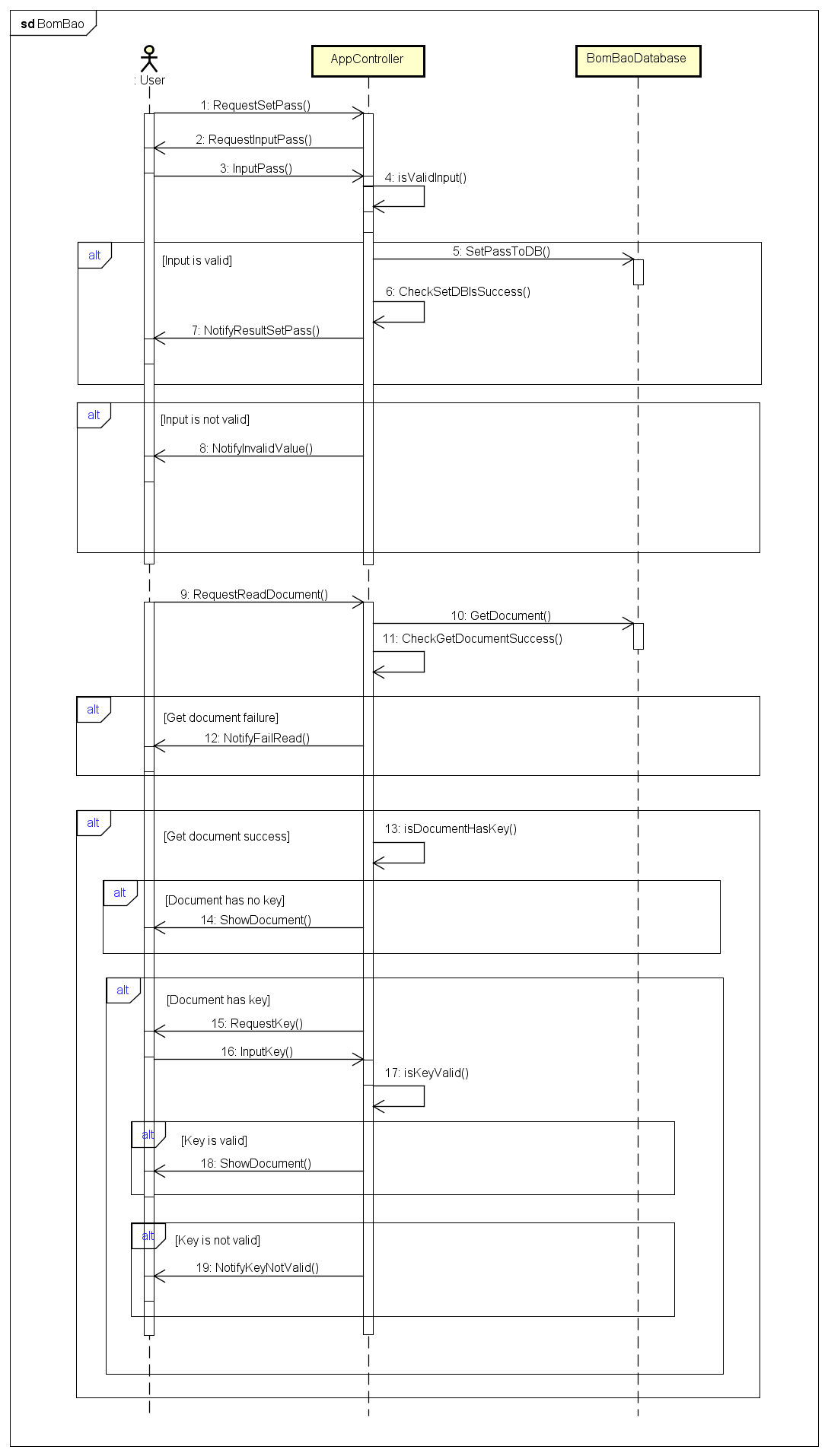
(15) Trường hợp ghi chú được bảo vệ, chương trình yêu cầu người dùng nhập mã bỏa vệ.

(16) Người dùng nhập mã.

(17) Chương trình sẽ kiểm tra tính hợp lệ của mật khẩu, sau đó kiểm tra xem mật khẩu có đúng với mật khẩu đã cài đặt hay không.

(18) Trường hợp mật khẩu người dùng nhập vào là đúng, chương trình sẽ hiển thị ghi chú tới người dùng.

(19) Trường hợp mật khẩu người dùng nhập vào là sai, chương trình sẽ thông báo tới người dùng để yêu cầu nhập lại hoặc gửi mã reset.



Hình 3.4: Sơ đồ tuần tự phần mật khẩu của chương trình Bom Báo

# KẾT LUẬN

Cùng với sự phát triển rất nhanh và mạnh của IoT và các nhóm ngành yêu cầu bảo mật ở mức nhẹ thì triển vọng của mật mã nhẹ là rất lớn. Trong luận văn của mình, tác giả đã đạt được một số kết quả như sau:

* Tìm hiểu một cách tổng quan về mật mã nhẹ nói chung và hàm băm nhẹ nói riêng.
* Tìm hiểu hệ mật PRESENT và hàm băm của nó theo công thức Davies Mayer và cấu trúc Merkle Damgard.
* Làm chủ mã nguồn và áp dụng được vào các ứng dụng của mình trong chương trình xác thực mật khẩu.

***Hướng phát triển tiếp theo:***

* Cài đặt bảo mật trên các thiết bị Arduino và những thiết bị bị giới hạn. Những thiết bị này được cài đặt các chương trình thu thập thông tin về thời tiết như nhiệt độ và độ ẩm.
* Nghiên cứu mã khối 16 bit và 128 bit của hệ mật PRESENT để có thể áp dụng vào bất cứ thiết bị nào được yêu cầu (thiết bị bị giới hạn hoặc là không).
* Cài đặt PRESENT theo hai hướng: Hướng đầu tiên là tối ưu hóa tốc độ xử lý, hướng thứ hai là tối ưu hóa không gian lưu trữ. Đưa ra đánh giá chi tiết về hai hướng cài đặt này để cuối cùng có thể áp dụng vào các thiết bị cụ thể.

# TÀI LIỆU THAM KHẢO

***Tài liệu tiếng anh***

[1] A. Bogdanov and L.R. Knudsen and G. Leander and C. Paar and A. Poschmann and M.J.B. Robshaw and Y. Seurin and C. Vikkelsoe, Present: *An Ultra-Lightweight Block Cipher*.

[2] Bogdanov and G. Leander and C. Paar and A. Poschmann and M.J.B Robshaw and Y. Seurin, Hash Function and RFID Tag: Mind the Gap 2008.

[3] Axel York Poschmann, *Cryptographic Engineering for Pervasive World*, Feb – 2009.

[4] Jen Clark, *IoT and the telecommunications industry*, Sep 13 – 2016.

[5] Joseph Sterling Grah, *Hash Functions in Cryptography*, 2008.

[6] Kerry A. McKay and Larry Bassham and Meltem Sonmez Turan and Nicky Mouha, Report on Lightweight Cryptography, March 2017.

[7] Naofumi Homma and WG members, *Cryptrec Cryptographic Technology Guideline (Lightweight Cryptography)*, Mar – 2017.

[8] Wenling Wu and Shuang Wu and Lei Zhang and Jian Zou and Le Dong, *LHash: A Lightweight Hash Function*.

[9] Zhijie Shi and Chujiao Ma and Jordan Cote and Bing Wang, Hardware implementation of hash function, 2013.

[10] <https://www.cryptolux.org/index.php/Lightweight_Hash_Functions>

[11] <https://en.oxforddictionaries.com/definition/internet_of_things>

[12] <http://www.lightweightcrypto.org/implementations.php>

[13] <https://vi.wikipedia.org/wiki/Internet_V%E1%BA%A1n_V%E1%BA%ADt>