

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

— * —

ĐỒ ÁN
TỐT NGHIỆP ĐẠI HỌC
NGÀNH CÔNG NGHỆ THÔNG TIN

**NGHIÊN CỨU GIAO TIẾP THIẾT BỊ NHÚNG
THEO CHUẨN USB TRÊN WINDOWS
VÀ ỨNG DỤNG TRONG DỊCH VỤ CHỮ KÝ SỐ**

Sinh viên thực hiện: **Hoàng Viết Huy**

Lớp ATTT - K59

Giáo viên hướng dẫn: TS. **Phạm Ngọc Hưng**

HÀ NỘI 05-2019

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên sinh viên: Hoàng Viết Huy

Điện thoại liên lạc: 0976986858

Email: hvhuybk@gmail.com.

Lớp: An toàn thông tin K59

Hệ đào tạo: Đại học chính quy

Đồ án tốt nghiệp được thực hiện tại: Trường Đại Học Bách Khoa Hà Nội

Thời gian làm ĐATN: Từ ngày 10/02/2019 đến 19/05/2019

2. Mục đích nội dung của ĐATN

Nghiên cứu và phát triển firmware, trình điều khiển cho thiết bị nhúng giao tiếp USB và ứng dụng trong thiết bị lưu trữ chữ ký số.

3. Các nhiệm vụ cụ thể của ĐATN

- Nghiên cứu và phát triển firmware cho thiết bị nhúng giao tiếp USB sử dụng bộ vi xử lý ARM.
- Nghiên cứu và phát triển trình điều khiển thiết bị giao tiếp USB trên Windows.
- Xây dựng kịch bản giao tiếp giữa Windows và thiết bị.
- Xây dựng thử nghiệm ứng dụng chữ ký số.

4. Lời cam đoan của sinh viên:

Tôi – Hoàng Viết Huy - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của TS Phạm Ngọc Hưng.

Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

	<i>Hà Nội, ngày 24 tháng 05 năm 2019</i> Tác giả ĐATN <i>Hoàng Viết Huy</i>
--	---

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

	<i>Hà Nội, ngày 24 tháng 05 năm 2019</i> Giáo viên hướng dẫn <i>TS Phạm Ngọc Hưng</i>
--	---

LỜI CẢM ƠN

Để có thể hoàn thành đồ án tốt nghiệp này, em xin gửi lời cảm ơn chân thành nhất tới tập thể các thầy giáo, cô giáo trường Đại học Bách Khoa Hà Nội nói chung, viện Công nghệ thông tin và truyền thông nói riêng, đã đào tạo cho em những kiến thức, những kinh nghiệm quý báu trong suốt thời gian học tập và rèn luyện.

Em xin gửi lời cảm ơn tới thầy giáo, TS. Phạm Ngọc Hưng - Giảng viên bộ môn Kỹ thuật máy tính, viện Công nghệ thông tin và truyền thông, trường Đại học Bách Khoa Hà Nội đã tận tình hướng dẫn em trong quá trình làm đồ án tốt nghiệp.

Tiếp theo, em xin gửi lời cảm ơn chân thành tới toàn thể các anh chị trong công ty An ninh mạng Viettel đã giúp đỡ rất nhiệt tình và tạo mọi điều kiện tốt nhất trong suốt quá trình em làm và hoàn thành đồ án tốt nghiệp tại công ty.

Cuối cùng là lời cảm ơn chân thành nhất tới những người thân thân trong gia đình, bạn bè đã luôn ở bên động viên, góp ý và tạo mọi điều kiện tốt nhất để tôi có thể hoàn thành đồ án tốt nghiệp này.

Tuy nhiên, do thời gian và kiến thức có hạn nên đồ án này chắc chắn không tránh khỏi những thiếu sót, em rất mong được sự đóng góp ý kiến của các thầy, các cô và toàn thể các bạn. Em xin chân thành cảm ơn.

Hà Nội ngày 24 tháng 5 năm 2019

Sinh viên: Hoàng Viết Huy

TÓM TẮT

Mục tiêu chính của đề án là nghiên cứu và phát triển nền tảng cho phép giao tiếp USB giữa PC và thiết bị nhúng dòng arm. Các công việc cụ thể bao gồm nghiên cứu và phát triển firmware cho thiết bị nhúng dòng arm, nghiên cứu và phát triển trình điều khiển thiết bị cho PC Windows, xây dựng kịch bản giao tiếp PC Windows với thiết bị và xây dựng thử nghiệm ứng dụng chữ ký số. Các thông tin trên sẽ được trình bày theo các chương như sau.

Chương 2 sẽ trình bày về việc nghiên cứu và phát triển firmware giao tiếp thiết bị. Đầu tiên, chương sẽ trình bày về cấu hình các thông số cho thiết bị. Tiếp theo trình bày về kịch bản nhận và thực thi lệnh từ user và việc tổ chức lưu dữ liệu trên bộ nhớ. Cuối cùng, trên firmware thiết bị, ta thiết kế khuôn dạng gói tin và các hàm xử lý với mỗi gói tin trên PC.

Chương 3 sẽ trình bày việc nghiên cứu và phát triển trình điều khiển thiết bị cho PC Windows, sau đó phát triển giải pháp giao tiếp sử dụng API Windows thông qua các API được export từ driver WinUSB. Tiếp theo là việc xây dựng giao tiếp với firmware thiết bị dựa vào các API vừa nêu trên. Phần cuối cùng của chương sẽ mô tả các hàm được xây dựng để giao tiếp với trình điều khiển thiết bị. Các hàm này được cung cấp giúp cho ứng dụng khác sau này có thể tái sử dụng cho các tính năng khác.

Chương 4 sẽ trình bày cơ sở lý thuyết về mật mã hóa khóa công khai và các thao tác với chữ ký số. Tiếp theo là quy trình sử dụng các hàm vừa xây dựng ở chương 3, ứng dụng trong dịch vụ chữ ký số. Cuối cùng, một số ảnh về giao diện cho ứng dụng. Các hình ảnh demo cho ứng dụng này.

Chương 5 sẽ nêu kết luận về kết quả nghiên cứu, phân tích các vấn đề đã làm được, chưa làm được, các đóng góp nổi bật. Từ đó rút ra bài học kinh nghiệm. Tiếp theo, trình bày hướng phát triển công việc này trong tương lai để hoàn thiện sản phẩm và các ứng dụng khác có thể hoạt động trên các vấn đề đã giải quyết.

MỤC LỤC

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP	2
LỜI CẢM ƠN.....	3
TÓM TẮT.....	4
MỤC LỤC	5
DANH MỤC HÌNH VẼ.....	7
CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI	1
1.1. Tổng quan.....	1
1.2. Mục tiêu cần đạt được	2
1.3. Giải pháp đề ra.....	2
1.4. Phân công công việc	3
CHƯƠNG 2: PHÁT TRIỂN FIRMWARE CHO THIẾT BỊ	4
2.1. Thiết bị thử nghiệm.....	4
2.2. Chuẩn USB.....	4
2.2.1. Tổng quan.....	5
2.2.2. Phần cứng	5
2.2.3. Giao thức	6
2.3. Thiết kế firmware	7
2.3.1. Cấu hình thông số giao diện USB	8
2.3.2. Nhận và thực thi lệnh từ PC	12
2.3.3. Tổ chức lưu trữ dữ liệu trên bộ nhớ.....	14
2.4. Xây dựng firmware	16
2.4.1. Cấu hình thông số USB và xử lý ngắt trên endpoint.....	17
2.4.2. Chức năng nhận và thực thi lệnh từ PC	19
2.4.3. Chức năng lưu trữ dữ liệu trên bộ nhớ.....	21
CHƯƠNG 3: GIAO TIẾP THIẾT BỊ NHÚNG USB TRÊN WINDOWS	24
3.1. Trình điều khiển thiết bị.....	24
3.2. Thiết kế giao thức	28
3.3. Thư viện API	30
CHƯƠNG 4: ỨNG DỤNG DỊCH VỤ CHỮ KÝ SỐ	33
4.1. Cơ sở lý thuyết.....	33
4.1.1. Tìm hiểu mật mã hóa khóa công khai.....	33
4.1.2. Thuật toán RSA trong mật mã hóa công khai.....	34
4.2. Các thao tác chữ ký số	35
4.2.1. Quy trình ký file bằng chữ ký số	35

4.2.2. Quy trình xác thực chữ ký số	36
4.3. Phân tích thiết kế ứng dụng.....	37
4.3.1. Biểu đồ chức năng	37
4.3.2. Biểu đồ hoạt động	38
4.4. Giao diện ứng dụng chữ ký số.....	41
CHƯƠNG 5: KẾT LUẬN.....	43
TÀI LIỆU THAM KHẢO.....	44

DANH MỤC HÌNH VẼ

Hình 1: Thiết bị USB eToken	1
Hình 2: Thiết kế kiến trúc USB eToken.....	2
Hình 3: Board thử nghiệm Keil MCB2300	4
Hình 4: Kết nối USB	5
Hình 5: Sơ đồ mạch kết nối USB.....	5
Hình 6: Giao tiếp pipe	6
Hình 8: Các chức năng chính của firmware	7
Hình 9: Phân cấp USB Descriptors.....	9
Hình 10: Quá trình trao đổi giữa PC và USB device	12
Hình 11: Cấu trúc chung của một gói tin	14
Hình 12: Cấu trúc của header block.....	15
Hình 13: Cấu trúc của data block.....	16
Hình 14: Tổng quát cách thức tổ chức lưu trữ dữ liệu trên bộ nhớ.	16
Hình 15: Cấu hình USB Configuration	17
Hình 16: Cấu trúc USB_DeviceDescriptor	18
Hình 17: Kiến trúc WinUSB.....	25
Hình 18. WinUSB driver	28
Hình 19. Quy trình sử dụng mật mã hóa khóa công khai.....	33
Hình 20: Mã hóa dữ liệu với Modulus n và số mũ riêng	36
Hình 21: Quy trình tạo chữ ký số.....	36
Hình 22: Giải mã dữ liệu với Modulus n và số mũ công khai	37
Hình 23: Quy trình xác minh chữ ký số	37
Hình 24: Biểu đồ chức năng ứng dụng chữ ký số.....	38
Hình 25: Biểu đồ hoạt động chức năng xác thực người dùng.....	38
Hình 26: Biểu đồ hoạt động chức năng set mã PIN mới.....	39
Hình 27: Biểu đồ hoạt động chức năng tạo và lưu khóa	39
Hình 28: Biểu đồ hoạt động chức năng tạo chữ ký số	40
Hình 29: Biểu đồ hoạt động chức năng xác thực chữ ký số.....	40
Hình 30: Giao diện chính ứng dụng chữ ký số	41
Hình 31: Giao diện tạo chữ ký số	41
Hình 32: Message Box thông báo xác thực thành công.....	42
Hình 33: Giao diện chọn file để ký hoặc verify chữ ký	42

DANH MỤC CÁC TỪ VIẾT TẮT VÀ THUẬT NGỮ

Số thứ tự	Từ viết tắt	Ý nghĩa
1	ĐATN	Đồ Án Tốt Nghiệp
2	OS	Operating System – Hệ điều hành
3	SDK	Software Development Kit – Bộ công cụ phát triển phần mềm
4	PC	Personal Computer – Máy tính cá nhân
5	USB	Universal Serial Bus

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

Trong chương này sẽ giới thiệu tổng quan, mục tiêu cần đạt được, lựa chọn kỹ thuật để giải quyết mục tiêu đề ra.

1.1. Tổng quan

USB (Universal Serial Bus) là một chuẩn kết nối tuần tự đa dụng trong máy tính. USB sử dụng để làm một chuẩn kết nối các thiết bị ngoại vi với máy tính, chúng thường được thiết kế dưới dạng các đầu cắm cho các thiết bị tuân theo chuẩn cắm-và-chạy.

Giao tiếp USB được ứng dụng trong nhiều thiết bị với giao thức đã được chuẩn hóa. Tuy nhiên, với các nhu cầu riêng biệt, thì yêu cầu cần các giao thức không theo chuẩn là nhu cầu cần thiết. Từ đó, việc hiểu về cách thức giao tiếp USB là rất quan trọng để có thể tùy chỉnh theo từng nhu cầu.

Giao thức được tùy biến có nhiều ứng dụng khác nhau, điển hình là lưu trữ khóa bí mật dùng để bảo đảm tính bí mật và tính toàn vẹn dữ liệu.

Theo tìm hiểu một số công ty lớn sử dụng USB làm thiết bị CA, em nhận thấy các công ty đang phụ thuộc khá nhiều vào các nhà sản xuất phần cứng (Independent Hardware Vendor - IHV) ví dụ nổi tiếng nhất như SecureMetric. Việc phụ thuộc vào các nhà phát triển phần cứng với cộng với việc đặt số lượng lớn giúp chi phí nghiên cứu và giá thành giảm xuống. Việc này mang lại lợi ích kinh tế trước mắt, tuy nhiên, việc đó cũng làm chúng ta phụ thuộc vào công nghệ của các công ty này. Thay vì việc tùy biến thiết bị theo ý muốn, chúng ta cố gắng ép mình theo khả năng của thiết bị.

Dựa trên các phân tích đánh giá trên, ta nhận thấy với việc sử dụng thiết bị của các nhà cung cấp thiết bị, khả năng tùy biến thiết bị của chúng ta sẽ thấp, và nếu tiếp tục công nghệ nhúng của chúng ta sẽ khó phát triển mạnh hơn nếu không làm chủ được công nghệ.



Hình 1: Thiết bị USB eToken [5]

1.2. Mục tiêu cần đạt được

Các công việc cụ thể bao gồm nghiên cứu và phát triển firmware cho thiết bị nhúng dòng arm, nghiên cứu và phát triển trình điều khiển thiết bị cho PC Windows, xây dựng kịch bản giao tiếp PC Windows với thiết bị và xây dựng ứng dụng thử nghiệm chữ ký số.

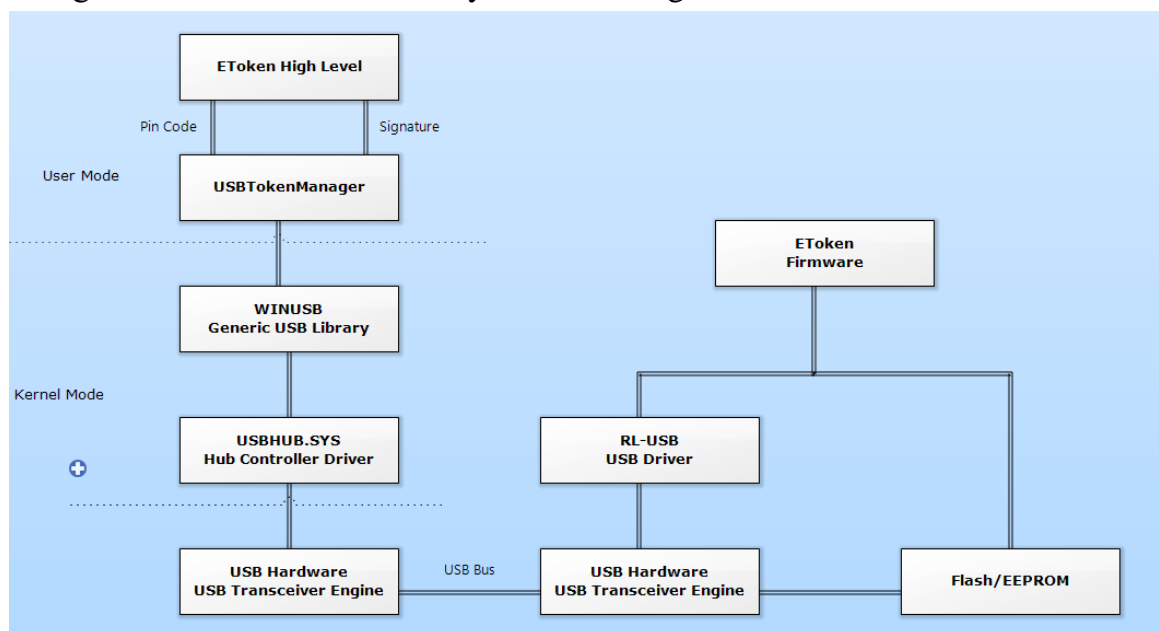
1.3. Giải pháp đề ra

Từ yêu cầu nêu ra bên trên, trước tiên, chúng ta cần tìm hiểu lập trình firmware cho thiết bị. Định nghĩa khuôn dạng các gói tin giao tiếp firmware và PC. Firmware sẽ được phát triển dựa trên các example của Keil.

Các công việc cụ thể bao gồm nghiên cứu và phát triển firmware cho thiết bị nhúng dòng arm, nghiên cứu và phát triển trình điều khiển thiết bị cho PC Windows, xây dựng kịch bản giao tiếp PC Windows với thiết bị và xây dựng thử nghiệm ứng dụng chữ ký số

Tiếp theo, chọn kỹ thuật giao tiếp USB với firmware của thiết bị, em chọn sử dụng công nghệ WinUSB trên Windows thay vì sử dụng User-mode framework (UMDf) hay kernel-mode driver framework (KMDF). Việc lựa chọn này giúp tiết kiệm thời gian phát triển, đồng thời đảm bảo có thể sử dụng đầy đủ tính năng cần cung cấp. Giải thích việc lựa chọn này sẽ được giải thích chi tiết trong Chương 3.

Giao diện ứng dụng trên Windows, sử dụng thư viện Winform trên nền .Net Framework. Việc lựa chọn Winform giúp giao diện ưa nhìn, tiết kiệm thời gian phát triển, và dễ tiếp cận. Ứng dụng trên Windows tương tác với thư viện SQLite cho phép tương tác với cơ sở dữ liệu chữ ký số để sử dụng.



Hình 2: Thiết kế kiến trúc USB eToken

1.4. Phân công công việc

Đồ án được thực hiện bởi nhóm tham gia bao gồm 2 sinh viên là Hoàng Viêt Huy, và Nguyễn Văn Việt.

Trong đó, Hoàng Viêt Huy thực hiện các công việc:

- Nghiên cứu và phát triển firmware cho thiết bị nhúng giao tiếp USB dòng arm.
- Nghiên cứu và phát triển trình điều khiển thiết bị giao tiếp USB trên Windows.
- Xây dựng kịch bản giao tiếp PC Windows với thiết bị.
- Xây dựng thử nghiệm ứng dụng chữ ký số trên Windows.

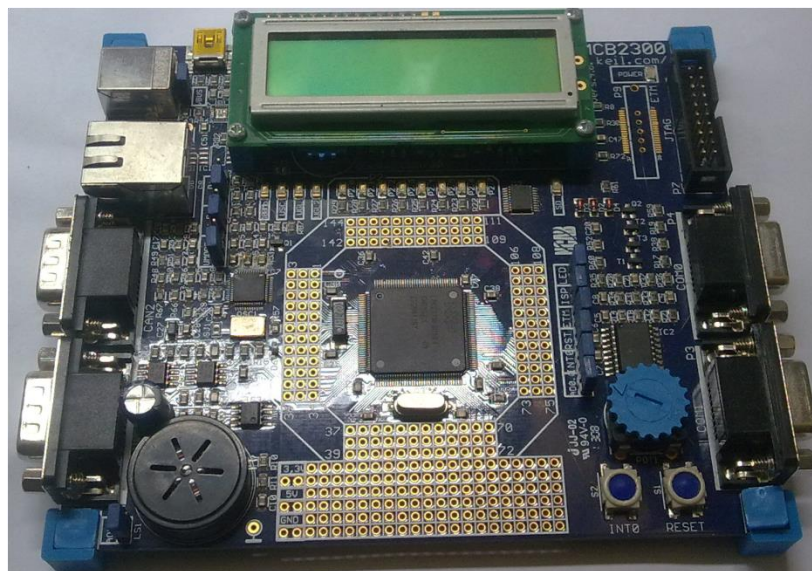
Nguyễn Văn Việt thực hiện các công việc:

- Nghiên cứu và phát triển firmware cho thiết bị nhúng giao tiếp USB dòng arm.
- Nghiên cứu và phát triển trình điều khiển thiết bị giao tiếp USB trên Linux.
- Xây dựng kịch bản giao tiếp PC Linux với thiết bị.
- Xây dựng thử nghiệm ứng dụng chữ ký số trên Linux.

CHƯƠNG 2: PHÁT TRIỂN FIRMWARE CHO THIẾT BỊ

2.1. Thiết bị thử nghiệm

- Để bắt đầu phát triển phiên bản thử nghiệm của thiết bị, chúng tôi có đề xuất sử dụng những thành phần thiết bị sau.
- Máy tính cá nhân PC: Laptop Asus K501LB-DM077D
 - CPU : Intel® Broadwell Core™ i5 _ 5200U (2.2GHz, 3M cache, up to 2.7Ghz).
 - Bộ nhớ : 8GB DDR3L Bus 1600 Mhz.
 - Ổ đĩa cứng : 1TB 5400rpm.
- Thiết bị phát triển:
 - Board phát triển Keil μ Vision 3 MBC2300. Board phát triển dùng để xây dựng phiên bản thử nghiệm (Prototype của thiết bị). Board này cung cấp khá nhiều giao tiếp, trong đó giao tiếp quan trọng với tất cả các USB Token là USB.



Hình 3: Board thử nghiệm Keil MCB2300

2.2. Chuẩn USB

USB (Universal Serial Bus – Bus tuần tự đa năng), là một chuẩn được phát triển để kết nối các thiết bị và một thiết bị điều khiển (thường là PC). Mục đích ban đầu của USB là thay thế các loại giao tiếp cũ như cổng tuần tự (serial port) và cổng song song (parallel port). USB kết nối các thiết bị ngoại vi như chuột, bàn phím, máy ảnh số, máy nghe nhạc, ổ đĩa flash và các ổ cứng gắn ngoài. Chính vì sự đa dạng và linh hoạt của USB, ngày nay nó đã được sử dụng trong hầu hết các thiết bị ngoại vi để kết nối với máy tính. USB được chuẩn hóa bởi USB Implement Forum (USBIF), tập hợp của các công ty dẫn đầu trong lĩnh vực công nghệ.

2.2.1. Tổng quan

USB có thiết kế bất đối xứng, một thiết bị host có nhiều cổng, các thiết bị ngoại vi kết nối đến các cổng đó theo kiến trúc phân tầng. Chỉ có một thiết bị điều khiển bus. Mỗi bus có thể nối tối đa với 127 thiết bị.



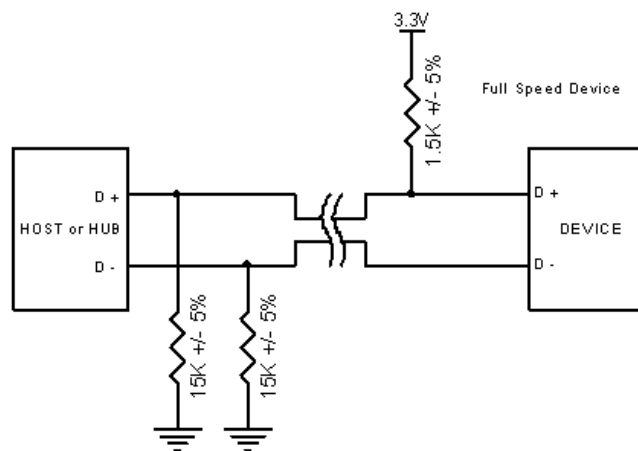
Hình 4: Kết nối USB

USB có bốn chuẩn tốc độ:

- Super Speed – 4.8Gbits/s
- High Speed – 480Mbits/s
- Full Speed – 12Mbits/s
- Low Speed – 1.5 Mbits/s.

Chuẩn Super Speed đang bắt đầu được thương mại hóa, nhưng số lượng thiết bị hỗ trợ chưa nhiều. Hiện nay chuẩn High Speed là phổ biến nhất. Vì điều khiển thử nghiệm LCP2378 hỗ trợ đến tốc độ FullSpeed, đủ cho việc lưu trữ và trao đổi các khóa riêng của mỗi người.

2.2.2. Phần cứng



Hình 5: Sơ đồ mạch kết nối USB

Kết nối USB có bốn dây. Hai dây cho nguồn điện (Vcc , GND) và hai dây cho tín hiệu vi sai (D+, D-). Thông tin được mã hóa NRZI và truyền đi trên cặp dây D+, D-.

Mức logic là '1' nếu D+ lớn hơn D- 200mV, và '0' nếu D+ nhỏ hơn D- 200mV. Thiết bị FullSpeed và LowSpeed được phân biệt bởi trở kéo trên dây tín hiệu là D+ hay D-. Thiết bị HighSpeed không có trở kéo, ban đầu nó sẽ hoạt động như FullSpeed, sau đó sẽ chuyển sang HighSpeed nếu host hỗ trợ.

Một ưu điểm của USB so với các chuẩn khác là nguồn điện. Thiết bị có thể lấy nguồn trực tiếp từ bus USB mà không cần nguồn điện ngoài. Điện áp trên bus là 5V, dòng cung cấp tối đa là 500mA mỗi cổng.

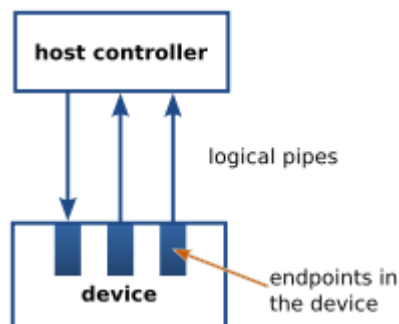
Tần số hoạt động của USB là 48MHz, sai số cho phép với thiết bị HighSpeed là +500ppm. Tương ứng của FullSpeed là 2500ppm và LowSpeed là 15000ppm.

2.2.3. *Giao thức*

Một thiết bị USB vật lý có thể có nhiều thiết bị logic. Thí dụ, webcam có thể tích hợp microphone. Đó là các thiết bị phức hợp. Mỗi thiết bị logic được USB host gán cho một địa chỉ riêng biệt trên bus và hoạt động như một thiết bị độc lập khác.

Thiết bị USB được hệ điều hành nhận diện và nạp trình điều khiển tương ứng thông qua định danh duy nhất gồm định danh nhà sản xuất (Vendor ID) và định danh sản phẩm (Product ID) , do tổ chức USBIF cấp phát.

Các thiết bị logic giao tiếp với host thông qua các kênh truyền logic gọi là Pipe. Pipe kết nối host tới một thực thể logic trên thiết bị gọi là EndPoint. Mỗi thiết bị có thể có tới 32 pipe, 16 pipe chiều từ host đến thiết bị và 16 pipe chiều ngược lại.



Hình 6: Giao tiếp pipe

Có hai loại pipe: pipe dòng và pipe thông điệp phụ thuộc vào phương pháp truyền. Có bốn phương pháp truyền:

Isochronous – phương pháp truyền đảm bảo tốc độ nhưng có khả năng mất mát dữ liệu (dữ liệu thời gian thực, hình ảnh, âm thanh).

Interrupt – phương pháp truyền đảm bảo khả năng đáp ứng, sử dụng cho các thiết bị chủ động (chuột, bàn phím...).

Bulk –phương pháp truyền một lượng lớn dữ liệu, sử dụng tất cả băng thông còn lại của bus, không có đảm bảo về độ trễ và tốc độ.

Control – thường để truyền một lượng nhỏ dữ liệu nhanh, đơn giản, được sử dụng để truyền lệnh và nhận phản hồi .

Pipe dòng là pipe một chiều, kết nối với các endpoint một chiều và truyền theo phương pháp isochronous, interrupt hoặc bulk. Pipe thông điệp là pipe hai chiều và chỉ được sử dụng để điều khiển thiết bị.

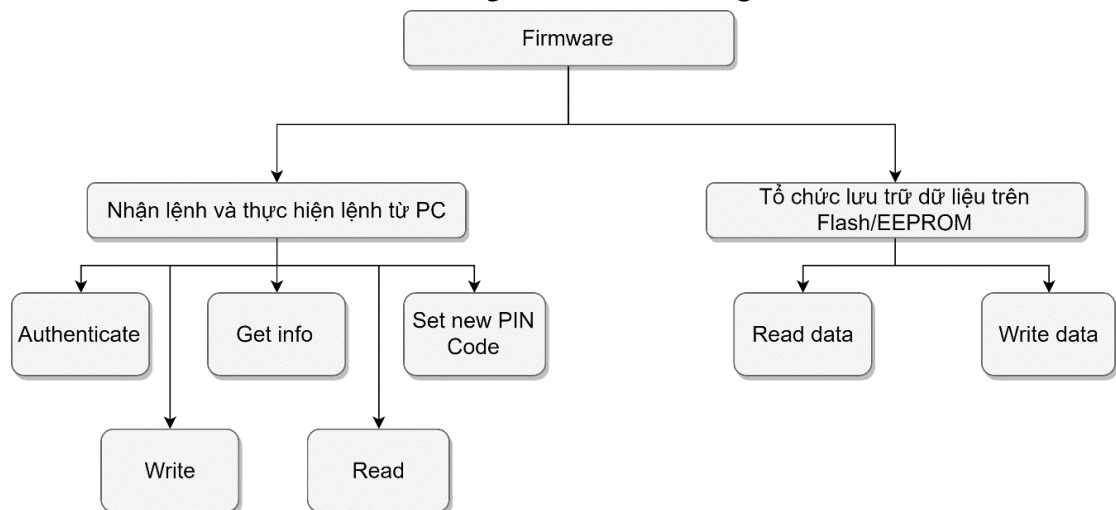
Mỗi thiết bị logic được gọi là interface, mỗi interface bao gồm một hay nhiều endpoint và thực hiện một chức năng của thiết bị vật lý. Tất cả các thiết bị USB đều có endpoint 0, đó là endpoint đặc biệt không thuộc interface nào, sử dụng để cấu hình thiết bị.

Khi thiết bị USB được kết nối vào USB host, host sẽ bắt đầu quá trình liệt kê (enumeration). Host sẽ gửi tín hiệu reset đến thiết bị, tốc độ được xác định qua quá trình reset. Sau đó, host sẽ đọc thông tin về thiết bị và gán cho nó một địa chỉ 7-bit duy nhất. Nếu host hỗ trợ thiết bị này, host sẽ nạp trình điều khiển tương ứng và thiết bị chuyển sang trạng thái đã cấu hình. Nếu usb host khởi động lại, toàn bộ quá trình liệt kê sẽ được thực hiện lại cho tất cả các thiết bị kết nối vào bus.

USB host điều khiển toàn bộ quá trình truyền nhận dữ liệu trên bus, không thiết bị nào có thể tự ý truyền nếu chưa có yêu cầu từ host.

2.3. Thiết kế firmware

Phần firmware của thiết bị sẽ bao gồm các chức năng chính sau:



Hình 7: Các chức năng chính của firmware

2.3.1. Cấu hình thông số giao diện USB

2.3.1.1. USB Descriptors

Tất cả các USB Device đều có cấu trúc descriptors phân cấp mô tả thông tin host như device là dạng gì, phiên bản USB nào hỗ trợ, bao nhiêu cách cấu hình, số lượng endpoints và kiểu endpoints của nó là gì.

Trong giao tiếp USB dạng giao tiếp cơ bản nhất là thông qua các Endpoint. Các Endpoint này là các vùng nhớ trên thiết bị, có nhiệm vụ để lưu trữ dữ liệu trong quá trình truyền nhận. Mỗi Endpoint được định danh bằng 8-bit. Bit cao nhất (bit 7) chỉ định hướng truyền nhận, 1 là IN (từ USB device lên USB host) và 0 là OUT (từ USB host xuống USB device), hướng truyền nhận được nhiều theo hướng từ phía Host. 4 bit thấp nhất (bit 0-3) đánh địa chỉ Endpoint (từ 0 -> 15), do đó có tối đa 16 Endpoint cho mỗi chiều truyền, nhận. Một Endpoint chỉ có thể mang dữ liệu theo một chiều IN hoặc OUT. Các bit còn lại (bit 4-6) được bỏ qua.

Các Endpoint có thể là một trong 4 loại tương ứng với 4 kiểu truyền của chuẩn USB:

Điều khiển (Control): là chế độ truyền được tất cả các thiết bị USB hỗ trợ để truyền các thông tin điều khiển với tốc độ tương đối chậm. Chúng thường được dùng cho việc cấu hình thiết bị, gửi lệnh, truy nhập tới các trạng thái thiết bị... Các Endpoint này thường có kích thước nhỏ. Mỗi thiết bị USB phải có ít nhất một Control Endpoint gọi là Endpoint 0. Endpoint này không thuộc về bất kỳ một interface nào và được sử dụng bởi USB Core để cấu hình thiết bị khi thiết bị được kết nối vào hệ thống.

Ngắt (Interrupt): sử dụng cho các thiết bị cần truyền một lượng dữ liệu nhỏ, tuân hoàn theo thời gian ví dụ như chuột, bàn phím, ở một tốc độ cố định mỗi khi USB host yêu cầu dữ liệu. Với mỗi Endpoint kiểu này có một chu kỳ ngắt (Interrupt Interval), việc truyền nhận sẽ được thực hiện cứ sau mỗi chu kỳ ngắt này.

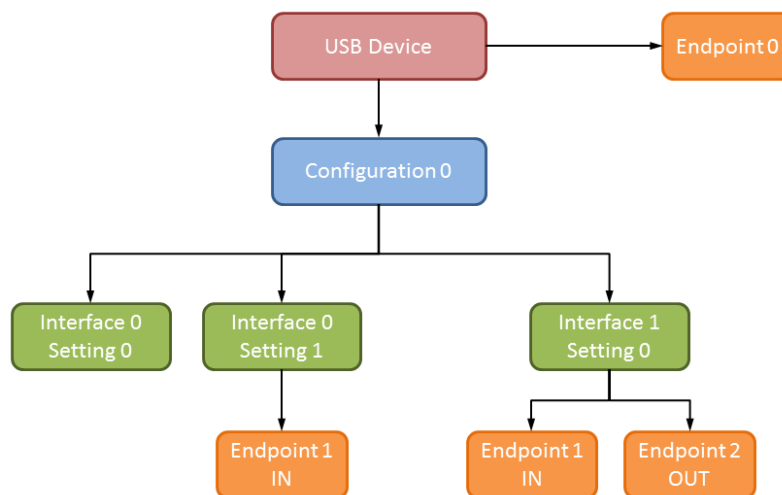
Khối (Bulk): sử dụng cho các thiết bị cần truyền một lượng dữ liệu lớn, yêu cầu độ chính xác tuyệt đối, yêu cầu tính toàn vẹn dữ liệu, không có ràng buộc quá chặt chẽ về thời gian thực. Kiểu truyền nhận này thường được sử dụng cho các máy in, các thiết bị lưu trữ, các thiết bị mạng. Bulk tương tự như giao thức TCP trong mạng Ethernet...

Đồng thời (Isochronous): sử dụng cho các thiết bị cần truyền một lượng dữ liệu lớn với tốc độ rất nhanh, đảm bảo ràng buộc về thời gian thực tuy nhiên chấp nhận hy sinh độ chính xác ở một mức nhất định như các thiết bị nghe nhạc, xem phim kết nối theo chuẩn USB. Chuẩn này tương tự giao thức UDP trong mạng Ethernet.

Mỗi Interface điều khiển một loại kết nối logic USB duy nhất, ví dụ như: chuột, bàn phím, âm thanh... Một thiết bị USB sẽ có thể có một hoặc nhiều Interface, một Interface có thể sử dụng một hoặc nhiều Endpoint, chẳng hạn một máy quay USB có thể bao gồm các Interface: âm thanh, video. Như vậy, đứng ở góc độ mức hệ thống, các Interface chính là các dịch vụ khác nhau mà thiết bị đó cung cấp còn các Endpoint chính là các cổng cần thiết cho mỗi dịch vụ. Tương ứng với khái niệm trong kiến trúc TCP/IP, ví dụ giao thức FTP là giao thức sử dụng để truyền file sẽ sử dụng hai cổng 20,21. Trong khi đó giao thức HTTP lại sử dụng port 80, giao thức Telnet sử dụng port 23.

Tiếp tục, các Interface lại được bao gồm trong các cấu hình (Configuration). Các Configuration thể hiện các trạng thái của một thiết bị. Ví dụ một thiết bị có các Configuration: hoạt động, chờ, khởi tạo.

Hình ảnh sau thể hiện mối quan hệ giữa các Endpoint, Interface, Configuration:



Hình 8: Phân cấp USB Descriptors [6]

2.3.1.2. Thiết kế thông số cho giao diện USB

Thiết bị giao tiếp với PC qua giao tiếp USB, PC có nhận dạng, điểm danh và cấu hình thiết bị thông qua hai mô tả chính là mô tả thiết bị (Device descriptor) và mô tả cấu hình (Configuration descriptor).

Mỗi thiết bị chỉ có 1 miêu tả thiết bị (Device Descriptors). Thông tin chứa trong mô tả thiết bị bao gồm phiên bản nào của USB mà thiết bị tuân theo, mã sản phẩm (Product IDs), mã nhà sản xuất (Vendor IDs), chính các thông tin này được sử dụng để xác định driver nào cần thiết để có thể giao tiếp được với thiết bị, cũng như số lượng cấu hình mà thiết bị có thể có. Số lượng cấu hình của thiết bị liên quan đến việc cấu hình mà thiết bị có thể theo (tất nhiên mỗi thời điểm chỉ 1).

Miêu tả cấu hình chỉ rõ các giá trị như điện năng sử dụng trong mỗi cấu hình, thiết bị là tự cấp nguồn (self-powered) hay lấy nguồn từ bus (bus-powered), rồi số lượng

giao diện (interface) của mỗi cấu hình. Trong quá trình enumerated, Host sẽ đọc miêu tả thiết bị, và đưa ra quyết định cấu hình nào được sử dụng. Chỉ được phép sử dụng 1 cấu hình. Trong mô tả cấu hình bao gồm mô tả interface mà cấu hình đó có và mô tả endpoint ứng với từng interface.

- Thông số của bản mô tả thiết bị:
 - bDeviceClass, bDeviceSubClass, bDeviceProtocol: được thiết lập là 0x00, khi đó trong mỗi interface sẽ định nghĩa class code.
 - bMaxPacketSize: độ dài tối đa của gói tin của endpoint 0.
 - idVendor và idProduct: các tác dụng định danh thiết bị trong hệ thống và dùng để match với driver tương ứng giúp giao tiếp với thiết bị trong hệ thống Windows.

```
/* USB Standard Device Descriptor */
const BYTE USB_DeviceDescriptor[] = {
    USB_DEVICE_DESC_SIZE,          /* bLength */
    USB_DEVICE_DESCRIPTOR_TYPE,    /* bDescriptorType */
    WBVAL(0x0200), /* 2.00 */    /* bcdUSB */
    0x00,                          /* bDeviceClass */
    0x00,                          /* bDeviceSubClass */
    0x00,                          /* bDeviceProtocol */
    8,                             /* bMaxPacketSize0 */
    WBVAL(0xAAAA),                /* idVendor */
    WBVAL(0xBBBB),                /* idProduct */
    WBVAL(0x0100), /* 1.00 */    /* bcdDevice */
    0x04,                          /* iManufacturer */
    0x20,                          /* iProduct */
    0x42,                          /* iSerialNumber */
    0x01                          /* bNumConfigurations */
};
```

- Thông số của bản mô tả cấu hình:

```
/* USB Configuration Descriptor */
/*AllDescriptors(Configuration,Interface,Endpoint,Class,Vendor*/

const BYTE USB_ConfigDescriptor[] = {
    /* Configuration 1 */
    USB_CONFIGUATION_DESC_SIZE,    /* bLength */
    USB_CONFIGURATION_DESCRIPTOR_TYPE, /* bDescriptorType */
    WBVAL(
        USB_CONFIGUATION_DESC_SIZE +
        USB_INTERFACE_DESC_SIZE +
        USB_ENDPOINT_DESC_SIZE*2
    ),
    0x01,                          /* bNumInterfaces */
};
```

```

    0x01,                                /* bConfigurationValue */
    0x00,                                /* iConfiguration */
    USB_CONFIG_BUS_POWERED /*|*/        /* bmAttributes */
/*USB_CONFIG_REMOTE_WAKEUP*/,
    USB_CONFIG_POWER_MA(100),           /* bMaxPower */
/* Interface 0, Alternate Setting 0, Vendor specific */
    USB_INTERFACE_DESC_SIZE,           /* bLength */
    USB_INTERFACE_DESCRIPTOR_TYPE,     /* bDescriptorType */
    0x00,                               /* bInterfaceNumber */
    0x00,                               /* bAlternateSetting */
    0x02,                               /* bNumEndpoints */
    USB_DEVICE_CLASS_VENDOR_SPECIFIC,  /* bInterfaceClass */
    USB_DEVICE_CLASS_VENDOR_SPECIFIC,  /* bInterfaceSubClass */
    USB_DEVICE_CLASS_VENDOR_SPECIFIC,  /* bInterfaceProtocol */
    0x5C,                               /* iInterface */
/* Endpoint, Bulk In */
    USB_ENDPOINT_DESC_SIZE,            /* bLength */
    USB_ENDPOINT_DESCRIPTOR_TYPE,      /* bDescriptorType */
    USB_ENDPOINT_IN(1),                /* bEndpointAddress 0x81*/
    USB_ENDPOINT_TYPE_BULK,             /* bmAttributes */
    WBVAL(0x0040),                     /* wMaxPacketSize */
    0x20,                               /* 32ms */
    /* bInterval */
/* Endpoint, Bulk Out */
    USB_ENDPOINT_DESC_SIZE,            /* bLength */
    USB_ENDPOINT_DESCRIPTOR_TYPE,      /* bDescriptorType */
    USB_ENDPOINT_OUT(2),                /* bEndpointAddress 0x02 */
    USB_ENDPOINT_TYPE_BULK,             /* bmAttributes */
    WBVAL(0x0040),                     /* wMaxPacketSize */
    0x20,                               /* 32ms */
    /* bInterval */

/* Terminator */
    0                                   /* bLength */
};

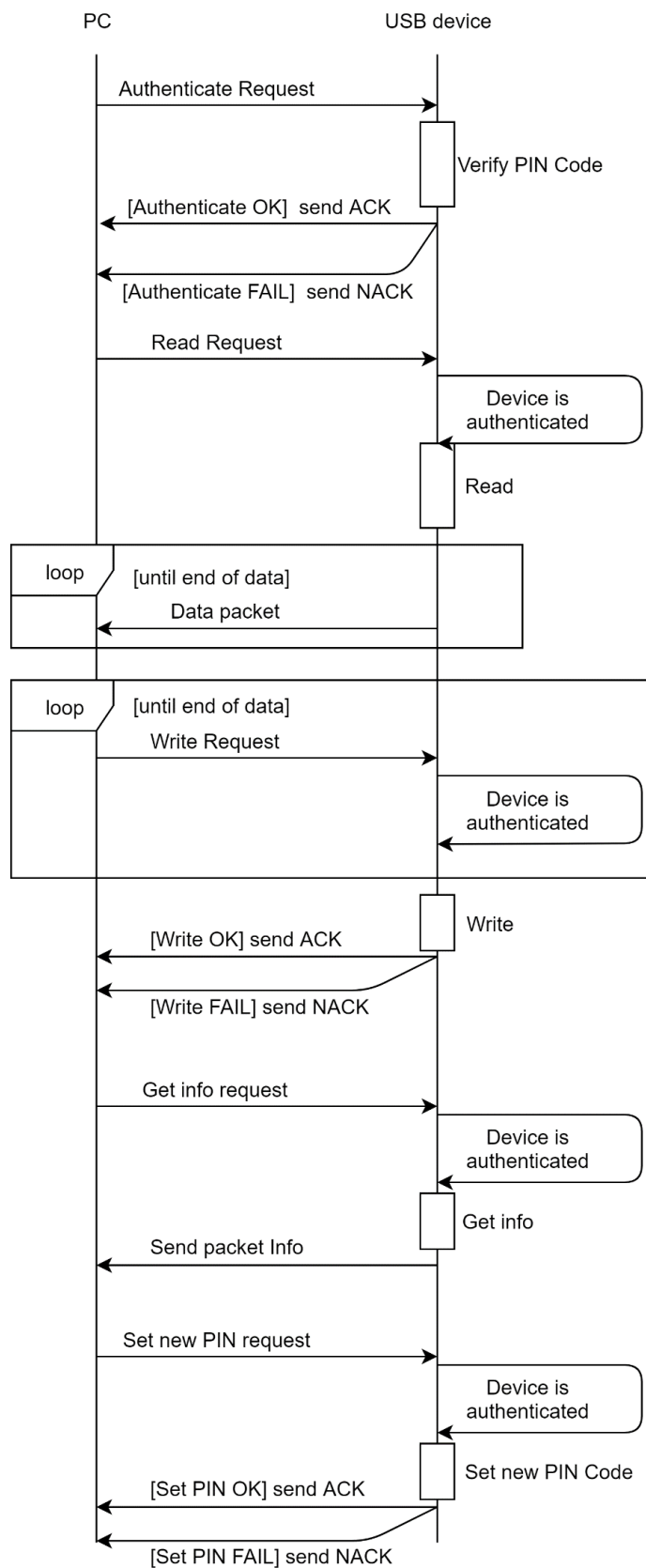
```

Thiết bị được thiết kế có 1 configuration và 1 interface, trong đó interface sử dụng 2 endpoint: endpoint 1 – IN truyền dữ liệu từ thiết bị tới PC và endpoint 2 – OUT nhận dữ liệu từ PC gửi xuống thiết bị. Hai endpoint này cho phép nhận và gửi các gói tin có kích thước tối đa là 64 bytes với kiểu truyền Bulk, dữ liệu truyền và nhận có độ chính xác tuyệt đối và toàn vẹn.

Trong phần mô tả interface chỉ định giá trị `USB_DEVICE_CLASS_VENDOR_SPECIFIC` cho 3 trường `bInterfaceClass`, `bInterfaceSubClass`, `bInterfaceProtocol`, với interface đó thiết bị thuộc lớp thiết bị riêng biệt từ đó có thể tùy biến cách thức giao tiếp của thiết bị với PC, đây cũng là một trong mục tiêu chính của đồ án này.

2.3.2. Nhận và thực thi lệnh từ PC

Biểu đồ quá trình trao đổi dữ liệu giữa thiết bị và PC:



Hình 9: Quá trình trao đổi giữa PC và USB device

Thiết bị sẽ lưu trữ nhiều khối dữ liệu khác nhau có kích thước bất kì tối đa 16KB, dữ liệu này có thể là các khóa riêng của người sở hữu thiết bị, hay khóa bí mật, ... tùy vào ứng dụng của thiết bị. Trước khi sử dụng các tính năng của thiết bị, thiết bị cần xác thực với PC, PC cần gửi gói tin chứa mã PIN đến thiết bị nếu mã PIN đúng sẽ chuyển sang được xác thực và sẵn sàng thực hiện các chức năng khác.

Thiết bị có thể thực hiện các thao tác sau đây:

- Kiểm tra xác thực
- Lấy thông tin về dữ liệu lưu trên thiết bị
- Thiết lập mã PIN Code mới
- Đọc dữ liệu lưu từ thiết bị
- Ghi dữ liệu mới xuống thiết bị

Kịch bản giao tiếp giữa thiết bị và PC:

- PC gửi gói tin chứa mã PIN cần xác thực, đọc dữ liệu từ Flash/EEPROM với tham số là số liệu được tính toán dựa trên mã PIN nhận được. So sánh mã PIN và dữ liệu được đọc lên, nếu đúng thì thiết bị chuyển sang trạng thái đã được xác thực và gửi gói tin ACK, nếu sai gửi gói tin NACK.
- PC gửi gói tin yêu cầu lấy Info, thiết bị kiểm tra trạng thái xác thực, nếu đã xác thực gửi gói tin chứa Info.
- PC gửi gói tin yêu cầu thiết lập mã PIN mới, thiết bị kiểm tra trạng thái xác thực, nếu đã xác thực thì thực hiện việc tạo mã PIN mới, khi đó các dữ liệu cũ sẽ không thể đọc lại được do bảng lưu thông tin về dữ liệu trên thiết bị không còn đúng với mã PIN mới. Nếu quá trình thiết lập mã PIN mới thành công thì gửi gói tin ACK, nếu không thành công thì gửi gói tin NACK.
- PC gửi gói tin yêu cầu ghi dữ liệu xuống thiết bị, thiết bị kiểm tra trạng thái xác thực, nếu đã xác thực thì thiết bị lưu lại gói tin và tiếp tục các gói tin tiếp theo được gửi xuống từ PC cho đến gói tin cuối cùng, sau đó ghép các phần dữ liệu trong các gói tin lại và ghi xuống bộ nhớ Flash/EEPROM. Nếu quá trình ghi xuống bộ nhớ thành công thì gửi gói tin ACK, nếu thất bại gửi gói tin NACK.
- PC gửi gói tin yêu cầu đọc dữ liệu, thiết bị kiểm tra trạng thái xác thực nếu đã xác thực thì thực hiện việc đọc dữ liệu từ bộ nhớ Flash/EEPROM. Nếu quá trình đọc thành công, chia dữ liệu thành các gói tin nhỏ hơn và gửi đi. Nếu quá trình đọc thất bại, gửi gói tin NACK.

Dữ liệu trao đổi qua lại là các gói tin có kích thước 64 bytes - đã được cấu hình trong mô tả cấu hình của giao tiếp USB. Nếu PC gửi một gói tin không đúng định dạng thì thiết bị sẽ bỏ qua và không trả lại thông tin gì.

Cấu trúc chung của một gói tin:

Mã lệnh - 1 byte	Data - 63 bytes
------------------	-----------------

Hình 10: Cấu trúc chung của một gói tin

Có ba thông tin được trao đổi: thông tin thiết bị, thông tin xác thực, thông tin về dữ liệu. Từ đó định nghĩa ra ba loại gói tin sau:

```
typedef struct _AUTHENTICATE_PACKET //Thông tin xác thực
{
    BYTE Username[USERNAME_SIZE];
    BYTE Password[PASSWORD_SIZE];
} AUTHENTICATE_PACKET;

typedef struct _SIGN_MESSAGE //Gói tin đọc ghi khóa riêng
{
    BYTE iCmd;
    BYTE iIndex; //Thứ tự gói tin gửi xuống thiết bị
    USHORT iLenSign;
    USHORT iOffset; //Offset của khóa riêng
    BYTE SignData[MAX_DATA_SIGN_SIZE];
} SIGN_MESSAGE, *PSIGN_MESSAGE;
```

Firmware thiết bị sử dụng một vòng lặp while vô hạn, nhận ngắt khi có gói tin từ PC gửi tới. Firmware nhận các gói tin thông qua hàm ngắt của Endpoint OUT. Mỗi lần nhận được gói tin, trường iCmd sẽ được phân tích để thực hiện thao tác tương ứng. Các gói tin gửi tới PC với trường iCmd với mã mang thông tin phản hồi. Các giá trị của trường iCmd:

```
enum USB_CMD //Định danh gói tin
{
    USB_CMD_READ = 0x01,
    USB_CMD_WRITE,
    USB_CMD_INFO,
    USB_CMD_AUTHENTICATE,
    USB_CMD_ADD,
    USB_CMD_SETPASSWORD,
    USB_CMD_ACK,
    USB_CMD_FAIL
};
```

2.3.3. Tổ chức lưu trữ dữ liệu trên bộ nhớ

Dữ liệu được lưu trữ trên bộ nhớ có thể là flash hoặc EEPROM.

Để đảm bảo dữ liệu lưu trên bộ nhớ được an toàn, firmware tổ chức lưu trữ theo nguyên tắc sau:

- Xen kẽ dữ liệu dư thừa cùng với dữ liệu thật
- Ngẫu nhiên hóa địa chỉ lưu trữ, kết hợp với mã hóa cùng mã PIN

Hai tính chất trên giúp cho nếu thiết bị nếu thất lạc, việc giải mã để lấy được dữ liệu sẽ khó khăn.

Tổ chức lưu trữ dữ liệu được thiết kế như sau:

- Dung lượng bộ nhớ 128 KBytes (trên thiết bị thử nghiệm)
- Bộ nhớ chia thành các block có kích thước 256 bytes
- Tổng cộng có $128 \text{ Kbytes} / 256 \text{ bytes} = 512$ block
- Mỗi gói dữ liệu có thể được chia nhỏ và lưu trên nhiều block khác nhau, các block này móc nối với nhau theo kiểu danh sách liên kết đơn.
- Một block đặc biệt – header block sẽ lưu trữ tất cả các thông tin để quản lý dữ liệu lưu trên bộ nhớ có địa chỉ là $\text{CRC16(PIN)} \bmod 512$.

Cấu trúc của Header block:

PIN Code	Count	Block map	data 0 Size	data 44 Size	data 0 Addr	data 44 Addr
10 bytes	2 bytes	64 bytes	2 bytes	2 bytes	2 bytes	2 bytes

Hình 11: Cấu trúc của header block

Nếu gửi đúng mã PIN sẽ tính toán ra được địa chỉ của header block, đọc block đó và so sánh với mã PIN có trong header block. Nếu gửi sai PIN vẫn tính ra địa chỉ để đọc block, nhưng block đó không phải header block nên không chứa mã PIN đúng nên sẽ không xác thực được mã PIN, thiết bị sẽ không thể sử dụng được.

Trong header block chứa Block map, block map có tác dụng đánh dấu các block đã được sử dụng, dùng trong quá trình tìm kiếm block trống khi ghi data vào bộ nhớ vì các block không ghi theo kiểu tuần tự. Block map có kích thước 64 bytes nên quản lý được $64 \times 8 = 512$ block.

Tiếp theo là thông tin về các gói data gồm 2 mảng mỗi mảng có 45 phần tử, mỗi gói data chiếm 4 bytes, gồm 2 bytes cho kích thước và 2 bytes cho địa chỉ block đầu tiên.

Cấu trúc của data block:

Next Addr	Flag	Data
2 bytes	2 bytes	252 bytes

Hình 12: Cấu trúc của data block

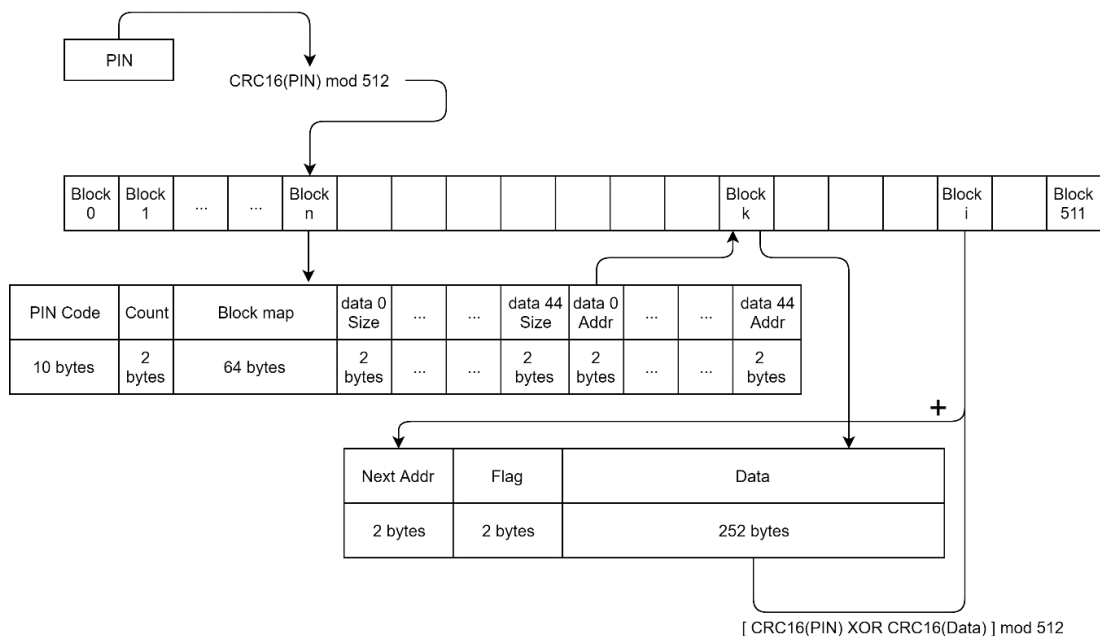
Mỗi data block chứa 252 bytes dữ liệu thực sự của các gói data. Trường Next Addr bằng địa chỉ của block tiếp theo cộng với **(CRC16(PIN) XOR CRC16(Data_i)) mod 512**.

Công thức để xác định địa chỉ của block tiếp theo khi cần đọc dữ liệu:

$$\text{Addr Block}_{i+1} = \text{Next Addr}_i - (\text{CRC16(PIN)} \text{ XOR } \text{CRC16(Data}_i)) \bmod 512$$

Như vậy nếu không biết mã PIN sẽ không tìm ra được địa chỉ của block tiếp theo.

Tổng quát lại cách thức tổ chức lưu trữ dữ liệu trên bộ nhớ:



Hình 13: Tổng quát cách thức tổ chức lưu trữ dữ liệu trên bộ nhớ.

2.4. Xây dựng firmware

Phần mềm được sử dụng là Keil uVision 4 ARM-MDK - bộ công cụ lập trình và gỡ rối cho việc viết firmware chạy trên chip.

Sử dụng thư viện RL-USB để xây dựng firmware và dễ dàng chuyển đổi trên các vi điều khiển **ARM** mà thư viện hỗ trợ.

Trong bộ phát triển uVision của Keil, có thư viện RL-USB hỗ trợ việc xây dựng firmware cho các thiết bị USB. Sử dụng thư viện này làm khuôn mẫu để phát triển firmware cho thiết bị thử nghiệm.

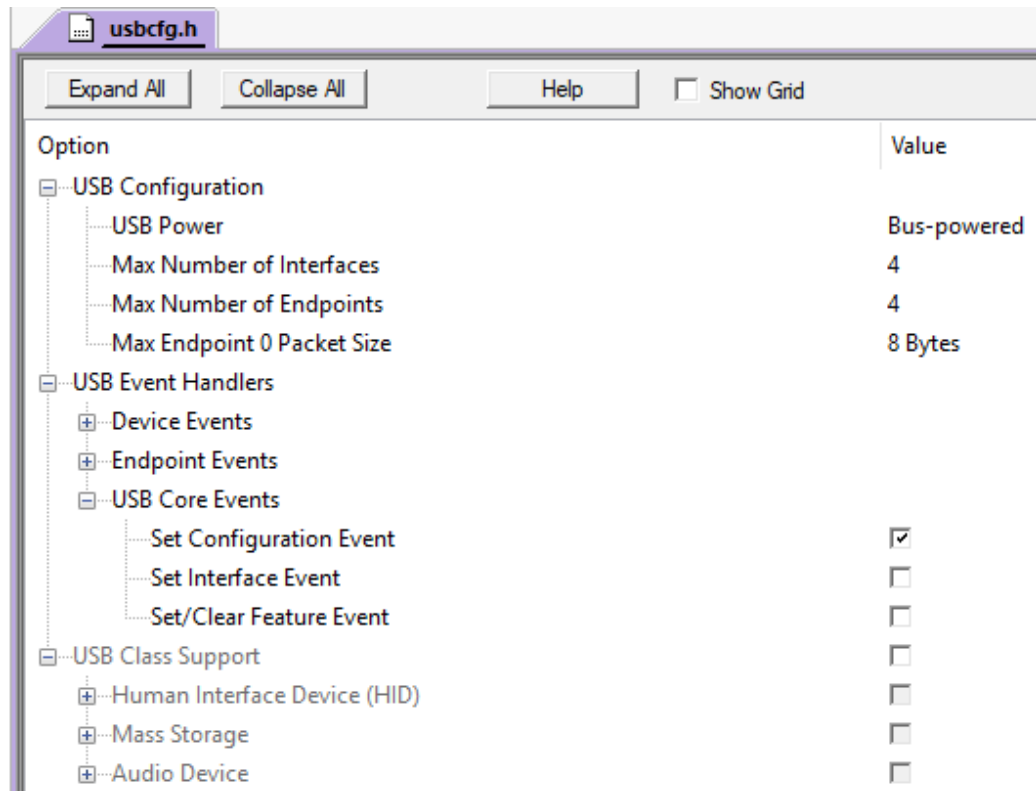
RL-USB là bộ thư viện của Keil giúp đơn giản hóa tối đa việc lập trình giao diện USB cho firmware. Dễ dàng thêm giao tiếp USB vào thiết bị đang có hiện tại hoặc

dựa trên 1 sản phẩm của **ARM**, chẳng hạn như bàn phím, máy ảnh kỹ thuật số, và máy nghe nhạc MP3. Nó dễ dàng cho các ứng dụng tương tác mạnh mẽ với các máy tính hỗ trợ thiết bị **USB**. **RL-USB** bao gồm trình điều khiển phần cứng, nhân **USB** và lớp chức năng **USB** cụ thể có thể dễ dàng cấu hình từ các ứng dụng.

2.4.1. Cấu hình thông số USB và xử lý ngắt trên endpoint

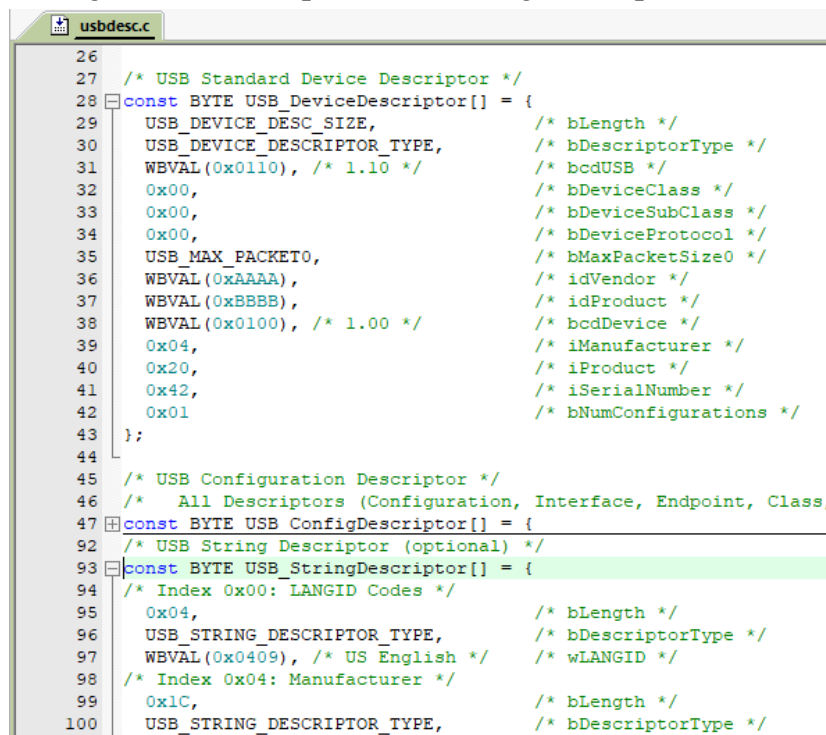
Thông số giao diện USB được cấu hình thông qua các file sau:

- `usbcfg.h` cho phép tùy chỉnh USB Configuration, USB Event handlers, USB support class.



Hình 14: Cấu hình USB Configuration

- usbdesc.c chứa các bản mô tả bao gồm USB Device Descriptor, USB Configuration Descriptor, USB String Descriptor.



```

26
27 /* USB Standard Device Descriptor */
28 const BYTE USB_DeviceDescriptor[] = {
29     USB_DEVICE_DESC_SIZE,          /* bLength */
30     USB_DEVICE_DESCRIPTOR_TYPE,     /* bDescriptorType */
31     WBVAL(0x0110), /* 1.10 */      /* bcdUSB */
32     0x00,                           /* bDeviceClass */
33     0x00,                           /* bDeviceSubClass */
34     0x00,                           /* bDeviceProtocol */
35     USB_MAX_PACKET0,               /* bMaxPacketSize0 */
36     WBVAL(0xAAAA),                /* idVendor */
37     WBVAL(0xBBBB),                /* idProduct */
38     WBVAL(0x0100), /* 1.00 */      /* bcdDevice */
39     0x04,                           /* iManufacturer */
40     0x20,                           /* iProduct */
41     0x42,                           /* iSerialNumber */
42     0x01                           /* bNumConfigurations */
43 };
44
45 /* USB Configuration Descriptor */
46 /* All Descriptors (Configuration, Interface, Endpoint, Class,
47 const BYTE USB_ConfigDescriptor[] = {
48     /* USB String Descriptor (optional) */
49     const BYTE USB_StringDescriptor[] = {
50         /* Index 0x00: LANGID Codes */
51         0x04,                       /* bLength */
52         USB_STRING_DESCRIPTOR_TYPE, /* bDescriptorType */
53         WBVAL(0x0409), /* US English */ /* wLANGID */
54         /* Index 0x04: Manufacturer */
55         0x1C,                       /* bLength */
56         USB_STRING_DESCRIPTOR_TYPE, /* bDescriptorType */

```

Hình 15: Cấu trúc USB_DeviceDescriptor

Các hàm xử lý ngắt trên endpoint có thể được tùy chỉnh trong file usbuser.c.

Thiết bị sử dụng endpoint 1 – IN để gửi dữ liệu, endpoint 2 – OUT để nhận dữ liệu.

Trong hàm xử lý ngắt của 2 endpoint này được xử lý như sau:

- Với endpoint 1 – IN

```

void USB_EndPoint1 (DWORD event) {
    switch(event)
    {
        case USB_EVT_IN:
            WReady = 1;
            break;
        default:
            break;
    }
}

```

Mỗi khi có yêu cầu đọc dữ liệu từ PC hàm ngắt của endpoint 1 thực hiện việc set cờ WReady = 1 – tức là PC đang yêu cầu đọc dữ liệu, khi thiết bị muốn gửi dữ liệu tới PC phải liên tục kiểm tra cờ này trước khi gửi, khi nào cờ WReady = 1 thì quá trình mới bắt đầu, sau khi gửi gói tin kết thúc cờ WReady lại được set bằng 0.

- Với endpoint 2 – OUT

```

void USB_EndPoint2 (DWORD event) {

    if (RReady==0)
    {
        USB_ReadEP(0x02, (BYTE*) &aPacket);
        RReady = 1;
    }
    else
    {
        USB_SetStallEP(0x02);
    }
}

```

Mỗi khi PC gửi dữ liệu xuống thiết bị hàm ngắt của endpoint 2 thực hiện việc kiểm tra cờ RReady. Nếu cờ này bằng 0 thì đọc dữ liệu mà PC gửi xuống rồi lưu lại vào biến aPacket và set cờ RReady bằng 1. Trong chương trình chính liên kiểm tra cờ này, nếu cờ bằng 1 báo hiệu rằng có gói tin mới được gửi đến, chương trình thực hiện set cờ Rready về lại bằng 0 để thông báo sẵn sàng nhận gói tin mới, sau xử lý gói tin vừa được lưu vào biến aPacket.

2.4.2. Chức năng nhận và thực thi lệnh từ PC

Đây là chức năng chính của firmware gồm một vòng lặp vô hạn liên tục kiểm tra cờ RReady. Nếu cờ này bằng 1 tức có gói tin mới được gửi tới và lưu trong biến aPacket, chương trình kiểm tra trường iCmd (mã lệnh của gói tin) để thực hiện xử lý tương ứng.

```

iState = 0;
while (1)
{
    if (RReady)
    {
        RReady = 0;
        switch (aPacket.iCmd)
        {
            case USB_CMD_AUTHENTICATE:
                // Code handle packet Authenticate
                // set value iState 0 or 1
                break;

            case USB_CMD_INFO:
                if (iState == 0) continue;
                // Code handle packet Info
                break;

            case USB_CMD_SETPASSWORD:
                if (iState == 0) continue;
                // Code handle packet Set password
                break;

            case USB_CMD_WRITE:

```

```

        if (iState == 0) continue;
        // Code handle packet write
        break;

    case USB_CMD_READ:
        if (iState == 0) continue;
        // Code handle packet read
        break;

    default:
        break;
    }
}
}

```

- Biến iState lưu trạng thái xác thực của thiết bị, được khởi tạo bằng 0 – thiết bị chưa được xác thực
- Vòng lặp vô hạn While(1){} liên tục kiểm tra cờ RReady, nếu cờ này bằng 1 thì thực hiện xử lý gói tin vừa được gửi tới thiết bị.
- Set cờ RReady bằng 0 để thông báo gói tin cũ đã được xử lý và có thể nhận các gói tin mới.
- Kiểm tra trường iCmd để thực hiện xử lý tương ứng:
 - USB_CMD_AUTHENTICATE: tính toán địa chỉ bộ nhớ cần đọc dựa trên mã PIN gửi tới, đọc bộ nhớ từ địa chỉ được tính toán, so khớp mã PIN gửi tới và mã PIN đọc được từ bộ nhớ, nếu đúng thì set cờ trạng thái xác thực iState bằng 1, ngược lại set cờ bằng 0. Cuối cùng gửi gói tin phản hồi tới PC.
 - USB_CMD_INFO: kiểm tra cờ trạng thái xác thực, nếu đã xác thực thì gửi gói tin Info tới PC.
 - USB_CMD_SETPASSWORD: kiểm tra cờ trạng thái xác thực, nếu đã xác thực thì thực hiện thiết lập mã PIN mới, set cờ iState bằng 0 (chuyển trạng thái thiết bị thành chưa được xác thực) nếu thiết lập thành công, giữ nguyên trạng thái xác thực và PIN cũ nếu thiết lập thất bại. Gửi phản hồi tới PC về kết quả của việc thiết lập mã PIN mới.
 - USB_CMD_WRITE: kiểm tra cờ trạng thái xác thực, nếu đã xác thực thì lưu gói tin lại vào biến tạm thời. Kiểm tra nếu PC đã gửi đủ các gói tin thì gộp các gói tin lại rồi lưu xuống bộ nhớ. Gửi phản hồi tới PC về kết quả của việc ghi dữ liệu xuống bộ nhớ.
 - USB_CMD_READ: kiểm tra cờ trạng thái xác thực, nếu đã xác thực thì đọc từ bộ nhớ dựa vào thông tin trong gói tin gửi đến. Gửi lại dữ liệu đọc được từ bộ nhớ tới PC nếu đọc thành công, ngược lại gửi gói tin phản hồi việc đọc thất bại tới PC.

2.4.3. Chức năng lưu trữ dữ liệu trên bộ nhớ

Chức năng này gồm 2 hàm đọc và ghi khối dữ liệu với bộ nhớ có nguyên mẫu hàm như sau:

```
BOOL ReadSignature(BYTE iIndex, BIO_SIGNATURE * pBio)
BOOL WriteSignature(BYTE iIndex, BIO_SIGNATURE * pBio)
```

Hai tham số của hàm là iIndex(index của khối dữ liệu lưu trên bộ nhớ, tối đa lưu được 31 khối dữ liệu khác nhau) và con trỏ tới khối dữ liệu cần ghi hoặc tới địa chỉ để lưu dữ liệu đọc được.

Hai hàm giao tiếp với bộ nhớ có thể khác nhau nếu bộ nhớ là flash hoặc EEPROM, tuy nhiên có nguyên mẫu hàm giống nhau như sau:

```
EEPROM_Read((BYTE*)&Block,BLOCK_SIZE, Addr);
EEPROM_Write((BYTE*)&Block,BLOCK_SIZE, Addr);
```

Ba tham số của hàm là con trỏ tới Block cần ghi hoặc tới Block để lưu dữ liệu đọc được, kích thước dữ liệu muốn đọc hoặc ghi và địa chỉ cần đọc hoặc ghi.

- Hàm đọc khối dữ liệu:

```
BOOL ReadSignature(BYTE iIndex, BIO_SIGNATURE * pBio)
{
    USHORT iBytesRead = 0;
    USHORT TmpChecksum;
    USHORT NextBase;

    if (iIndex >= 31)
        return FALSE;

    if (RomHeader.aSignSizes[iIndex]==0)
        return FALSE;
    pBio->iSize = RomHeader.aSignSizes[iIndex];

    EEPROM_Read((BYTE*)&Block,BLOCK_SIZE,
RomHeader.aSignAddrs[iIndex] << 8);
    memcpy(pBio->aData,Block.Data,PAYLOAD_SIZE);
    iBytesRead += PAYLOAD_SIZE;

    while (iBytesRead < pBio->iSize)
    {
        if (Block.NextBlockOffset == 0xFFFF) // Last data block
in signature
            return FALSE;
        TmpChecksum
Checksum16((USHORT*)Block.Data,PAYLOAD_SIZE);
        NextBase = Block.NextBlockOffset - (iPinChecksum ^
TmpChecksum)%BLOCK_COUNT; // Calculate next block addresss
        EEPROM_Read((BYTE*)&Block,BLOCK_SIZE, NextBase << 8);
```

```

        memcpy(pBio->aData+iBytesRead,Block.Data,PAYLOAD_SIZE);
        iBytesRead += PAYLOAD_SIZE;
    };
    return TRUE;
}

```

Hàm này sử dụng tham số *iIndex* để lấy được địa chỉ của block đầu tiên được lưu trong Header Block. Từ địa chỉ khối block đầu tiên của khối dữ liệu, đọc từng khối một rồi tính toán địa chỉ của block tiếp theo, rồi đọc các block tiếp theo cho khi đến block cuối.

- Hàm ghi khối dữ liệu

```

BOOL WriteSignature(BYTE iIndex, BIO_SIGNATURE * pBio)
{
    USHORT i,iBytesWritten = 0;
    USHORT TmpBase,TmpChecksum,NextBase;
    BOOL bFound = FALSE;

    if (iIndex >= 31)
        return FALSE;

    TmpBase = iHeaderBase >> 8;
    if (RomHeader.aSignSizes[iIndex] == 0)
        RomHeader.iCount++;
    RomHeader.aSignSizes[iIndex] = pBio->iSize;
    for (i = TmpBase + 1; i < BLOCK_COUNT; i++)
        if (!(RomHeader.aMap[i>>3] & (1<<(7-i%8))))
        {
            RomHeader.aSignAddrs[iIndex] = i;
            RomHeader.aMap[i>>3] |= (1<<(7-i%8));
            TmpBase = i;
            bFound = TRUE;
            break;
        };
    if (bFound==FALSE)
        for (i = 0; i < TmpBase; i++)
            if (!(RomHeader.aMap[i>>3] & (1<<(7-i%8))))
            {
                RomHeader.aSignAddrs[iIndex] = i;
                RomHeader.aMap[i>>3] |= (1<<(7-i%8));
                TmpBase = i;
                bFound = TRUE;
                break;
            };
    if (bFound == FALSE)
        return FALSE;

    iBytesWritten = 0;
    do
    {

```

```

memcpy(Block.Data, pBio->aData+iBytesWritten, PAYLOAD_SIZE);
Block.Flags = 0xFFFF;
if (iBytesWritten + PAYLOAD_SIZE >= pBio->iSize)
    Block.NextBlockOffset = 0xFFFF;
else
{
    TmpChecksum = Checksum16((USHORT*)Block.Data, PAYLOAD_SIZE);
    TmpChecksum ^= iPinChecksum;
    NextBase = TmpChecksum % BLOCK_COUNT;
    bFound = FALSE;
    for (i = NextBase; i < BLOCK_COUNT; i++)
        if (!(RomHeader.aMap[i]>>3] & (1<<(7-i%8))))
        {
            bFound = TRUE;
            RomHeader.aMap[i]>>3] |= (1<<(7-i%8));
            break;
        };
    if (bFound==FALSE)
    for (i=0; i<NextBase; i++)
    if (!(RomHeader.aMap[i]>>3] & (1<<(7-i%8))))
    {
        bFound = TRUE;
        RomHeader.aMap[i]>>3] |= (1<<(7-i%8));
        break;
    };
    if (bFound == FALSE)
        return FALSE;

    Block.NextBlockOffset = NextBase + i;
}

EEPROM_Write((BYTE*)&Block, BLOCK_SIZE, TmpBase<<8);
iBytesWritten += PAYLOAD_SIZE;
TmpBase = i; // Moving to next data block
} while (iBytesWritten < pBio->iSize);
EEPROM_Write((BYTE*)&RomHeader, BLOCK_SIZE, iHeaderBase);
return TRUE;
}

```

Hàm này tìm một block trống để lưu block đầu tiên của khối dữ liệu, lưu địa chỉ của block đầu tiên này vào Header Block để phục vụ việc đọc sau này. Với các block còn lại lưu theo kiểu danh sách móc nối, địa chỉ của block sau được tính toán liên quan với dữ liệu của block trước và mã PIN.

CHƯƠNG 3: GIAO TIẾP THIẾT BỊ NHÚNG USB TRÊN WINDOWS

Phần mềm điều khiển trên PC làm nhiệm vụ giao tiếp với phần sụn trên thiết bị để thực hiện các thao tác trao đổi thông tin. Phần mềm được chia thành hai lớp con:

- Trình điều khiển thiết bị (driver).
- Thư viện API .

3.1. Trình điều khiển thiết bị

Để ứng dụng tầng trên có thể trao đổi dữ liệu với firmware thiết bị, hệ điều hành cần phải nạp trình điều khiển (Driver) tương ứng và cung cấp giao diện cho thiết bị. Do đây là thiết bị tự xây dựng nên ta phải cung cấp trình điều khiển cho hệ điều hành.

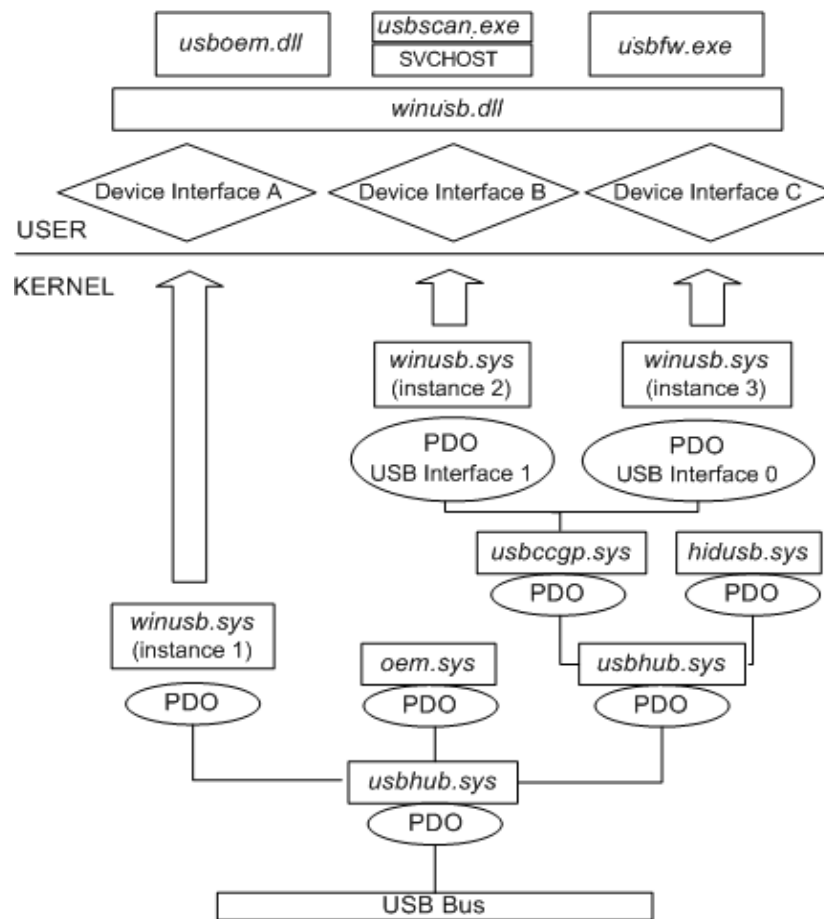
Có hai phương pháp tiếp cận để xây dựng trình điều khiển:

- Xây dựng một trình điều khiển riêng chạy ở nhân hệ điều hành, trình điều khiển sẽ liên lạc với phần cứng usb host và trao đổi thông tin với ứng dụng qua bus USB.
- Sử dụng một trình điều khiển chung cho mọi thiết bị USB. Cấu hình lại driver để hỗ trợ thiết bị. Xây dựng thư viện liên kết động chạy ở chế độ người dùng và điều khiển trình điều khiển này trao đổi thông tin với ứng dụng.

Phương pháp thứ nhất tương đối phức tạp và thích hợp với các ứng dụng cần tốc độ cao, nhưng dễ sai sót, có thể làm dump hệ điều hành (Blue Screen of Death - BSOD). Phương pháp thứ hai đơn giản và dễ triển khai hơn. Chúng tôi đã quyết định chọn phương án thứ hai. Trình điều khiển chung được sử dụng là WinUSB cho các hệ điều hành của Microsoft.

WinUSB hỗ trợ ứng dụng trao đổi dữ liệu với các endpoint của thiết bị, từ đó có thể xây dựng giao thức riêng để truyền thông tin. WinUSB gồm hai thành phần chính:

- Winusb.sys là driver chạy ở nhân hệ điều hành, bên trên các driver điều khiển phần cứng USB.
- Winusb.dll là thư viện liên kết động chạy ở chế độ người dùng, cung cấp giao diện tương tác với Winusb.sys.



Hình 16: Kiến trúc WinUSB [7]

Công việc còn lại là cấu hình WinUSB để hỗ trợ thiết bị. Các thành phần tối thiểu để cài đặt WinUSB driver gồm có:

- Thư viện WinUSB co-installer: WinUSBCoInstaller.dll.
- Thư viện KMDF (Kernel Mode Driver Framework) co-installer: WdfCoInstallerxxxxx.dll với xxxx là phiên bản của KMDF. Thí dụ WdfCoInstaller01005.dll. WinUSB hoạt động dựa vào vào KMDF do vậy trong bộ cài cần có KMDF.
- File INF mô tả ứng dụng.

Từ file INF chuẩn của Microsoft, cần sửa những thông tin sau:

- Trong mục [MyDevice_WinUSB.NTx86] sửa phần VID và PID cho phù hợp với định danh nhà sản xuất và định danh sản phẩm. Trong trường hợp thiết bị đó là 0xAAAA và 0xB BBB.
- Trong mục [Dev_AddReg] sửa GUID bằng GUID của thiết bị. Ở đây GUID cho mỗi thiết bị là duy nhất, có thể sinh trực tuyến hoặc sử dụng công cụ sinh GUID của Microsoft đi kèm với WDF.
- Trong mục [Strings] sửa các chuỗi cho phù hợp với thiết bị.

Nội dung file INF trong trình điều khiển thiết bị

```

;
; MyDriver.inf
;
; Installs WinUsb
;

[Version]
Signature = "$Windows NT$"
Class      = USB
ClassGUID  = {36FC9E60-C465-11CF-8056-444553540000}
Provider   = %ManufacturerName%
DriverVer=
CatalogFile=MyDriver.cat

; ===== Manufacturer/Models sections =====

[Manufacturer]
%ManufacturerName% = Standard,NT$ARCH$

[Standard.NT$ARCH$]
%DeviceName% =USB_Install, USB\VID_AAAA&PID_BBBB

; ===== Installation =====

[USB_Install]
Include=winusb.inf
Needs=WINUSB.NT

[USB_Install.Services]
Include=winusb.inf
AddService=WinUsb,0x00000002,WinUsb_ServiceInstall

[WinUsb_ServiceInstall]
DisplayName      = %WinUsb_SvcDesc%
ServiceType      = 1
StartType        = 3
ErrorControl      = 1
ServiceBinary    = %12%\WinUSB.sys

[USB_Install.HW]
AddReg=Dev_AddReg

[Dev_AddReg]
; By default, USBDevice class uses iProduct descriptor to name
thedevice in
; Device Manager on Windows 8 and higher.

```

```

; Uncomment for this device to use %DeviceName% on Windows 8
andhigher:
;HKR,,FriendlyName,,%DeviceName%
HKR,,DeviceInterfaceGUIDs,0x10000,"{6d22450a-fe34-46d2-
b01965dbalae0aaa}"

[USB_Install.CoInstallers]
AddReg=CoInstallers_AddReg
CopyFiles=CoInstallers_CopyFiles

[CoInstallers_AddReg]
HKR,,CoInstallers32,0x00010000,"WdfCoInstaller$KMDFCOINSTALLERVERS
ION$.dll,WdfCoInstaller"

[CoInstallers_CopyFiles]
WdfCoInstaller$KMDFCOINSTALLERVERSION$.dll

[DestinationDirs]
CoInstallers_CopyFiles=11

; ===== Source Media Section =====

[SourceDisksNames]
1 = %DiskName%

[SourceDisksFiles]
WdfCoInstaller$KMDFCOINSTALLERVERSION$.dll=1

; ===== Strings =====

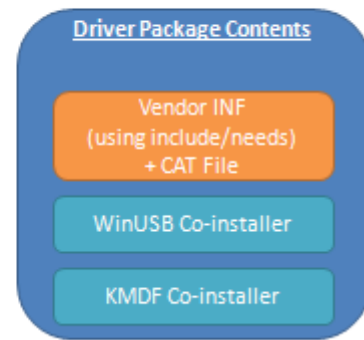
[Strings]
ManufacturerName="SOICT Research"
;ClassName="Universal Serial Bus devices"
DiskName="USBToken Installation Disc"
WinUsb_SvcDesc="USBToken v2.0"
DeviceName="USBToken v2.0"
REG_MULTI_SZ = 0x00010000

```

Các thành phần trong trình điều khiển thiết bị

Driver Package Contents

- The KMDF co-installer and WinUSB co-installer must come from the same version of the WDK.
- The co-installers must come from the latest version of the WDK to support all of the latest Windows releases.
- All contents of the package should be digitally signed with a Winqual release signature.

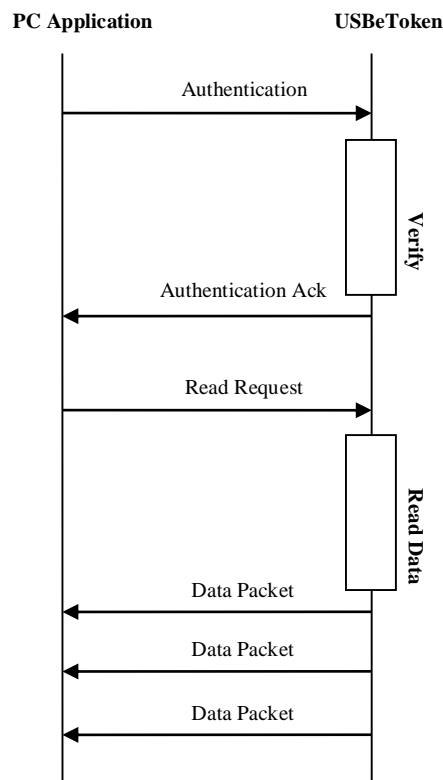


Hình 17. WinUSB driver [8]

3.2. Thiết kế giao thức

Mỗi thiết bị sẽ lưu trữ dữ liệu khóa riêng cho chủ thể. Mỗi khóa riêng có thể có kích thước bất kỳ, tối đa 16KB.

Trước khi PC có thể thực hiện trao đổi dữ liệu, nó cần được xác thực trước. PC sẽ gửi mã PIN đến thiết bị, nếu mã pin đúng, thiết bị sẽ chuyển sang trạng thái sẵn sàng gửi dữ liệu cá nhân về PC. Biểu đồ trao đổi dữ liệu giữa PC và thiết bị



Hình 3.6: Quá trình trao đổi giữa PC và thiết bị

Thiết bị có thể thực hiện được các thao tác sau:

- Xác thực mã pin.
- Đọc một thông tin khóa riêng

- Ghi một thông tin khóa riêng
- Thiết lập mã pin mới.

Thiết bị có nhiều trạng thái, tùy theo số lượng khóa riêng được lưu trữ trên nó.

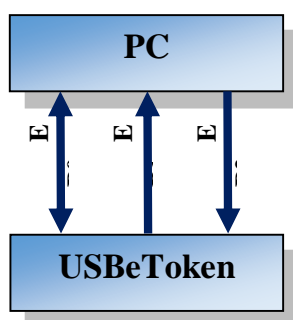
Quá trình đọc khóa riêng diễn ra như sau:

- PC gửi mã PIN để thiết bị xác thực.
- Thiết bị sẽ nhận mã PIN được gửi, thực hiện đọc từ bộ nhớ, so sánh với mã PIN vừa gửi. Nếu đúng thiết bị sẽ chuyển sang trạng thái sẵn sàng và gửi gói tin ACK về cho ứng dụng.
- PC gửi yêu cầu đọc khóa riêng. Thiết bị sẽ đọc dữ liệu và gửi trả lại PC.
- PC tiến hành sử dụng khóa riêng, thực hiện các yêu cầu tiếp theo như ký file sử dụng khóa riêng đã lưu.
- ...

Quá trình ghi các khóa riêng diễn ra tương tự như đọc. Các khóa riêng sẽ lần lượt được ghi vào bộ nhớ theo thứ tự được ghi.

Quá trình xóa các khóa riêng được thực hiện qua việc thiết lập lại mã PIN. Mỗi khi người dùng thiết lập lại mã PIN, toàn bộ các khóa riêng cũ sẽ bị xóa (vị trí lưu trữ các khóa riêng phụ thuộc vào mã PIN và do đó nó không còn chính xác khi thiết lập PIN mới).

Thiết bị và các ứng dụng tầng trên sẽ trao đổi dữ liệu thông qua hai endpoint kiểu bulk: endpoint 1 cho chiều từ thiết bị đến PC và endpoint 2 cho chiều từ PC đến thiết bị. Endpoint 0 là endpoint đặc biệt để liệt kê và khởi tạo thiết bị. Dữ liệu trao đổi trên bus sẽ là các gói tin kích thước 64 byte. Nếu PC gửi một gói tin không hợp ngữ cảnh, thiết bị sẽ bỏ qua và không gửi trả bất kỳ thông tin nào.



Hình 3.7. Các Endpoint sử dụng trong USBToken

Cấu trúc chung của một gói tin:

Lệnh(1 byte)	Dữ liệu(63 byte)
---------------------	-------------------------

Table 0-1. Cấu trúc chung của một gói tin

Có ba loại thông tin gửi nhận trên bus: thông tin xác thực, thông tin thiết bị và thông tin về khóa riêng. Tương ứng sẽ có ba cấu trúc khác nhau cho mỗi loại thông tin đó.

```
enum USB_CMD      //Định danh gói tin
{
    USB_CMD_READ = 0x01,
    USB_CMD_WRITE,
    USB_CMD_INFO,
    USB_CMD_AUTHENTICATE,
    USB_CMD_ADD,
    USB_CMD_SETPASSWORD,
    USB_CMD_ACK,
    USB_CMD_FAIL
};

typedef struct _AUTHENTICATE_PACKET    //Thông tin xác thực
{
    BYTE  Username[USERNAME_SIZE];
    BYTE  Password[PASSWORD_SIZE];
} AUTHENTICATE_PACKET;

typedef struct _SIGN_MESSAGE           //Gói tin đọc ghi khóa riêng
{
    BYTE  iCmd;
    BYTE  iIndex; //Thứ tự gói tin gửi xuống thiết bị
    USHORT iLenSign;
    USHORT iOffset; //Offset của khóa riêng
    BYTE  SignData[MAX_DATA_SIGN_SIZE];
} SIGN_MESSAGE , *PSIGN_MESSAGE;
```

3.3. Thư viện API

Thư viện API (USB DLL) sẽ giao tiếp với thiết bị và các ứng dụng tầng cao hơn của hệ thống.

Thư viện API cung cấp các hàm sau:

- Lấy thông tin path device đăng ký trong PC
- Lấy thông tin cấu hình thiết bị
- Xác thực một thiết bị
- Đọc một khóa riêng từ thiết bị.
- Ghi một khóa riêng vào thiết bị.
- Thiết lập lại mã PIN.

Để thực hiện được điều đó, thư viện sẽ tương tác với trình điều khiển thiết bị WinUSB đã thiết kế ở trên.

Biến toàn cục lưu các thông tin cần thiết về thiết bị:

```
typedef struct _DEVICE_DATA {
    BOOL bIsHandlesOpen;
```

```

WINUSB_INTERFACE_HANDLE hWinUSBInterfaceHandle;
HANDLE                   hFileDeviceHandle;
TCHAR                   DevicePath[MAX_PATH];
UCHAR                   uBulkInPipeId;
UCHAR                   uBulkOutPipeId;
UCHAR                   uInterruptPipeId;
} DEVICE_DATA, *PDEVICE_DATA;

```

Nguyên mẫu các hàm thư viện được thiết kế như sau:

- Lấy thông tin path device đã đăng ký

```

HRESULT
OpenDevice(
    _Out_ PDEVICE_DATA DeviceData,
    _Out_opt_ PBOOL FailureDeviceNotFound
)

```

Trong đó: - Hàm trả về *S_OK* nếu thành công.
 - *DeviceData* trả về một số thông tin *DEVICE_DATA*
 - *FailureDeviceNotFound* trả về *TRUE* nếu không tìm thấy Device nào.

- Lấy thông tin về một thiết bị

```

BOOL GetConfigDevice();

```

Trong đó: - Hàm trả về *TRUE* nếu thông tin lấy thành công.
 - Hàm sẽ lấy các thông tin cần thiết còn lại trong struct *DEVICE_DATA*

- Xác thực một thiết bị

```

BOOL AuthenticateDevice(PCHAR szPassword, PCHAR szUserName)

```

Trong đó: - Hàm trả về *TRUE* nếu xác thực thành công.
 - *szUsername* và *szPassword* là tên và mật khẩu.

- Đọc một khóa riêng

```

BOOL ReadSignature(PBYTE pSignature, PUSHORT usSignSizeTransferred)

```

Trong đó: - Hàm trả về *TRUE* nếu thành công
 - *pSignature* là kết quả khóa riêng đọc được
 - *usSignSizeTransferred* là số byte đã đọc được

- Ghi một khóa riêng

```

BOOL WriteSignature(PBYTE pSignature, USHORT usSignSize);

```

Trong đó: - Hàm trả về *TRUE* nếu thành công
 - *pSignature* là nội dung khóa riêng muốn ghi
 - *usSignSize* là kích thước gói tin để ghi xuống thiết bị.

- Thiết lập lại mã PIN

```

BOOL SetPasswordDevice(PCHAR szPassword, PCHAR szUserName)

```

Trong đó: - Hàm trả về *TRUE* nếu thiết lập thành công
 - *szUsername* và *szPassword* là tên và mật khẩu.

Để thực hiện được các hàm trên, USBTokenManager cần sử dụng các hàm sau của WinUSB.

- Mở một thiết bị

```
hDev = CreateFile(devicePath,
    GENERIC_WRITE | GENERIC_READ,
    FILE_SHARE_WRITE | FILE_SHARE_READ,
    NULL,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
    NULL);
```

Với hDev là handle của thiết bị, devicePath là đường dẫn tới thiết bị

- Khởi tạo handle của WinUSB

```
bResult = WinUsb_Initialize(deviceHandle, &usbHandle);
```

với usbHandle là handle của WinUSB

- Đọc dữ liệu từ một PIPE

```
bResult = WinUsb_ReadPipe(usbHandle,
    iPipeID,
    szBuffer,
    24,
    &bytesRead,
    NULL);
```

Với usbHandle là handle có được từ bước 2, iPipeID là số nguyên chỉ PIPE IN của thiết bị, trong trường hợp này là 0x81

- Ghi dữ liệu ra PIPE

```
bResult = WinUsb_WritePipe(usbHandle,
    iPipeID,
    szBuffer,
    24,
    &bytesWritten,
    NULL);
```

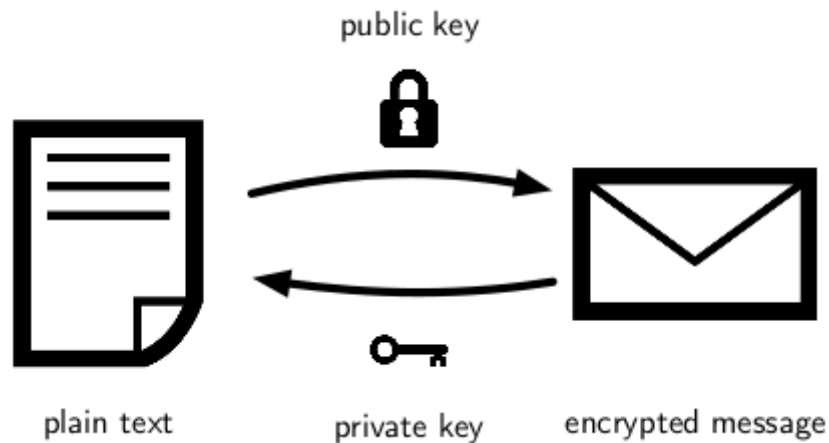
Với usbHandle là WinUSB handle có từ bước 2, iPipeID là số nguyên chỉ PIPE OUT của thiết bị, trong trường hợp này là 0x02.

CHƯƠNG 4: ỨNG DỤNG DỊCH VỤ CHỮ KÝ SỐ

4.1. Cơ sở lý thuyết

4.1.1. Tìm hiểu mật mã hóa khóa công khai

Mật mã hóa khóa công khai hay còn được gọi với một cái tên khác là mã hóa khóa bất đối xứng (Public Key Cryptography), nó được thiết kế sao cho khóa sử dụng trong quá trình mã hóa khác biệt với khóa được sử dụng trong quá trình giải mã. Hơn thế nữa, khóa dùng trong quá trình giải mã không thể được tính toán hay suy luận từ khóa dùng để mã hóa và ngược lại, tức là hai khóa này có quan hệ với nhau về mặt toán học nhưng không thể suy diễn được ra nhau. Thuật toán này được gọi là Public-Key bởi vì khóa dùng cho việc mã hóa được công khai cho tất cả mọi người. Một người hoàn toàn xa lạ có thể dùng khóa này để mã hóa dữ liệu nhưng chỉ duy nhất người mà có khóa giải mã tương ứng mới có thể đọc được dữ liệu mà thôi. Do đó trong thuật mã hóa này Encryption key được gọi là Public key còn Decryption Key được gọi là Private key.



Hình 18. Quy trình sử dụng mật mã hóa khóa công khai [9]

Tại sao lại có sự xuất hiện của Mã hóa khóa công khai? Như đã trình bày với các bạn ở trên, thuật toán mã hóa khóa riêng tuy có tốc độ thực hiện rất nhanh nhưng nó có một số nhược điểm như sau:

- Khóa phải được trao đổi theo một kênh bí mật.
- Nếu mất khóa thì thông tin hoàn toàn có thể bị lấy cắp hoặc giả mạo.
- Khóa cần phải thay đổi một cách định kì
- Khi số lượng người dùng tăng lên thì số lượng khóa được sử dụng cũng tăng lên.

Vậy là điểm yếu của thuật toán mã hóa khóa riêng nằm ở khâu quản lý khóa thế nào cho hợp lý. Mã hóa khóa công khai ra đời đã giải quyết được vấn đề này. Hình minh họa ở trên cho chúng ta thấy được quá trình truyền tin an toàn dựa vào hệ thống

mã hóa khóa công khai. Như các bạn thấy, trong hệ thống mã hóa này thì mỗi một người sử dụng khi tham gia vào đều được cấp 2 khóa : Một khóa dùng cho việc mã hóa dữ liệu (Public key) và một khóa dùng cho việc giải mã dữ liệu (Private key), trong đó Public key được đưa ra cho tất cả mọi người cùng biết, còn Private key phải được giữ kín một cách tuyệt đối. Giả sử hai phía muốn truyền tin cho nhau thì quá trình truyền sử dụng mã hóa khóa công khai được thực hiện như sau :

Sender yêu cầu cung cấp hoặc tự tìm khoá công khai của Receiver trên một Server chịu trách nhiệm quản lý khoá công khai.

Sau đó hai phía thống nhất thuật toán dùng để mã hóa dữ liệu, Sender sử dụng khóa công khai của Receiver cùng với thuật toán đã thống nhất để mã hóa thông tin bí mật.

Thông tin sau khi mã hóa được gửi tới Receiver, lúc này chính Sender cũng không thể nào giải mã được thông tin mà anh ta đã mã hóa (khác với mã hóa khóa riêng).

Khi nhận được thông tin đã mã hóa, Receiver sẽ sử dụng khóa bí mật của mình để giải mã và lấy ra thông tin ban đầu.

Vậy là với sự ra đời của Mã hóa khóa công khai thì khóa được quản lý một cách linh hoạt và hiệu quả hơn. Người sử dụng chỉ cần bảo vệ khóa Private key. Hệ thống này an toàn hơn nhiều so với mã hóa khóa riêng, người mã hóa không thể giải mã được dữ liệu đã mã hóa bằng khóa công khai của người khác. Tuy nhiên nhược điểm của Mã hóa khóa công khai nằm ở tốc độ thực hiện, nó chậm hơn mã hóa khóa riêng cỡ ~1000 lần. Do đó người ta thường kết hợp hai hệ thống mã hóa khóa riêng và công khai lại với nhau và được gọi là Hybrid Cryptosystems (Hệ thống mã hóa lai). Một số thuật toán mã hóa công khai phổ biến :

- RSA : Cái tên RSA là ba chữ cái bắt đầu từ ba cái tên của ba tác giả: Ron Rivest, Adi Shamir, Len Adleman. Sử dụng đồng thời cho mã hóa khóa công khai và chữ ký điện tử. Độ an toàn của thuật toán mã hóa RSA dựa trên việc phân tích một số nguyên tố rất lớn thành hai số nguyên tố.

4.1.2. Thuật toán RSA trong mật mã hóa công khai

Trong mật mã học, RSA là một thuật toán mật mã hóa khóa công khai. Đây là thuật toán đầu tiên phù hợp với việc tạo ra chữ ký điện tử đồng thời với việc mã hóa. Nó đánh dấu một sự tiến bộ vượt bậc của lĩnh vực mật mã học trong việc sử dụng khóa công cộng. RSA đang được sử dụng phổ biến trong thương mại điện tử và được cho là đảm bảo an toàn với điều kiện độ dài khóa đủ lớn.

Thuật toán RSA có hai khóa: khóa công khai (hay khóa công cộng) và khóa bí mật (hay khóa cá nhân). Mỗi khóa là những số cố định sử dụng trong quá trình mã hóa và giải mã. Khóa công khai được công bố rộng rãi cho mọi người và được dùng để mã hóa. Những thông tin được mã hóa bằng khóa công khai chỉ có thể được giải

mã bằng khóa bí mật tương ứng. Nói cách khác, mọi người đều có thể mã hóa nhưng chỉ có người biết khóa cá nhân (bí mật) mới có thể giải mã được.

4.2. Các thao tác chữ ký số

Cơ chế xác thực thông tin bảo vệ hai bên trao đổi thông điệp khỏi bên thứ ba. Tuy nhiên cơ chế này không bảo vệ được một bên khi bên kia cố ý vi phạm, ví dụ như:

- A có thể làm giả một thông điệp và tuyên bố rằng thông điệp này do B gửi. A chỉ cần tạo một thông điệp và thêm vào mã xác thực sử dụng khóa mà A và B chia sẻ.
- B có thể phủ nhận đã gửi thông điệp. Bởi vì A có thể giả mạo thông điệp nên không có cách nào chứng minh sự thật là B đã gửi.

Trong trường hợp không có sự tin tưởng hoàn toàn giữa người gửi và người nhận, cần có một cơ chế tốt hơn xác thực. Giải pháp tốt nhất cho vấn đề này là chữ ký số.

Dựa trên nền tảng các tính chất trên, chúng ta có thể hình thành các yêu cầu cho chữ ký số như sau:

- Chữ ký số là một số các bit phụ thuộc vào thông điệp được ký được đính kèm với thông điệp.
- Chữ ký số phải sử dụng những thông tin độc lập với người gửi để ngăn chặn giả mạo.
- Phải tương đối dễ dàng để tạo ra chữ ký số
- Phải tương đối dễ dàng để nhận ra và xác minh chữ ký số.
- Không thể giả mạo được một chữ ký số, tức là tạo một thông điệp mới cho một chữ ký đã tồn tại hoặc tạo một chữ ký giả cho một thông điệp cho trước.
- Có thể sao chép được chữ ký số để lưu trữ.

4.2.1. Quy trình ký file bằng chữ ký số

Quy trình ký dữ liệu bằng chữ ký số diễn ra như sau:

- Bước 1: Băm dữ liệu
 - Tin nhắn (dữ liệu) đi qua một hàm mật mã để tạo ra một mảng băm của dữ liệu.
 - SHA1 tạo ra mảng băm 160 bit (20 byte).
 - SHA224, SHA256, SHA384, SHA512, MD4, MD5 là một vài thuật toán băm thông điệp khác có sẵn trong openssl.
- Bước 2: Thêm đệm cho giá trị băm
 - Giá trị băm (20 byte trong trường hợp SHA1) được mở rộng đến kích thước RSA key bằng cách thêm đệm tiền tố vào trước mã băm.
 - Sơ đồ đệm mặc định trong openssl là PKCS1.
- Bước 3: Lấy modulus n và số mũ riêng từ khóa bí mật

- Dữ liệu modulus n và số mũ riêng được lấy từ thiết bị đã được lưu vào thiết bị trước đó. Việc lấy dữ liệu này được lấy theo
- Bước 4: Ký mã băm với modulus n và số mũ riêng

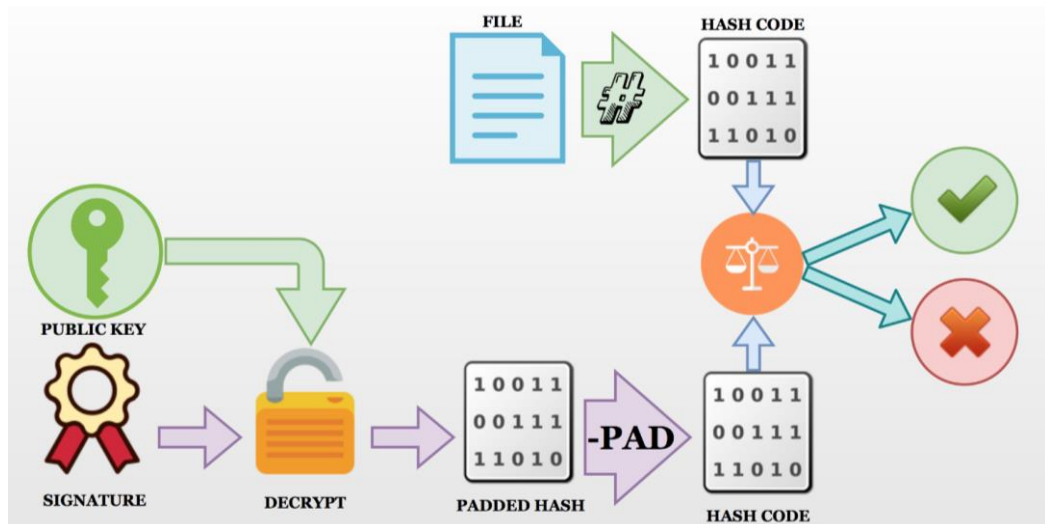
$$s = m^d \pmod{n}$$

Diagram illustrating the signing process equation: $s = m^d \pmod{n}$. The components are labeled as follows:

- Private exponent** (d) points to the exponent d in the equation.
- Message to be signed** (m) points to the message m in the equation.
- Modulus** (n) points to the modulus n in the equation.
- Signature** (s) points to the result s in the equation.

Hình 19: Mã hóa dữ liệu với Modulus n và số mũ riêng [10]

- Mã hóa tin nhắn với mô đun và số mũ riêng để có chữ ký



Hình 20: Quy trình tạo chữ ký số [10]

4.2.2. Quy trình xác thực chữ ký số

Quy trình xác thực chữ ký số được diễn ra như sau:

- Bước 1: Lấy thông tin modulus n và số mũ công khai từ khóa công khai
 - Các nội dung của khóa công khai: Public key chứa Modulus n, số mũ công khai và kích thước khóa.
 - Khóa công khai được cung cấp từ server lưu trữ database tương ứng với chủ thể được cung cấp.

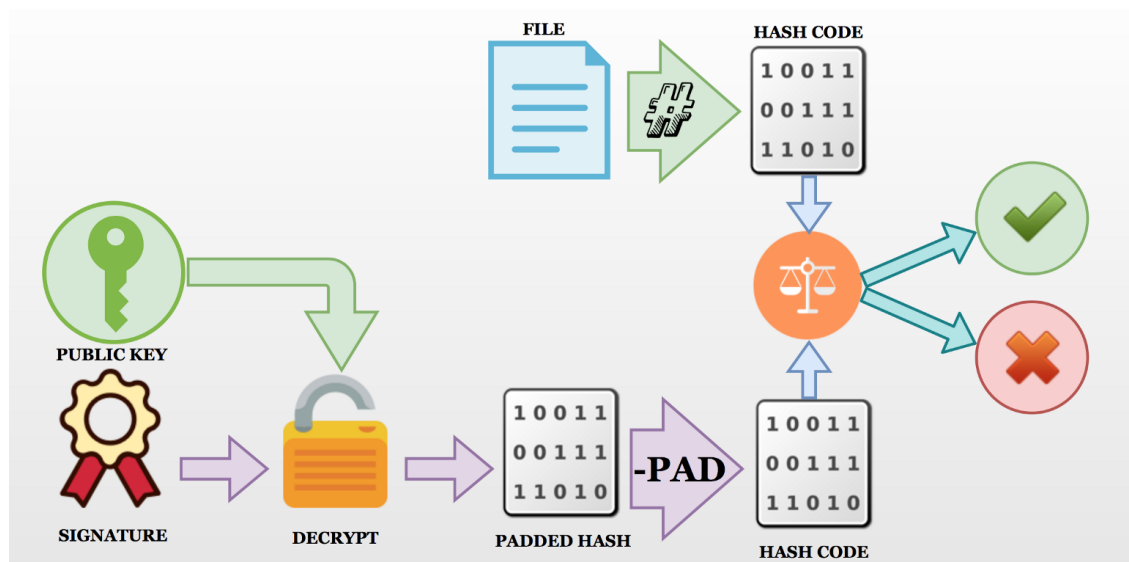
- Bước 2: Chuyển đổi chữ ký sang dạng mã băm đã được chèn thêm padding

$$m = s^e \pmod n$$

Public exponent → e
Message → m Signature → s Modulus → n

Hình 21: Giải mã dữ liệu với Modulus n và số mũ công khai [10]

- Bước 3: Xóa phần padding để có được mã băm của gói tin
- Bước 4: So sánh mã băm vừa được giải mã từ chữ ký số. So sánh mã băm này với mã băm của gói tin để kết luận tính toàn vẹn của dữ liệu.



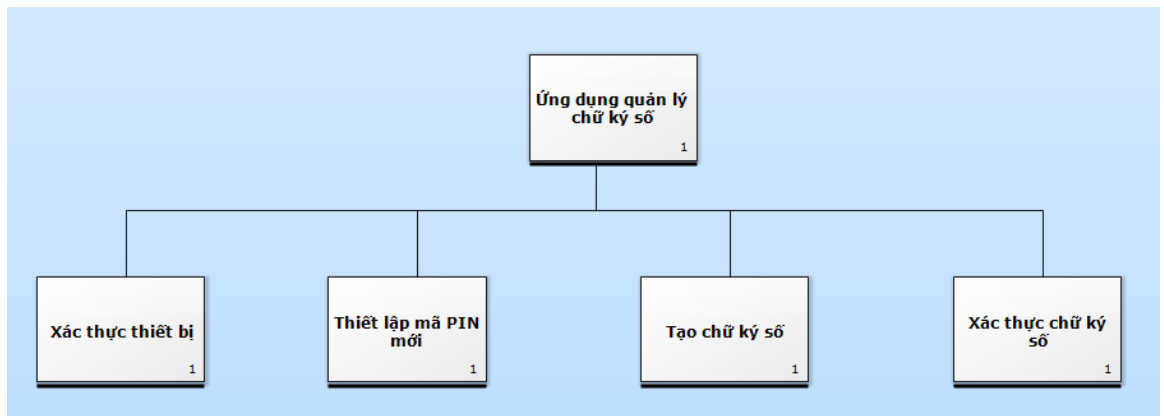
Hình 22: Quy trình xác minh chữ ký số [10]

4.3. Phân tích thiết kế ứng dụng

4.3.1. Biểu đồ chức năng

Các chức năng của ứng dụng bao gồm:

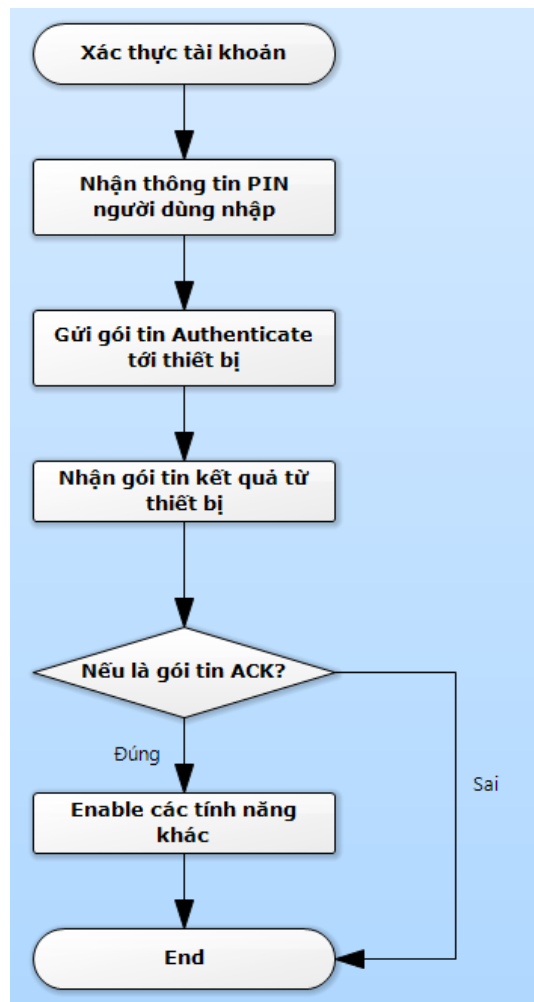
- Xác thực người dùng với thiết bị
- Thiết lập mã PIN mới
- Tạo chữ ký số
- Xác thực chữ ký số



Hình 23: Biểu đồ chức năng ứng dụng chữ ký số

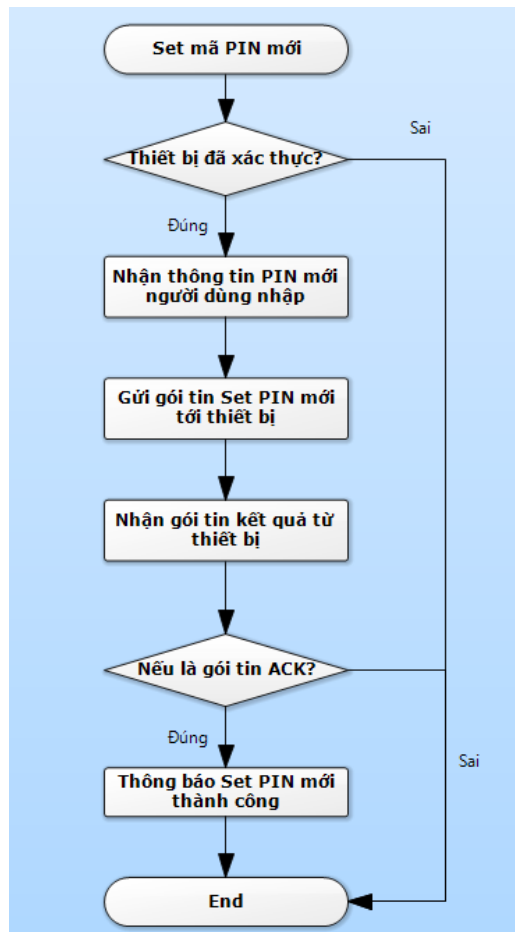
4.3.2. Biểu đồ hoạt động

- Sau đây là biểu đồ hoạt động cho các chức năng được nêu trên:
- Xác thực người dùng với thiết bị



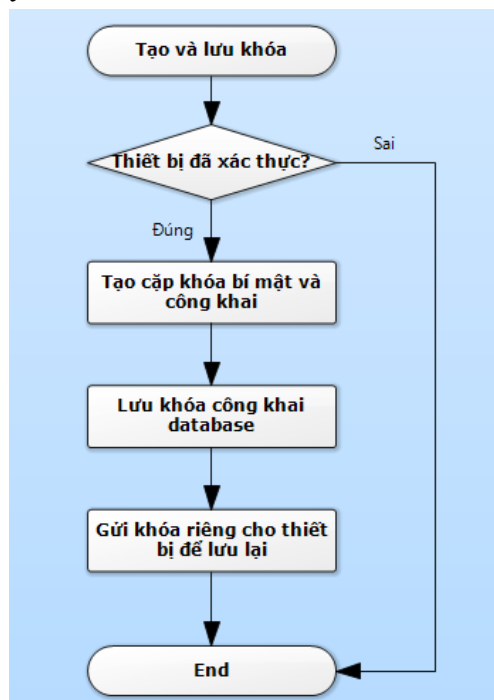
Hình 24: Biểu đồ hoạt động chức năng xác thực người dùng

- Chức năng set mã PIN mới

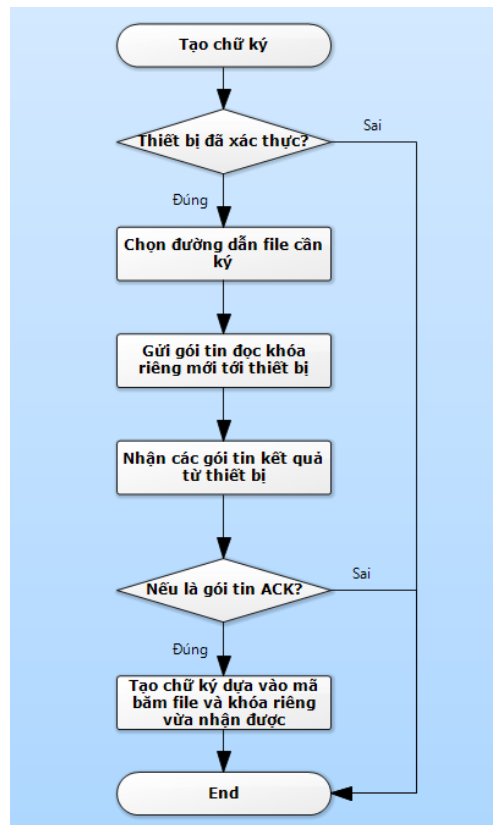


Hình 25: Biểu đồ hoạt động chức năng set mã PIN mới

- Chức năng tạo chữ ký số:

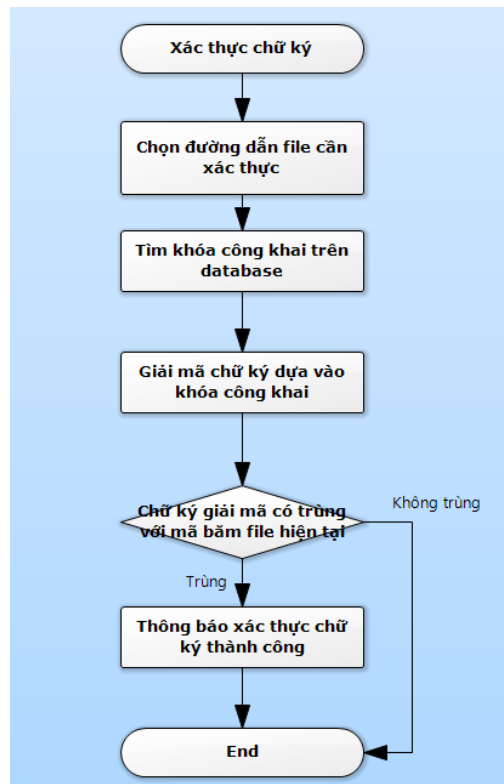


Hình 26: Biểu đồ hoạt động chức năng tạo và lưu khóa



Hình 27: Biểu đồ hoạt động chức năng tạo chữ ký số

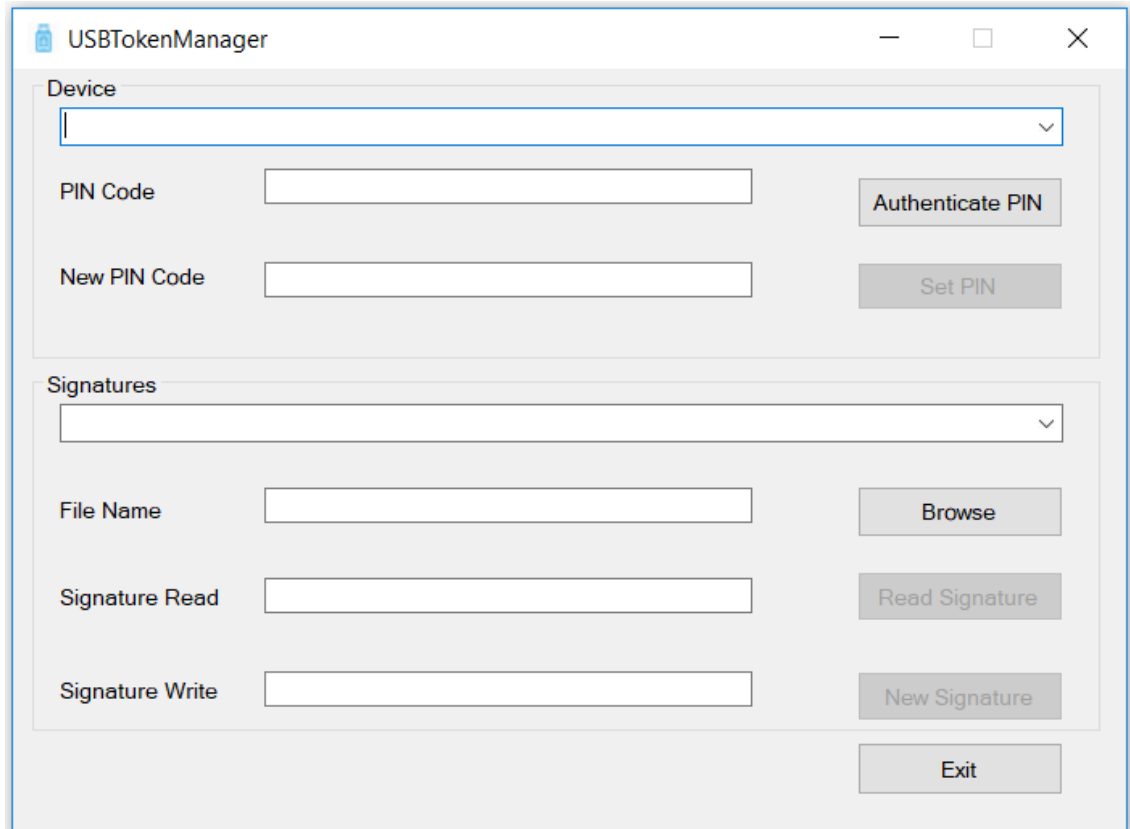
- Chức năng xác thực chữ ký số



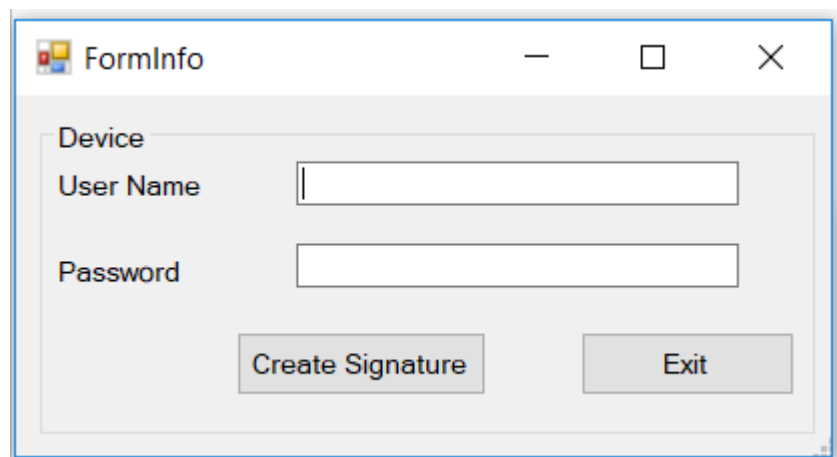
Hình 28: Biểu đồ hoạt động chức năng xác thực chữ ký số

4.4. Giao diện ứng dụng chữ ký số

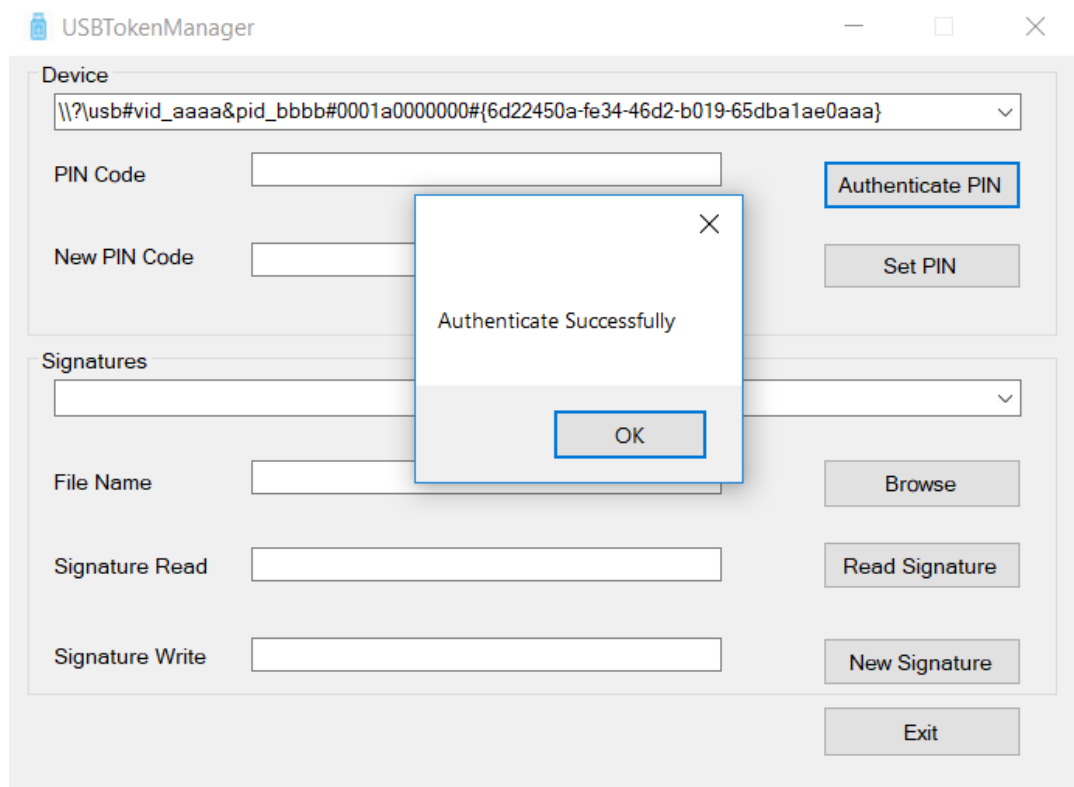
Chúng ta có nhiều các tạo giao diện ứng dụng trên Windows. Một vài giải pháp hay được sử dụng đó là Winform trên .Net Framework, lập trình MFC, lập trình Qt... Giao diện ứng dụng này được tạo sử dụng Winform chạy trên nền .Net Framework. Chúng ta sử dụng Winform thay vì các cách đã nêu Winform khá dễ tiếp cận, giao diện đẹp, đồng thời giúp tiết kiệm thời gian phát triển sản phẩm.

The screenshot shows a Windows application window titled "USBTokenManager". It features a "Device" dropdown menu at the top. Below it are two input fields: "PIN Code" and "New PIN Code", each with a corresponding button ("Authenticate PIN" and "Set PIN" respectively). A "Signatures" dropdown menu is positioned below the PIN fields. Underneath are three more input fields: "File Name", "Signature Read", and "Signature Write", each with a button ("Browse", "Read Signature", and "New Signature" respectively). An "Exit" button is located at the bottom right of the window.

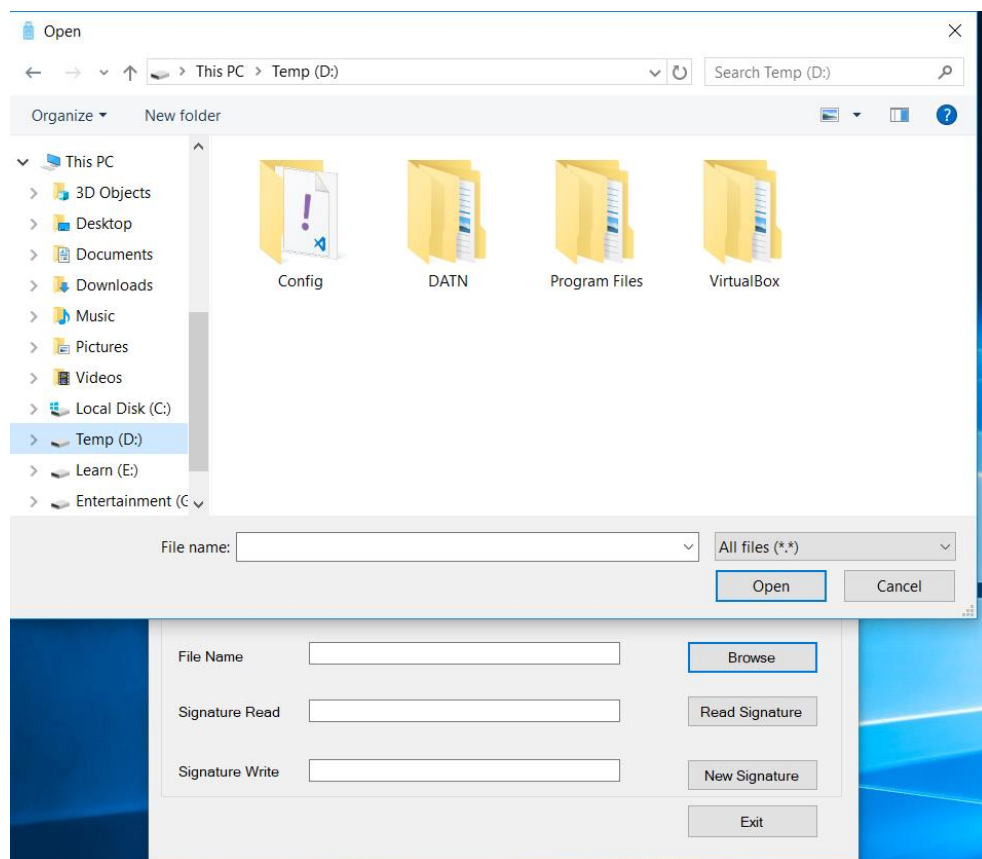
Hình 29: Giao diện chính ứng dụng chữ ký số

The screenshot shows a Windows application window titled "FormInfo". It contains two input fields: "User Name" and "Password". Below these fields are two buttons: "Create Signature" and "Exit".

Hình 30: Giao diện tạo chữ ký số



Hình 31: Message Box thông báo xác thực thành công



Hình 32: Giao diện chọn file để ký hoặc verify chữ ký

CHƯƠNG 5: KẾT LUẬN

Trải qua một quá trình nghiên cứu nghiên cứu giao tiếp thiết bị nhúng theo chuẩn USB và ứng dụng trong dịch vụ chữ ký số, từ mức phác thảo ý tưởng đến sản phẩm hoàn thiện, chúng tôi cũng đã gặp không ít những khó khăn. Từ việc xây dựng kiến trúc đến lựa chọn giải pháp đến các khó khăn khách quan về mặt phần cứng. Việc tìm hiểu ý nghĩa của các khái niệm về thiết bị nhúng, lập trình firmware cho thiết bị nhúng đều là những kiến thức mới đối với em và yêu cầu em học tập. Thêm vào đó, em là một sinh viên xuất thân từ chuyên ngành An toàn thông tin do đó các kiến thức về vi điều khiển, lập trình vi điều khiển của em khá ít ỏi. Việc tìm hiểu một lĩnh vực mới trong thời gian một kỳ tương đối gian nan và yêu cầu nhiều sự nỗ lực từ phía em. May mắn thay, em nhận được sự giúp đỡ nhiệt tình từ TS. Phạm Ngọc Hưng trong suốt quá trình nghiên cứu. Bên cạnh đó, là các thầy và các bạn trong phòng lab của bộ môn kỹ thuật máy tính đã giúp đỡ để em có thể hoàn thành đồ án một cách trọn vẹn.

Đóng góp chính của em trong đồ án này bao gồm việc sửa lại mã nguồn firmware từ example của Keil, để thiết bị có thể giao tiếp gói tin có kích thước lớn thay vì một byte một lần gửi như firmware ban đầu. Tiếp theo là xây dựng kịch bản giao tiếp giữa thành phần firmware thiết bị với trình điều khiển trên PC, và cuối cùng là ứng dụng lưu trữ chữ ký số.

Định hướng sản phẩm trong tương lai sẽ làm được thiết kế mạch và đi sản xuất thiết bị thực tế. Bên cạnh đó, xây dựng thêm các ứng dụng từ kịch bản giao tiếp đã có để có thêm ứng dụng từ nền tảng trên.

Đồ án này được thực hiện trong thời gian ngắn nhưng khối lượng công việc cũng như các vấn đề cần giải quyết là khá nhiều. Vì thế không thể tránh khỏi những thiếu sót còn tồn tại, người viết đồ án rất mong nhận được sự đóng góp ý kiến từ phía các thầy cô và bạn bè để có thể phát triển và hoàn thiện sản phẩm hơn nữa trong tương lai.

TÀI LIỆU THAM KHẢO

- [1] "MCB2300 Evaluation Board," 2019. [Online]. Available: <http://www.keil.com/mcb2300/>.
- [2] "WinUSB (Winusb.sys)," 4 2017. [Online]. Available: <https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/winusb>.
- [3] "Microsoft Visual C++ Fundamentals," 2019. [Online]. Available: <http://www.functionx.com/vccli/general/introprogramming.htm>.
- [4] "USB in a NutShell," 2019. [Online]. Available: <https://www.beyondlogic.org/usbnutshell/usb1.shtml>.
- [5] "Technology will simplify employee's ID verification," [Online]. Available: <https://www.linkedin.com/pulse/technology-simplify-employees-id-verification-g%C3%A9rard-le-comte/>.
- [6] "An Introduction to USB Protocol," 2019. [Online]. Available: <https://javancook.com/2015/an-introduction-to-usb-protocol/>.
- [7] "WinUSB Architecture and Modules," 2017. [Online]. Available: <https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/winusb-architecture>.
- [8] Microsoft, "How to Use WinUSB to Communicate with a USB Device," 2019.
- [9] "Những khái niệm cơ bản trong Mật mã học," [Online]. Available: <https://viblo.asia/p/nhung-khai-niem-co-ban-trong-mat-ma-hoc-RnB5p7MrlPG>.
- [10] "RSA sign and verify using Openssl : Behind the scene," 2019. [Online]. Available: <https://medium.com/@bn121rajesh/rsa-sign-and-verify-using-openssl-behind-the-scene-bf3cac0aade2>.