FIDO2-CTAP2.1 PoC on STM32F4-Discovery

鉿安智慧

Intern： 陳可瀚

Table of Contents

Abstract

FIDO2標準旨在實現無密碼登入與驗證，不論在web端或是app端都能提供一個更方便且更安全的方法通過身份驗證。FIDO2標準包含兩個部分，分別是WebAuthn和CTAP，而我負責的是CTAP的部分，CTAP全名Client To Authenticator Protocol，顧名思義這個protocol規範了Authenticator和platform的互動方式、可用的API和傳輸資料的形式等等。而此次的PoC我主要是用Yubico公司發售的Yubikey5當作範本，然後使用STM32F4 Discovery板子搭配STM32 Cube IDE用USB HID的形式實現CTAP2.1的功能。其中我使用STM板上的user button當作 ”User Presence”、使用輸入PIN碼的方式當作 “User Verification”。

CTAP2.1 API

一、簡述：

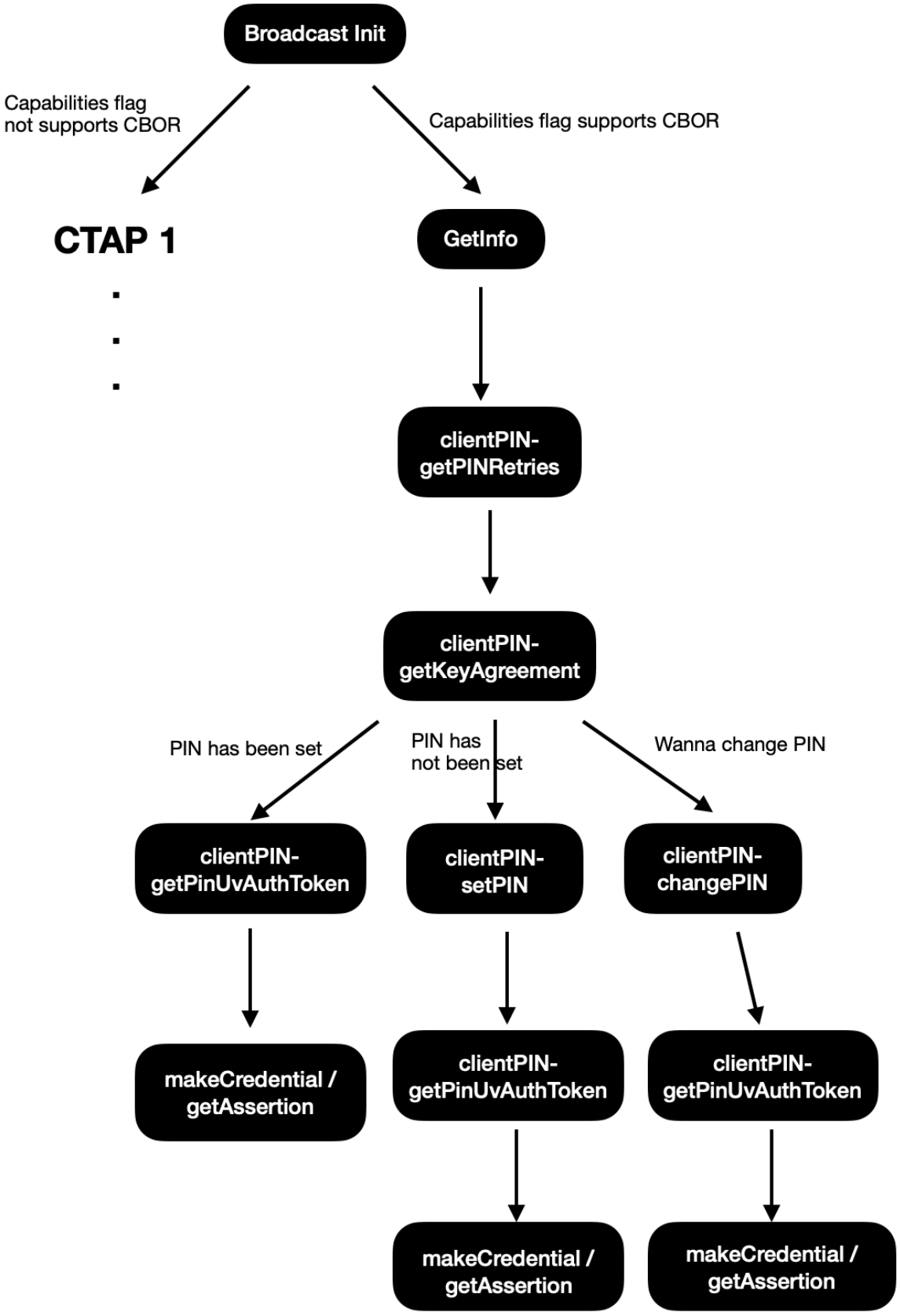
CTAP2.1 我們需要完成的API總共有九個：

- authenticatorMakeCredential (0x01)
- authenticatorGetAssertion (0x02)
- authenticatorGetNextAssertion (0x08)
- authenticatorGetInfo (0x04)
- authenticatorClientPIN (0x06)
- authenticatorReset (0x07)
- authenticatorCredentialManagement (0x0A)
- authenticatorSelection (0x0B)
- authenticatorConfig (0x0C)

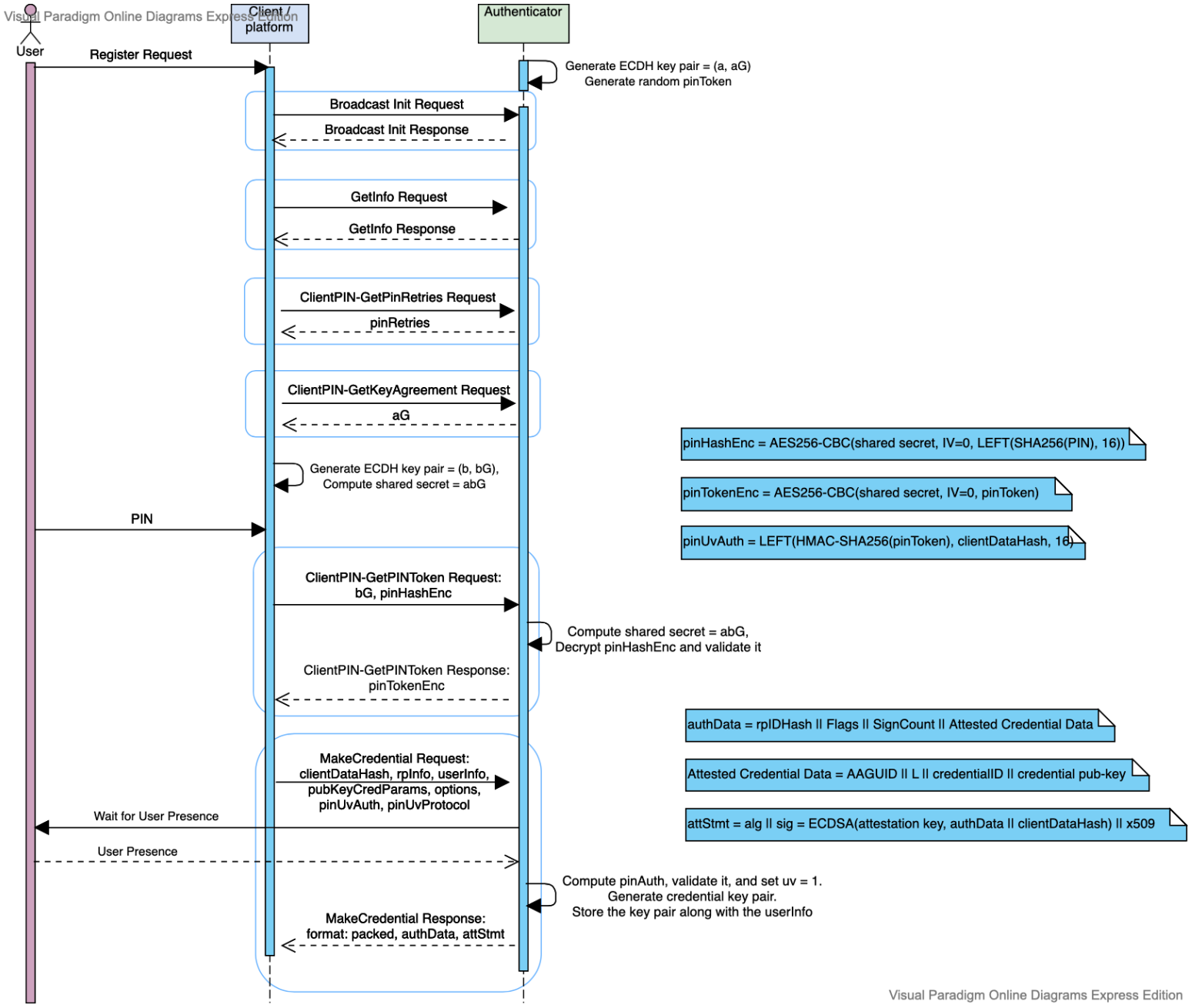
主要分成幾個類別：
第一類是Register用的(0x01)
第二類是Authenticate(Login)用的(0x02)、(0x08)
第三類是取得詳細資訊的(0x04)
第四類是處理PIN碼用的(0x06)系列APIs
第五類是設定管理類APIs (0x07)、(0x0A)、(0x0B)、(0x0C)

目前大致上完成的API為(0x01)、(0x02)、(0x08)、(0x04)、(0x06)，也就是除了設定管理類APIs其餘都完成PoC(某些API還是存在一點bug，後面會說明)。

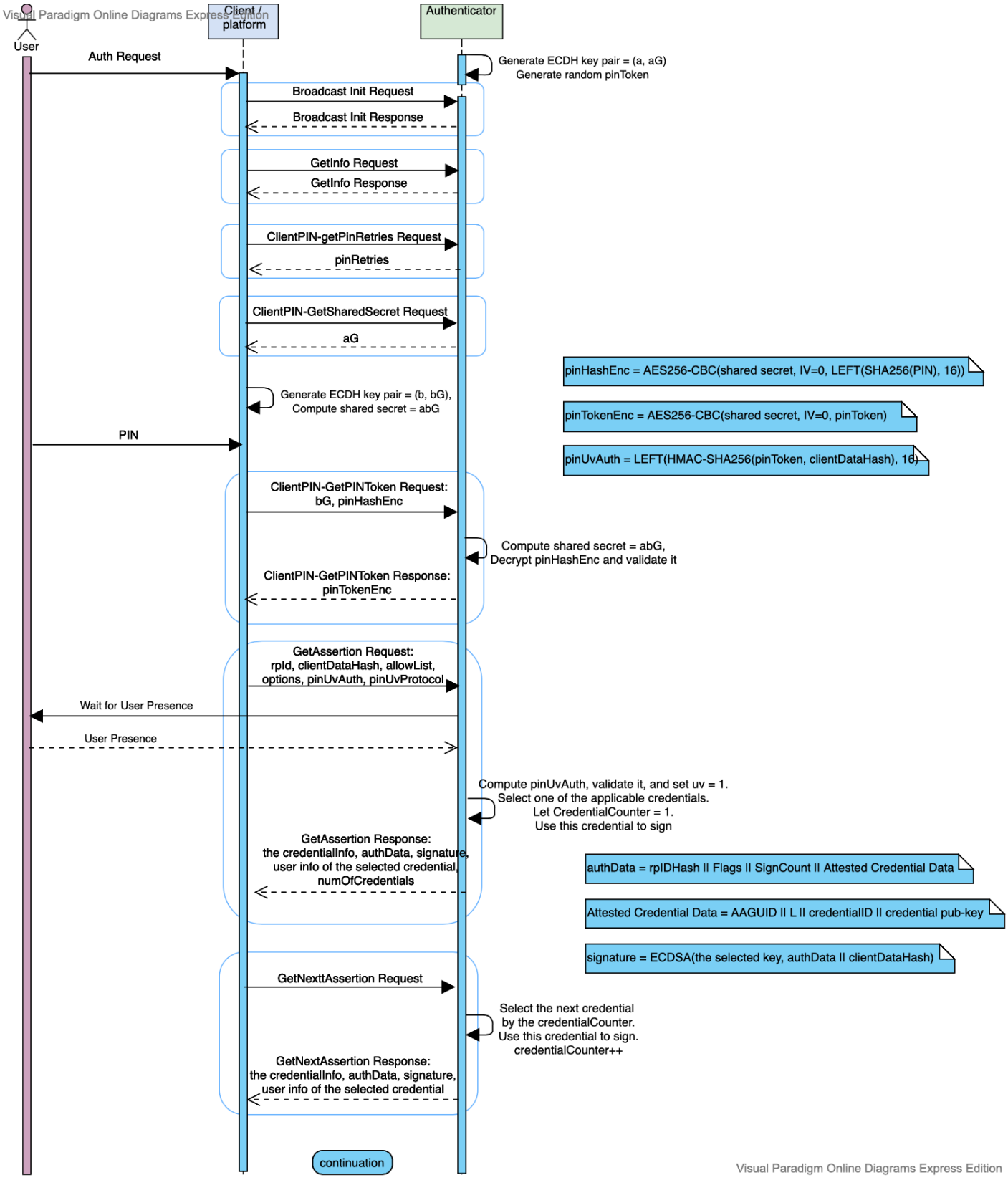
二、 Register and Login process decision tree (用BUS HOUND觀察Yubikey的結果)：



三、Register Sequence Diagram (Suppose PIN has been set)：



四、Authenticate(Login) Sequence Diagram (Suppose PIN has been set)：



五、APIs說明：

(0x01)、authenticatorMakeCredential：

1. 在接收Request的部分處理六個參數分別是：clientDataHash(0x01)、rp(0x02)、user(0x03)、pubKeyCredParams(0x04)、pinUvAuthParam(0x08)、pinUvAuthProtocol(0x09)。
(0x01)：clientDataHash是一個32bytes的byte-array。
(0x02)：rp是由string形式的rpID和string形式的rp name所組成。
(0x03)：user包含byte-array形式的userID，string形式的user name，和string形式的display name。
(0x04)：pubKeyCredParams包含數組可用的密碼演算法，每組演算法包含IANA_COSE認定的演算法代號和type為public-key的字串。
(0x08)：pinUvAuthParam是HMAC-SHA- 256(pinUvAuthToken, clientDataHash)過後的結果，此為16bytes的byte-array。
(0x09)：pinUvAuthProtocol設定為整數1。

2. 處理data並回傳Response：
首先等待user按下按鈕，接著檢查HMAC-SHA- 256 (pinUvAuthToken, clientDataHash)是否和Request傳過來的相同，若相同則使用pubKeyCredParams裡所提供的演算法(通常會是ES256)產生一對credential-key-pair，接著使用attestation-private-key做ECDSA產生簽章，而要被簽的內容為(32bytes的rpID hash ‖ 32bytes的clientDataHash ‖ credentialID ‖ credential-public-key)，然後就可回傳Response。
*若上述步驟發生錯誤，則會回傳相應的錯誤代碼。
而Response的結構是：
(0x01)：fmt是字串”packed”。
(0x02)：authData是(32bytes的rpID hash ‖ 1byte的flags ‖ 4bytes的signCounter ‖ 16bytes的aaguID ‖ credentialID length ‖ credentialID ‖ credential-public-key)的byte-array。
(0x03)：attStmt包含70~72bytes的signature和x509憑證。

3. 尚未解決的問題：
- 因為這裡的x509憑證是self-signed憑證，雖然可以通過Demo網站上的測試(它沒有檢查憑證)，但是無法通過自架網站上fido server的檢驗，因此這部分還要思考怎麼解決。

(0x02)、authenticatorGetAssertion：

1. 接收Request的部分處理六個參數分別是：
rpId (0x01)、clientDataHash (0x02)、allowList (0x03)、options (0x05)、pinUvAuthParam (0x06)、pinUvAuthProtocol (0x07)。

(0x01)：rpId是一個字串。
(0x02)：clientDataHash是32bytes的byte-array。
(0x03)：allowList可能包含數個credential ID，每個credential ID是一個byte-array。
(0x05)：options是一個map，{“up”: true, “uv”: false}。
(0x06)：pinUvAuthParam是HMAC-SHA- 256(pinUvAuthToken, clientDataHash)過後的結果，此為16bytes的byte-array。
(0x09)：pinUvAuthProtocol設定為整數1。

2. 處理data並回傳Response：
等待user按下按鈕，接著檢查HMAC-SHA- 256 (pinUvAuthToken, clientDataHash)是否和Request傳過來的相同，若相同則判斷allow list給的credential ID是否大於一，若大於一個，則選擇一個credentail ID，若只有一個credential ID則選擇這個credential ID，用選擇好的credential ID找到相對應的key做ECDSA並回傳Response。

Response的結構：
(0x01)：credential也就是由byte-array組成的credential ID。
(0x02)：authData是(32bytes的rpID hash ‖ 1byte的flags ‖ 4bytes的signCounter)的byte-array。
(0x03)：signature是70~72bytes的byte-array，是用此把credential ID找到credential-private-key做ECDSA產生的簽章，其被簽的內容是
(rpID hash ‖ 1byte user presence ‖ 4bytes signCounter ‖ clientDataHash)
(0x05)：numberOfCredentials，只有當allow list裡的credential ID大於一個才會出現，此為credential ID的個數(integer)。

3. 尚未解決的問題：
- 在使用Demo website測試時發現一開始傳來的Request中並沒有pinUvAuthParam和pinUvAuthProtocol這兩個參數。

- 一開始傳來的Request中的options中的up參數它設為false。

- 以上兩點的問題，雖然最後還是能夠成功Login (rp端會再次getInfo然後再次呼叫getAssertion，此時就有pinUvAuthParam和pinUvAuthProtocol)，但是這個行為怪怪的，目前判斷有可能是某些管理類API尚未完成所造成的(像是0x0a的feature detection)。

(0x08)、authenticatorGetNextAssertion：

1. Request沒有參數

2. 處理data並回傳Response：
在allow list中挑選一個尚未被選到的credential ID，用這個選擇好的credential ID找到相對應的key做ECDSA並回傳Response。而Response的結構和authenticatorGetAssertion一模一樣，差別在於不用回傳numberOfCredentials參數。

3. 尚未解決的問題：
- 因為我還不會在STM32F4的板子上永久性儲存資料(斷電後資料還在)，因此當rp傳來的allow list裡的credential ID是我重新啟動前註冊過的資料，我就無法找到對應的key做Response。

- 另外因為Demo site的設計，當我們輸入user name後登入時，通常只會有一組credential，因此我就無法測試authenticatorGetNextAssertion這個API。

(0x04)、authenticatorGetInfo：

1. Request沒有參數

2. 回傳Response(共有10個參數)：
(0x01)：versions是一個array，第一個元素是”FIDO_2_0"字串，第二個元素是"FIDO_2_1_PRE"字串。
(0x02)：extensions是一個array，這邊只包含了一個”credProtect"字串。
(0x03)：aaguid是一個16bytes的byte-array。
(0x04)：options是一個map，包含四個鍵值對，第一個”plat”字串設為false，第二個”rk”字串設為false，第三個”clientPin”的真假值取決於PIN是否已經被設定，第四個”up”設為true。 *(不要帶”uv”參數)
(0x05)：maxMsgSize設為integer 1200。
(0x06)：pinUvAuthProtocols是一個array，只有一個integer 1。
(0x07)：maxCredentialCountInList是integer 8。
(0x08)：maxCredentialIdLength是integer 128。
(0x09)：transports是一個array，只有一個字串”usb”。
(0x0a)：algorithms是一個array，第一個是COSE演算法代號-7和字串”public-key”，第二個是COSE的演算法代號-9和字串”public-key”。

3. 尚未解決的問題：
- aaguid是一個代表產品的代號，所以我先隨便寫一個16bytes的byte-array，可能之後要更改過來。

(0x06)、AuthenticatorClientPIN：
此API有五個subCommand：getPINRetries(0x01)、getKeyAgreement(0x02)、setPIN(0x03)、changePIN(0x04)、getPinUvAuthTokenUsingPin(0x05)。

1. getPINRetries (0x01)：

(1) Request的參數：
(0x01) pinUvAuthProtocol是integer 1。
(0x02) subCommand是integer 3。

(2) Response回傳pinRetries。

2. getKeyAgreement (0x02)：

(1)　Request的參數：
(0x01) pinUvAuthProtocol是integer 1。
(0x02) subCommand是integer 2。

(2) 產一亂數a，在p256的橢圓曲線上計算a倍的G，並回傳結果。

Response結構：
(0x01)：KeyAgreement是由一個map構成的：
{
keyType: EC2,
alg: ES256,
curve: p256,
x-coordinate: 32-byte byte array
y-coordinate: 32-byte byte array
}

3. SetPIN (0x03)：

(1). Request的五個參數：
(0x01) pinUvAuthProtocol是integer 1。
(0x02) subCommand是integer 3。
(0x03) keyAgreement是platform的ECDH公鑰bG，是一個map，其結構與前述的aG相同。
(0x04) pinUvAuthParam是LEFT(HMAC-SHA-256(sharedSecret, newPinEnc), 16)，其資料型別是16bytes的byte-array。
(0x05) newPinEnc是AES256-CBC(sharedSecret, IV=0, newPin)，newPIN的range最大只能有63bytes最小4bytes，經過padding後的newPIN會是64bytes，所以encrypt後的密文會是32bytes。

(2) 處理data回傳Response：
首先在橢圓曲線上計算sharedSecret abG，然後計算LEFT(HMAC-SHA-256(sharedSecret, newPinEnc), 16)看是不是和Request傳來的一模一樣，如果相同使用AES256-CBC解密newPinEnc得到newPIN，去除尾巴的padding後把LEFT(SHA-256(newPin), 16)儲存在裝置上，如果以上都沒問題則回傳CTAP2_OK (0x0)，否則回傳相對應的錯誤代碼。

4. changePIN (0x04)：

(1). Request的六個參數：
(0x01) pinUvAuthProtocol是integer 1。
(0x02) subCommand是integer 4。
(0x03) keyAgreement是platform的ECDH公鑰bG，是一個map，其結構與前述的都相同。
(0x04) pinUvAuthParam是LEFT(HMAC-SHA-256(sharedSecret, newPinEnc ‖ pinHashEnc), 16)，其資料型別是16bytes的byte-array。
(0x05) newPinEnc是AES256-CBC(sharedSecret, IV=0, newPin)，newPIN的range最大只能有63bytes最小4bytes，經過padding後的newPIN會是64bytes，所以encrypt後的密文會是32bytes。
(0x06) pinHashEnc是AES256-CBC(sharedSecret, IV=0, LEFT(SHA-256(curPin),16))的32bytes byte-array。

(2) 處理data回傳Response：
如果pinRetries不等於0，計算sharedSecret abG，然後計算LEFT(HMAC-SHA-256(sharedSecret, newPinEnc ‖ pinHashEnc), 16)看是不是和Request傳來的一模一樣，如果一樣把pinRetreis減一，然後解密pinHashEnc，把解密後的結果和裝置內儲存的SHA-256(Pin)比對，若正確把pinRetreis設至最大值後解密newPinEnc，把得到的newPin去掉尾巴的padding後在裝置上儲存LEFT(SHA-256(newPin), 16)，如果以上都沒問題則回傳CTAP2_OK (0x0)，否則回傳相對應的錯誤代碼。

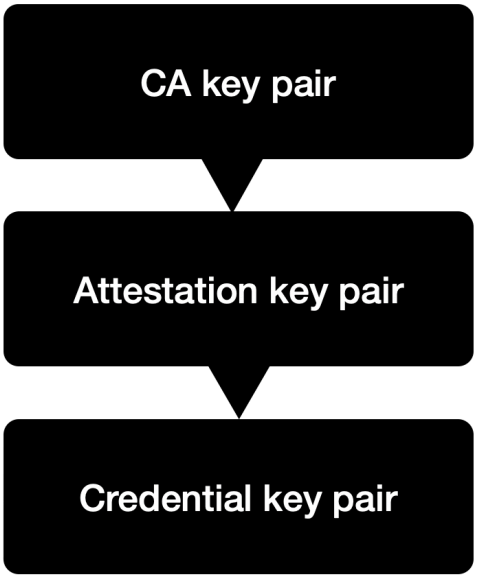
5. getPinUvAuthTokenUsingPin (0x05)：

(1)　Request的四個參數：
(0x01) pinUvAuthProtocol是integer 1。
(0x02) subCommand是integer 5。
(0x03) keyAgreement是platform的ECDH公鑰bG，是一個map，其結構與前述的都相同。
(0x06) pinHashEnc是AES256-CBC(sharedSecret, IV=0, LEFT(SHA-256(curPin),16))的32bytes byte-array。

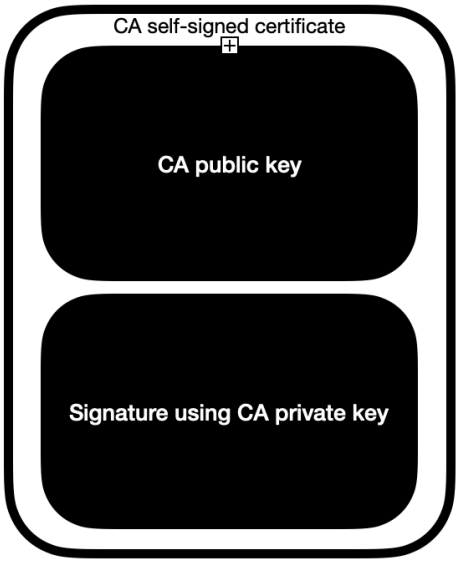
(2) 處理data回傳Response：
如果pinRetries不等於0，計算sharedSecret abG，pinRetries減一，然後解密pinHashEnc得到SHA-256(curPin)，比對和裝置內儲存的SHA-256(curPin)，如果相同把pinRetreis設至最大值然後回傳AES256-CBC(sharedSecret, IV=0, pinUvAuthToken)，如果不同重新在橢圓曲線上產生aG。

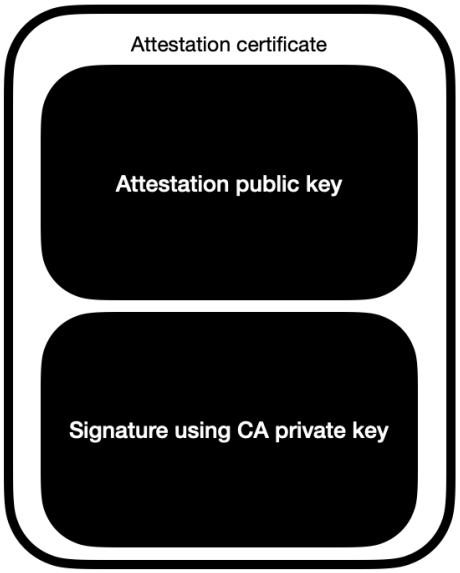
六、Attestation certificate

1. Key relation：



2. Certificates form：





3. 使用openssl 自產 X.509 self-signed certificate：

步驟：

(1) Create CA private key

(2) Generate CA self-signed certificate

(3) Create attestation private key

(4) Generate attestation certificate signing request (CSR)

(5) Sign the CSR with CA’s key and generate the certificate (attestation)

在openssl上相對應的command：

(1) openssl genpkey -algorithm RSA -out ca-priv-key.pem -pkeyopt rsa_keygen_bits:2048

(2) openssl req -key ca-priv-key.pem -new -x509 -days 365 -out ca-cert.pem -sha256 -subj /CN=CACert

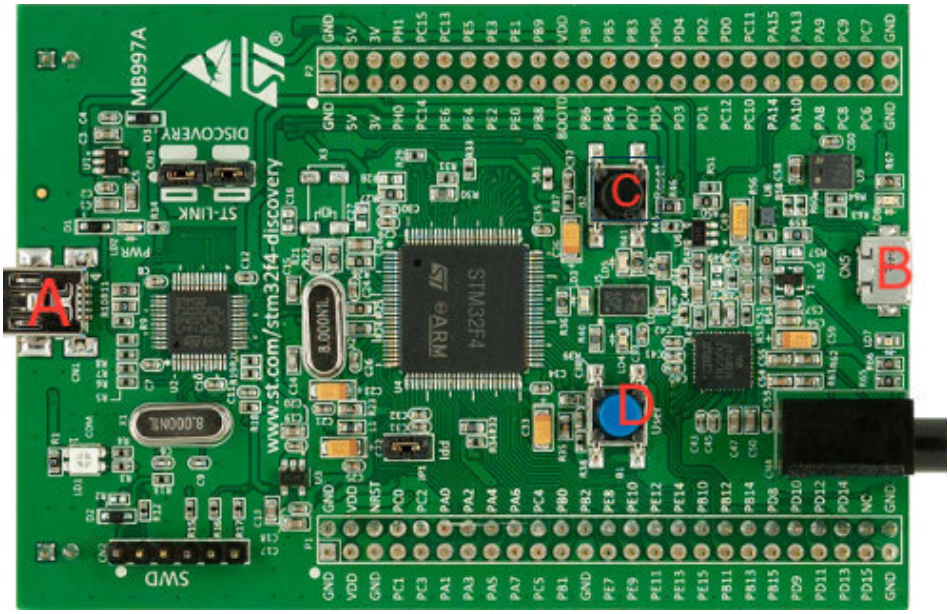
(3) openssl ecparam -name prime256v1 -genkey -noout -out att-priv-key.pem

(4) openssl req -new -sha256 -key att-priv-key.pem -out att-csr.csr

(5) openssl x509 -req -in att-csr.csr -out cert.pem -CA ca-cert.pem -CAkey ca-priv-key.pem -CAcreateserial -days 365 -sha256

STM32F4-Discovery板與STM32 cube IDE

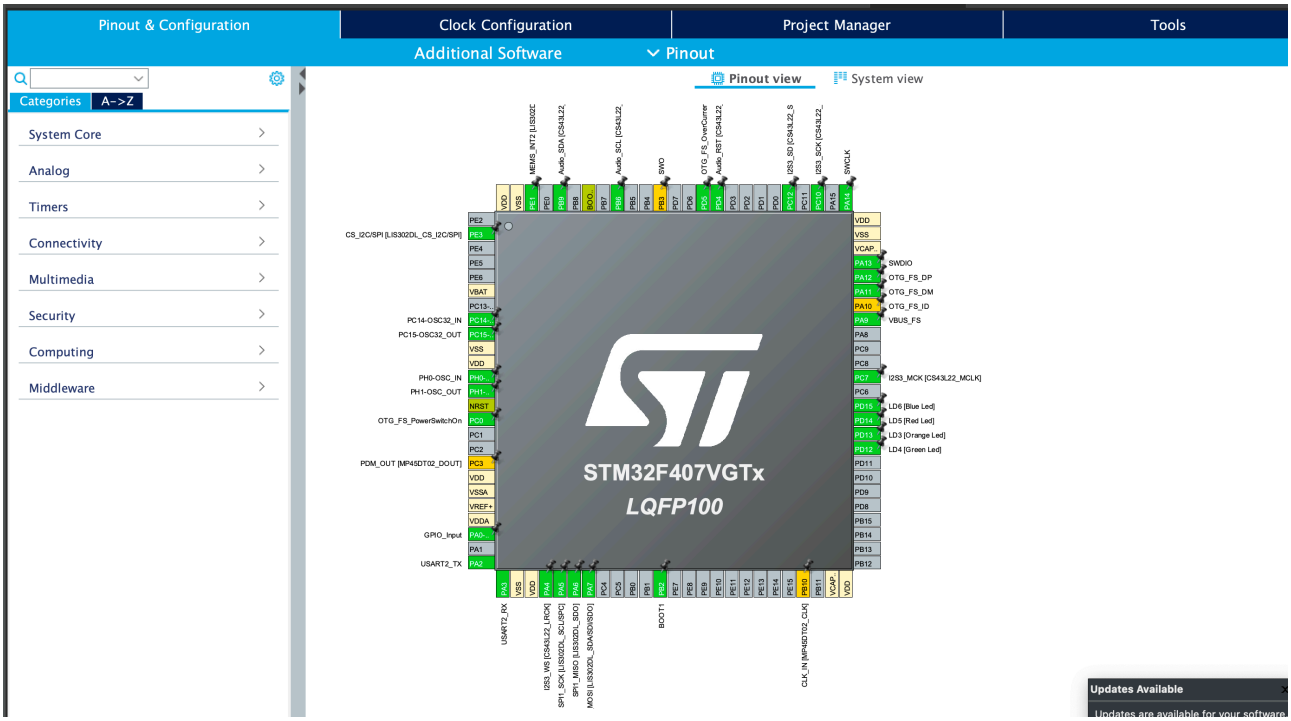
一、STM32F4-Discovery板



- A端是debugger端，也是供電端，寫好的code從此處灌入。
- B端是連接USB接上電腦後和電腦傳輸資料的端口。
- C按鈕是重新啟動按鈕，按下此按鈕code會重新run。
- D按鈕我把它設定成user button，用來作user presence用的。

二、STM32 cube IDE

下載連結：https://www.st.com/en/development-tools/stm32cubeide.html
步驟：安裝完IDE後，new一個STM32 Project，然後選擇STM32F4 Discovery板子，選完之後需要設定STM32 晶片的一些東西。



- 把Connectivity裡的USART2的Mode改成Asynchronous。
- 把Connectivity裡的USB_OTG_FS的Mode改成Device_Only。
- 把Security裡的RNG Activate。
- 把Middleware裡的USB_DEVICE的Mode選成Custom Human Interface Device，然後把USB_DEVICE的Configuration的REPORT_DESC_SIZE設成34，OUTREPORT_BUF_SIZE設成64。
- 這樣就完成ioc檔的設定，點擊最上面齒輪符號後IDE就會自動產生code。

Source Code

一、描述：

Source code連結：

(1) 我寫的code大部分都是放在 /Core/Src/main.c 檔裡。

(2) usb-hid 接收資料的邏輯則放在 /USB_DEVICE/APP/usbd_custom_hid_if.c 檔中的static int8_t CUSTOM_HID_OutEvent_FS(uint8_t event_idx, uint8_t state) 函數中。

(3) 我總共找了三個網路上的library放在程式裡，第一個是 /Core/Src/AES_CBC_lib/ 用來操作AES_CBC的加解密，第二個是 /Core/Src/hmac-sha256/ 用來做hmac，最後一個是 /Core/Src/libecc/，這裡面有豐富的橢圓曲線演算法、雜湊函數和finite-field運算，其中我使用的橢圓曲線是SECP256R1，雜湊函數是SHA-256。

(4) 有關Attestation certificate的資訊我是放在 /Core/Inc/attestation.h 檔案中。其中 att_priv_key陣列存放的是我在外部自產的一把SECP256R1 32-bytes的key，而att陣列存放的是我在外部自產的X.509 attestation certificate，後面會說明我是如何產出憑證的。

二、自定義的全域變數 (main.c 檔)：

大部分的變數都可以用變數名稱來判斷是什麼，因此這裡我只列出需要注意的部分。

(1) extern u8 totalOutDataPacket; 和 extern u8 nowOutDataPacket; 這兩個變數是在main.c檔和usbd_custom_hid_if.c檔中共同使用的，這兩個變數是在記錄Request傳來的原始資料總共有幾包和目前接收到第幾包，因為USB-HID的report descriptor中描述一包packet的最大容量是64 bytes，所以收到Request可能會有很多包傳過來。

(2) extern u8 receivedBuf[1024]; 和 u8 receivedData[1024]; receivedBuf這個array裝的是整個Request的原始資料，而receivedData這個array裝的是整理過後的Request資料，也就是Request的payload data，後面有一個自訂function是將receivedBuf轉換成receivedData。

(3) u8 myCID[4]; 這個array定義了USB-HID通訊時使用的channel ID，這個array可以自定義內容 (只要不是 {0xff, 0xff, 0xff, 0xff}就好)。

(4) u16 payloadOffset; 和 u8 payloadData[1024]; payloadData這個array是用來裝Response的payload data，而payloadOffset是用來記錄目前payloadData這個array用到第幾個byte。

(5) u8 sendBuf[1024]; 這個array是最後要傳送Response時的資料，通常我會使用我下面自定義的function：void sendCBORResponse(void); 來把payloadData包裝成sendBuf並傳送Response。而如果當payloadData沒有很長或者只是要傳送錯誤代碼時，我就會直接把資料寫入sendBuf並傳送。

(6) u8 encodedSig[75]; 和u8 encodedSigLen; 其中encodedSig這個array裝的是經過ANS.1編碼過的簽章，而encodedSigLen是記錄編碼過後的簽章長度，其長度介於70~72bytes。下面有個自訂function: encodeSignature(void); 是將my_sig_r 和 my_sig_s這兩array所代表的簽章經過ANS.1編碼放在encodedSig中。

(7) uint32_t signCount; 記錄credentials authenticate(Login)過的次數。因此，這裏必須修改為記錄每個credential的signCount，也就是每個credential有自己的signCount。

三、需要保存在authenticator中的資料 (斷電後不能不見的變數)：

(1) u8 isSetPIN;

(2) u8 pinRetries;

(3) u8 consecutivePinErr;

(4) u8 pinHash[16];

(5) 已經註冊過的 credential IDs 以及 相對應的credential key pairs。這裏建議以key-value對的資料結構存取，以方便之後查找key。

(6) 每個credential的signCount。

四、自訂函數：

(1) 與API相關：

void broadcastInit(void); 對應到CTAPHID 的 init command

void getInfo(void); 對應到authenticatorGetInfo

void getPINRetries(void); 對應到AuthenticatorClientPIN的getPINRetries

void getKeyAgreement(void); 對應到AuthenticatorClientPIN的getKeyAgreement

void setPIN(void); 對應到AuthenticatorClientPIN的setPIN

void changePIN(void); 對應到AuthenticatorClientPIN的changePIN

void getPinUvAuthToken(void); 對應到AuthenticatorClientPIN的getPinUvAuthTokenUsingPin

void makeCredential(void); 對應到authenticatorMakeCredential

void getAssertion(void); 對應到authenticatorGetAssertion

void getNextAssertion(void); 對應到authenticatorGetNextAssertion

(2) 其他自定義的function：

void genECDHKeyPair(void); 使用libecc裡的function產生一把ECDH用的key-pair。主要是先產生一亂數a，再把它放在橢圓曲線上計算aG。

void genPINToken(void); Spec上是寫產生一亂數，此亂數的長度是16的倍數bytes。而我為了實作方便我這裡就寫死一個16bytes的bytes-array。

void clearPayloadData(void); 清空payloadData相關的變數。

void clearSendBuf(void); 清空sendBuf陣列。

void sendCBORResponse(void); 把payloadData包裝成CBOR格式並回傳response。

void extractReceivedData(void); 把接收到的raw data取出Request中的payloadData的部分。

void computeSharedSecret(void); 把收到的bG乘以authenticator自產的ECDH private key，得到sharedSecret abG。

int isCorrectPIN(u8 *pinHashEnc, u8 encLen); 先把pinHashEnc解密之後得到 LEFT(SHA-256(curPin),16)，再比對authenticator裡的看看是否相同。

void encodeSignature(void);
ECDSA的簽章是由一個32-byte的r和一個32-byte的s所組成。此函數將這個簽章轉換成ANS.1的DER編碼，其結構如下：第一byte是0x30，第二byte是此byte後的byte-length，再來是r的數據結構，最後是s的數據結構。而r和s的數據結構是由0x02開頭，第二byte是此byte後的byte-length，再來是32byte的值，而那個32byte的值如果最左邊的那個byte的MSB是1則要padding，也就是在32byte的值前加上0x00。
舉個例子：
R = [e9, 77, a3, …, 02], S = [19, 65, bc, …, 82]，則此簽章編碼完會變成
{30, 45, 02, 21, 00, [e9, 77, a3, …, 02], 02, 20, [19, 65, bc, …, 82]} 總共71 bytes。

五、code中相依於STM32晶片的部分(未來移植到別的晶片上時需要注意)：

(1) 我code裡很常用到的函式：
USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS, sendBuf, 64); ，
還有 /USB_DEVICE/APP/usbd_custom_hid_if.c 檔案裡的東西，
還有其他有關USB設定的檔案。

(2) 讀取user button是否被按下的函式：

```
if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET) {
    // some code here;
}
```

(3) delay函式：

```
HAL_Delay(150); // delay 150ms
```

(4) 亂數產生器的部分：我是使用.ioc檔裡的晶片設定把RNG開啟(上面有說明過)，之後IDE自動生成code後有關RNG的地方應該都是相依於STM32晶片。而 /Core/Src/libecc/src/external_deps/rand.c 檔中”下面”的static int fimport(unsigned char*buf, u16 buflen); 函式會call到RNG相關的函式，也就是HAL_RNG_GetRandomNumber(&hrng);這個函式。

Testing & Debug

一、描述：

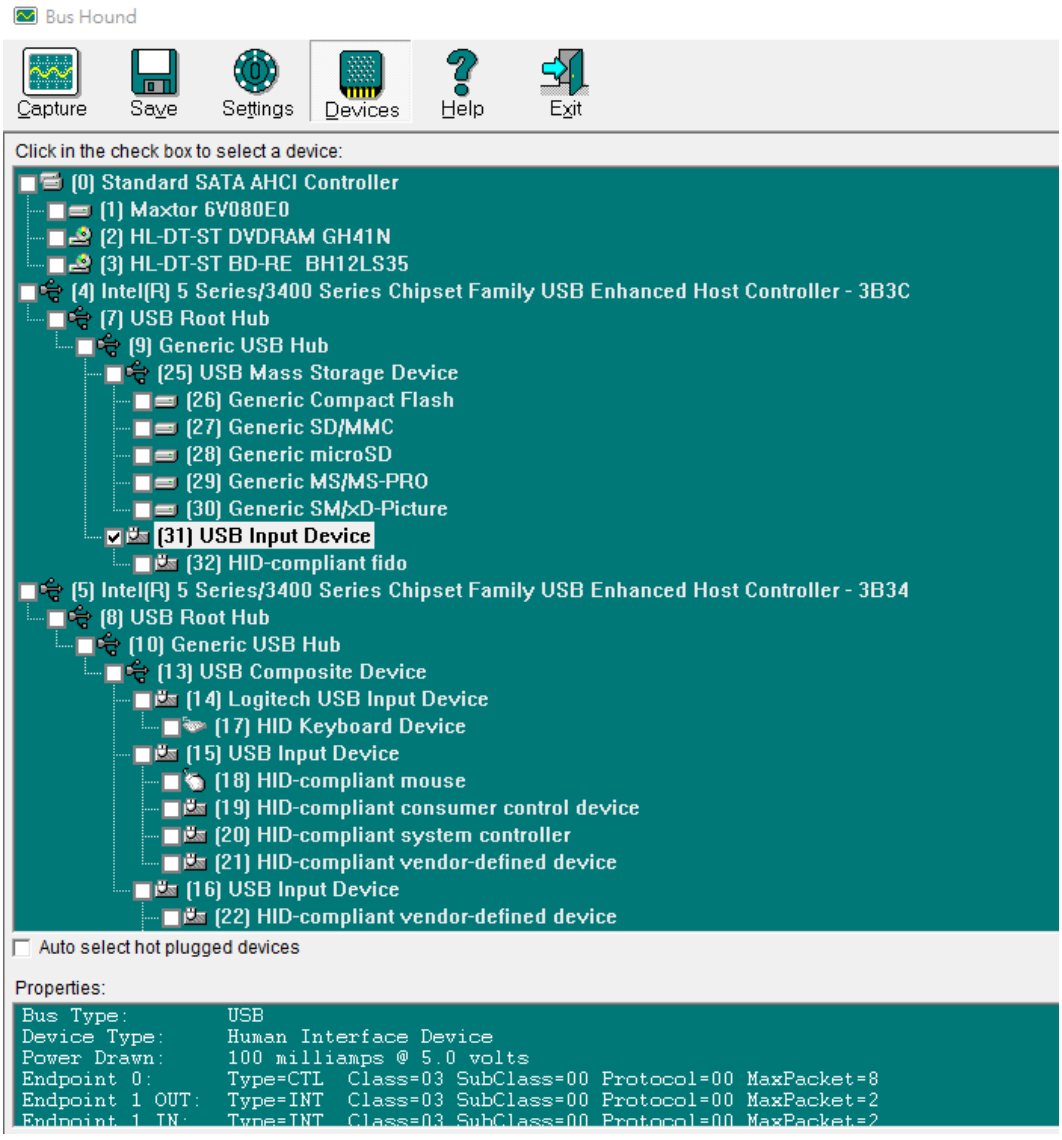
我一共會使用兩個軟體和兩個DEMO網站做測試與Debug。軟體的部分BUS HOUND是負責監聽authenticator與platform互動時USB傳輸的資料，Tera Term是負責USART的輸出也就是程式中printf的部分，至於DEMO網站的部分沒有限定一定要用哪一個，網路上還有其他許多可用來測試的網站。

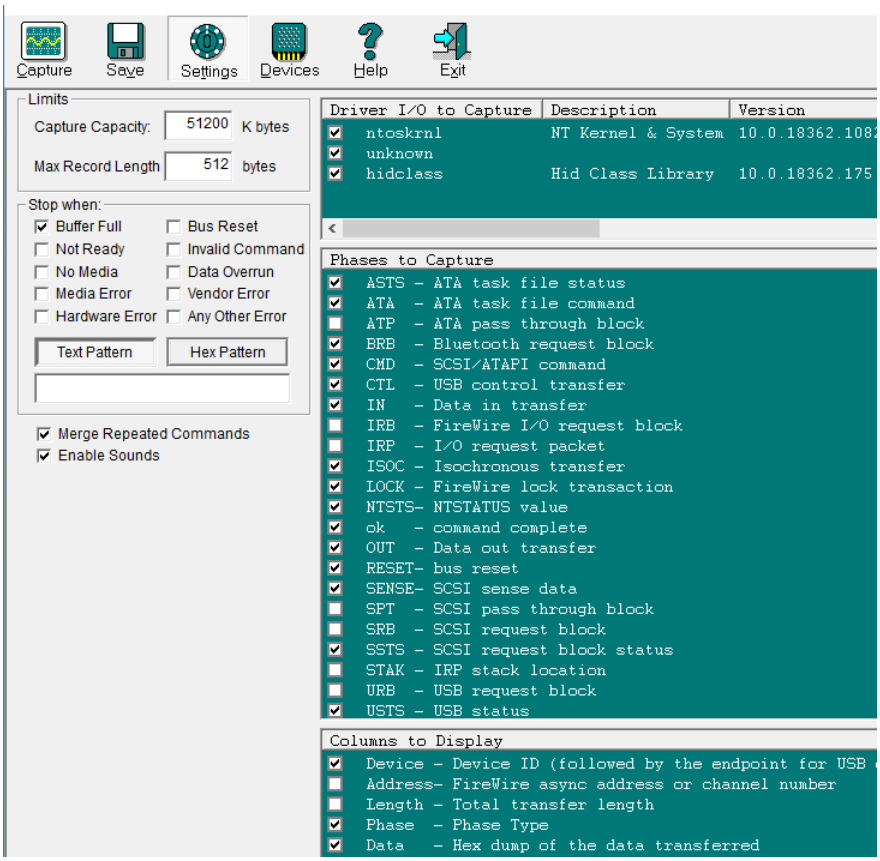
二、BUS HOUND：

http://www.perisoft.net/bushound/details.htm 官網下載，但公司有免費完整版可用。

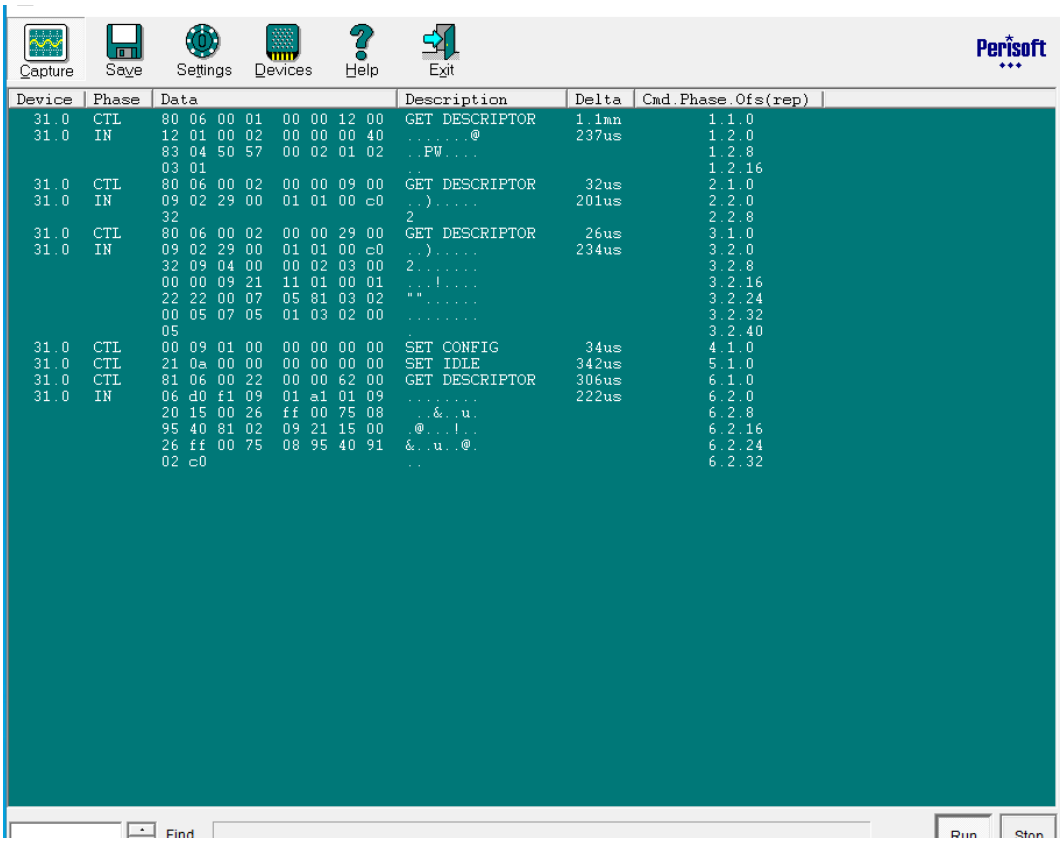
使用方式：

1. 先把板子的B端接上USB線插上電腦，然後BUS Hound的Device介面就會跳出HID-compilant fido，接著勾選它上面的USB Input Device。
2. 進入Settings介面，把Capture Capacity調成51200 Kbytes、Max Record Length調成512 bytes。
3. 接著進入Capture介面，按下Run就可抓取資料。




Capture
Save
Settings
Devices
Help
Exit
Limits
Capture Capacity: 51200 K bytes
Max Record Length 512 bytes
Stop when:
Buffer Full
Bus Reset
Not Ready
Invalid Command
No Media
Data Overrun
Media Error
Vendor Error
Hardware Error
Any Other Error
Text Pattern
Hex Pattern
Merge Repeated Commands
Enable Sounds
Driver I/O to Capture
Description
Version
ntoskrnl NT Kernel & System 10.0.18362.108
unknown
hidclass Hid Class Library 10.0.18362.175
Phases to Capture
ASTS - ATA task file status
ATA - ATA task file command
ATP - ATA pass through block
BRB - Bluetooth request block
CMD - SCSI/ATAPI command
CTL - USB control transfer
IN - Data in transfer
IRB - FireWire I/O request block
IRP - I/O request packet
ISOC - Isochronous transfer
LOCK - FireWire lock transaction
NTSTS- NTSTATUS value
ok - command complete
OUT - Data out transfer
RESET- bus reset
SENSE- SCSI sense data
SPT - SCSI pass through block
SRB - SCSI request block
SSTS - SCSI request block status
STAK - IRP stack location
URB - USB request block
USTS - USB status
Columns to Display
Device - Device ID (followed by the endpoint for USB
Address- FireWire async address or channel number
Length - Total transfer length
Phase - Phase Type
Data - Hex dump of the data transferred

步驟二


Capture
Save
Settings
Devices
Help
Exit
Perisoft
Device Phase Data Description Delta Cmd.Phase.Ofs(rep)
31.0 CTL 80 06 00 01 00 00 12 00 GET DESCRIPTOR 1.1mn 1.1.0
31.0 IN 12 01 00 02 00 00 00 40 ....@ 237us 1.2.0
83 04 50 57 00 02 01 02 ..PW.... 1.2.8
03 01 .. 1.2.16
31.0 CTL 80 06 00 02 00 00 09 00 GET DESCRIPTOR 32us 2.1.0
31.0 IN 09 02 29 00 01 01 00 c0 ..).... 201us 2.2.0
32 2 2.2.8
31.0 CTL 80 06 00 02 00 00 29 00 GET DESCRIPTOR 26us 3.1.0
31.0 IN 09 02 29 00 01 01 00 c0 ..).... 234us 3.2.0
32 09 04 00 00 02 03 00 2..... 3.2.8
00 00 09 21 11 01 00 01 ...!.... 3.2.16
22 22 00 07 05 81 03 02 "".... 3.2.24
00 05 07 05 01 03 02 00 ..... 3.2.32
05 3.2.40
31.0 CTL 00 09 01 00 00 00 00 00 SET CONFIG 34us 4.1.0
31.0 CTL 21 0a 00 00 00 00 00 00 SET IDLE 342us 5.1.0
31.0 CTL 81 06 00 22 00 00 62 00 GET DESCRIPTOR 306us 6.1.0
31.0 IN 06 d0 f1 09 01 a1 01 09 ..... 222us 6.2.0
20 15 00 26 ff 00 75 08 ..&..u. 6.2.8
95 40 81 02 09 21 15 00 .@...!.. 6.2.16
26 ff 00 75 08 95 40 91 &..u..@. 6.2.24
02 c0 .. 6.2.32
Find
Run
Stop

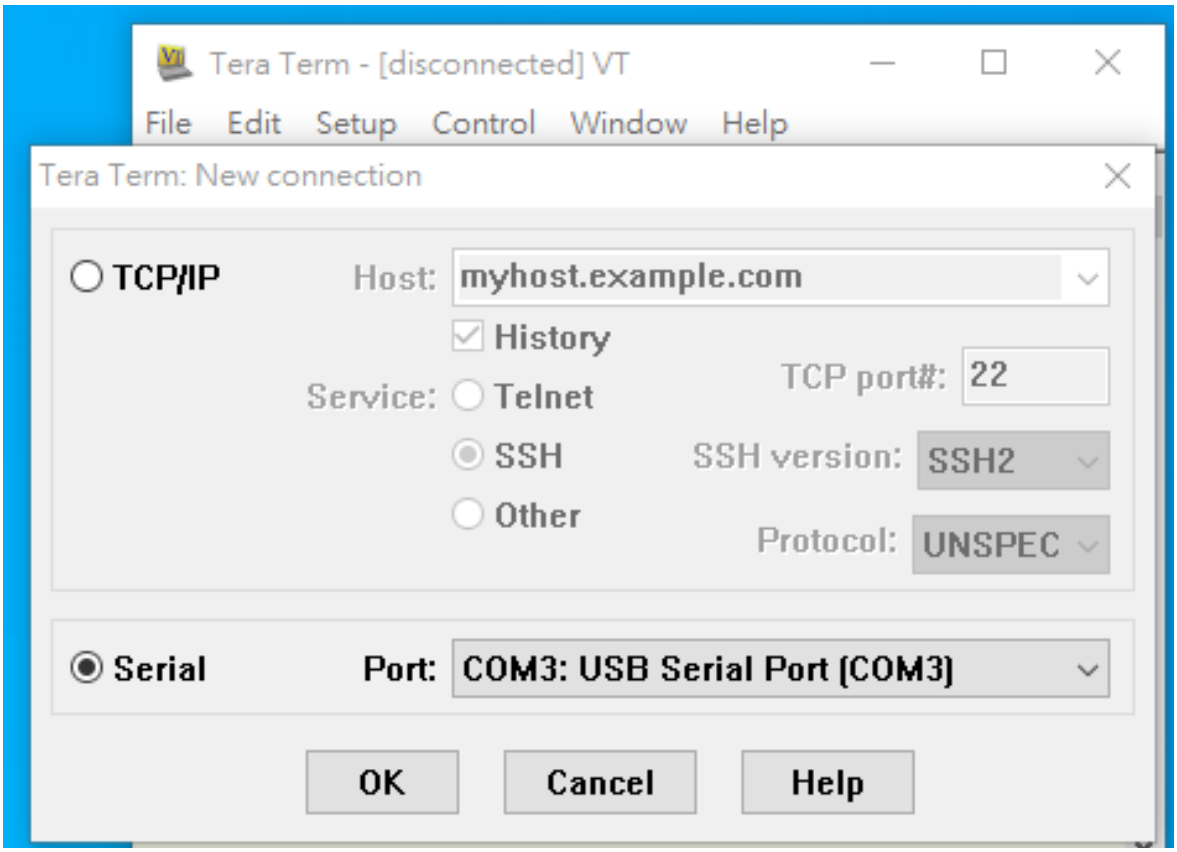
步驟三

三、Tera Term：

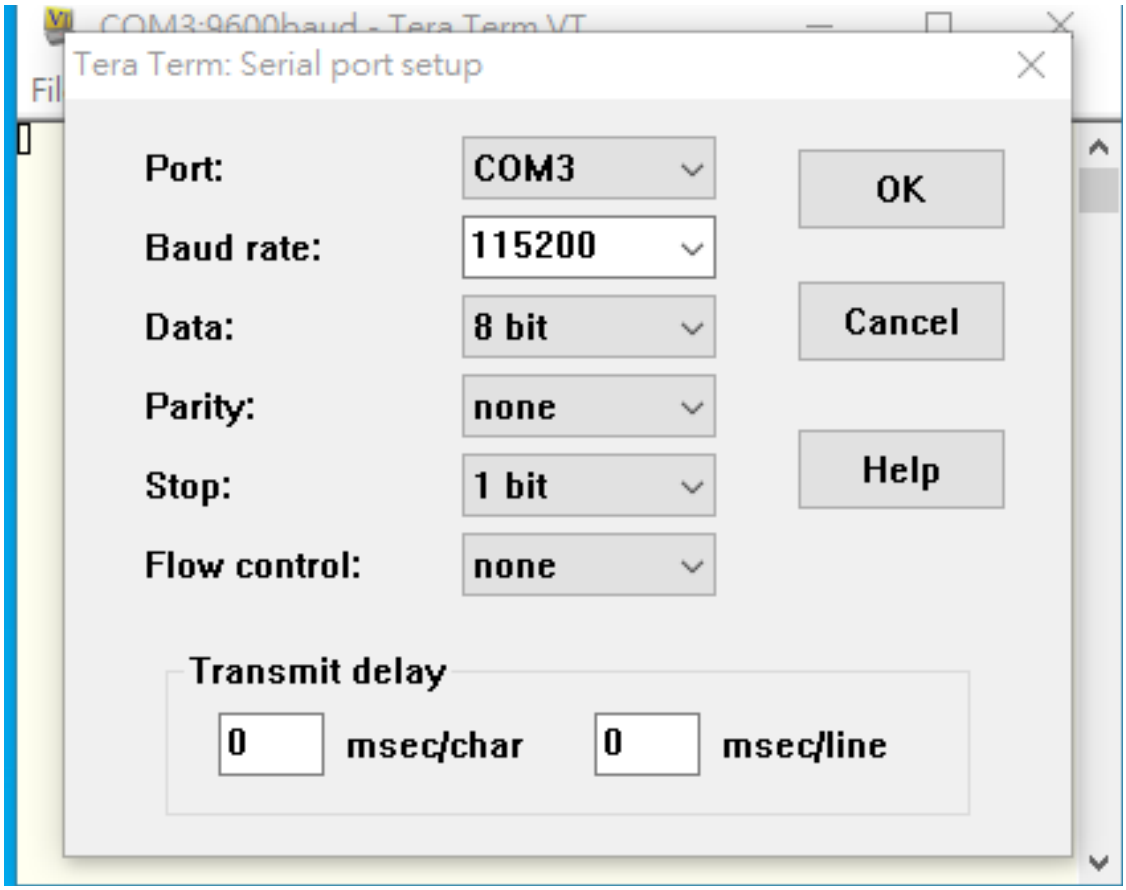
從https://osdn.net/projects/ttssh2/releases/ 下載。

使用方式：

1. 在USART線插入電腦後，開啟Tera Term軟體。
2. 打開軟體後會有以下畫面，點選Serial並選擇COM3: USB Serial Port，按下OK。
3. 進入主頁面後點選上排Setup的Serial Port…，把Baud rate設定成115200，按下OK。
4. 執行程式後，即可把東西print出來。



步驟二


COM3:9600baud - Tera Term VT
Tera Term: Serial port setup
Port:
COM3
Baud rate:
115200
Data:
8 bit
Parity:
none
Stop:
1 bit
Flow control:
none
OK
Cancel
Help
Transmit delay
0
msec/char
0
msec/line

步驟三

四、DEMO網站：

第一個：Webauthn me: https://webauthn.me。
特點：介面清楚、不會檢查憑證來源、不檢查aaguid。
缺點：Register和Login的過程有時間限制(各只有15秒)、無法測到GetNextAssertion API、Login的動作必須接在Register後，也就是說網站不會儲存帳號。

第二個：singular key: https://webauthn.singularkey.com
特點：功能完整、什麼都檢查、網站的server會儲存帳號、時間限制很長。
缺點：因為什麼都檢查，所以在做PoC的時候無法通過aaguid和自簽憑證的部分。

Reference

一、Protocol：

CTAP2.1：https://fidoalliance.org/specs/fido2/fido-client-to-authenticator-protocol-v2.1-rd-20191217.pdf

Web Authn Spec：https://www.w3.org/TR/webauthn/

二、學習USB和USB HID：

USB概述：http://wiki.csie.ncku.edu.tw/embedded/USB

Enumeration： https://wwssllabcd.github.io/blog/2012/11/28/usb-emulation/

學習實作USB-HID的滑鼠：http://aws.micromousetaiwan.org/?p=1968

還有可參考我之前做的CTAP1簡報的前半部。

還有很多零碎的資料忘記了，反正都是google來的。

三、CBOR編碼：

編碼說明：https://en.wikipedia.org/wiki/CBOR

線上Parser：http://cbor.me

四、我所使用的外部library：

libecc：https://github.com/ANSSI-FR/libecc

AES_CBC：https://github.com/kokke/tiny-AES-c

hmac-sha256：https://github.com/aperezdc/hmac-sha256/blob/master/hmac-sha256.c

五、X.509 Certificate：

https://blog.cssuen.tw/create-a-self-signed-certificate-using-openssl-240c7b0579d3

https://stackoverflow.com/questions/10175812/how-to-create-a-self-signed-certificate-with-openssl

X.509我是綜合許多網站教學整理出來的指令，其他的我忘了是參考什麼網站。