Martin Endler, endlemar@fel.cvut.cz

FIDO2 USB Security Key

B4MSVP Final Report

June 2024

Martin Endler

Introduction

Passwords as a means of authentication suffer from many problems. To increase security, *multi-factor* and *passwordless* authentication have been introduced. **FIDO2** is a set of standards based on *asymmetric cryptography* that enables easy, secure, and phishing-resistant authentication. It supports *passwordless*, *second-factor,* and *multi-factor* user experiences with embedded (or bound) authenticators (such as biometrics or PINs) or external (or roaming) authenticators (such as FIDO Security Keys, mobile devices, wearables, etc.). All major OSs, browsers, and a growing number of websites and applications support FIDO2 (or the previous more limited standard FIDO U2F).

Goal

The goal of this project is to create a new open-source implementation of an external FIDO2 authenticator from scratch that would be well-documented, thoroughly tested, and production-ready. By focusing on the quality and the documentation, this implementation could help others understand the FIDO2 standards by providing a detailed yet accessible insight into the inner workings of these protocols (which the existing implementations lack). In general, this project has the potential to contribute to the popularization of FIDO2 technology.

The result of this project will be an open-source software implementation of a FIDO2 USB hardware external authenticator together with the final project report that will document it. The working of the implementation will be demonstrated on a suitable hardware platform (such as STM32F4). However, the hardware implementation is not the focus of this project.

Existing Work

Before we started working on our own implementation, we did an extensive search to see what open-source implementations of FIDO2 authenticators were available.

Open-Source Software and Hardware

SoloKeys

- "the first open source FIDO2 security key"
- **v1** launched on Kickstarter in 2018 and **v2** in 2021
- As of today (February 6, 2024), it seems that the project development has stopped, see this discussion. Kickstarter backers of v2 received their keys in 2023.
- **v1**
  - **https://github.com/solokeys/solo1**
  - **software written in C for STM32L432**
  - Solo: the first open source FIDO2 security key. USB & NFC. by Conor Patrick — Kickstarter (October 3, 2018)
- v2
  - https://github.com/solokeys/solo2
  - Solo V2 — Safety Net Against Phishing by Conor Patrick — Kickstarter (January 26, 2021)
  - software written in **Rust for NXP LPC55S69**

Nitrokey

- Open Source IT-Security Hardware from a German company
- Nitrokey FIDO2
  - https://github.com/Nitrokey/nitrokey-fido2-firmware
  - a fork of **solo1**
- Nitrokey 3
  - https://github.com/Nitrokey/nitrokey-3-firmware
  - developed in collaboration with SoloKeys, similar to Solo 2

OpenSK by Google

- An open-source implementation for security keys written in **Rust** that supports both **FIDO2** and **FIDO U2F**.
- Actively developed by **Google**.
- Written in Rust, uses Tock OS, the main development target platform is **Nordic nRF52840**.

Other Relevant Projects

2 bachelor theses at CTU in Prague (both from CTU FIT):

- [3] Matěj Borský (May 12, 2022). FIDO2 Authentication Simulator.
  - a simulator of a USB FIDO2 authenticator
  - Linux only, uses UHID - User-space I/O driver support for HID subsystem — The Linux Kernel documentation for emulating the authenticator HID device
  - source code **not available** online

- [4] Martin Kolárik (June 4, 2020). FIDO2 KeePass Plugin.
  - **not much relevant to our project**; however, there is a nice overview of FIDO2 in Chapter 1.2
  - deals with the design and development of a plugin for KeePass open-source password manager
  - the plugin should enable the use of a FIDO2 authenticator instead of (in addition to) a master password for unlocking the password database

2 open-silicon designs:

OpenTitan

- OpenTitan: Open source silicon root of trust (RoT) (ASIC / FPGA)
- OpenTitan Big Number Accelerator (OTBN) Technical Specification
- Ibex RISC-V Core Wrapper - OpenTitan Documentation
- Based on this GitHub thread (last activity about six months ago), it seems that someone is working toward getting OpenSK running on the OpenTitan FPGA platform.

TROPIC01

- TROPIC stands for a TRuly OPen IC
- partially open secure element designed by the Czech company Tropical Square, which is fabless ASIC design spinoff of Satoshi Labs
- the TROPIC01 is designed to be used in Satoshi Labs's crypto HW wallets Trezor
  - Some of the Trezor HW wallets can be used as a FIDO2 authenticator. See this about FIDO2, this about U2F, and this comment.
- uses https://github.com/lowRISC/ibex (small open-source 32-bit RISC-V core, also used in the OpenTitan project)
- part of the chip responsible for elliptic cryptography developed by a student at CTU FIT
  - this part is now called SPECT (Secure Processor of Elliptic Curves for Tropic)
  - see this talk Informatické večery: Kryptografický koprocesor nejen pro bitcoinovou hardwarovou peněženku - FIT ČVUT
  - bachelor thesis (the talk above is more up-to-date): Víceúčelová hardwarová platforma pro kryptografii nad eliptickými křivkami
  - paper: Versatile Hardware Framework for Elliptic Curve Cryptography | IEEE Conference Publication (also freely available on CTU DSpace here)

FIDO2 Overview

FIDO2 is a set of standards based on *asymmetric cryptography* that enables easy, secure, and phishing-resistant authentication. It supports *passwordless*, *second-factor,* and *multi-factor* user experiences with embedded (or bound) authenticators (such as biometrics or PINs) or external (or roaming) authenticators (such as FIDO Security Keys, mobile devices, wearables, etc.).

The specifications are:

- Client to Authenticator Protocol (CTAP) by FIDO Alliance
- Web Authentication (WebAuthn) API by World Wide Web Consortium (W3C)

Note: FIDO U2F is a predecessor of FIDO2 that can be used only for two-factor authentication (as a second factor).

Functional Description

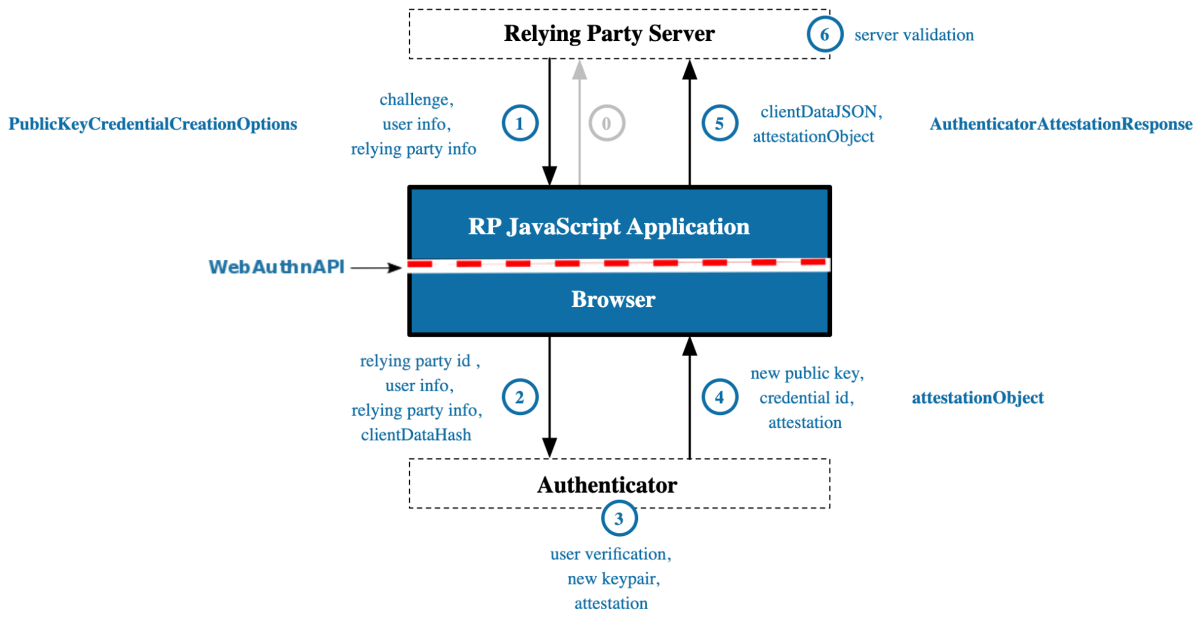
The basic idea is that the credentials belong to the user and are managed by a WebAuthn/FIDO2 **Authenticator**, with which the WebAuthn **Relying Party** interacts through the client platform. Relying Party scripts can (with the user's consent) request the browser to create a new credential for future use by the Relying Party. [1]

The key parts are:

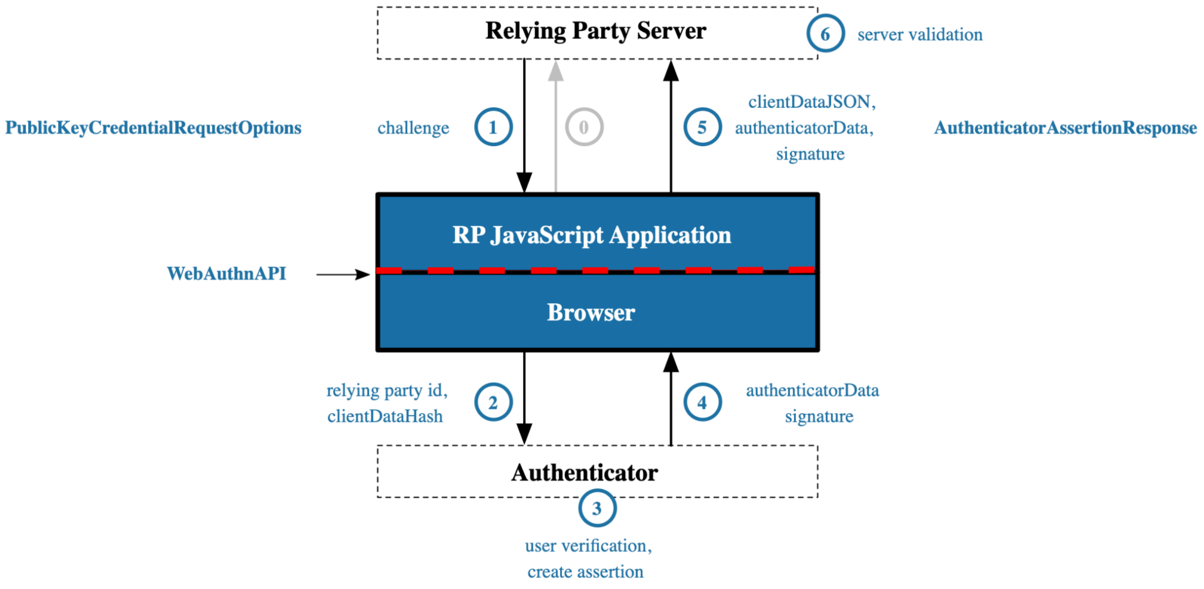
Relying party (RP)

An entity whose application utilizes the WebAuthn API to register and authenticate users, and which stores the public key.

- **register flow** – Authenticator creates a new credential (a public-private key-pair) and shares the public key with RP.


Relying Party Server
6
server validation
PublicKeyCredentialCreationOptions
challenge, user info, relying party info
1
0
5
clientDataJSON, attestationObject
AuthenticatorAttestationResponse
RP JavaScript Application
WebAuthnAPI
Browser
relying party id , user info, relying party info, clientDataHash
2
4
new public key, credential id, attestation
attestationObject
Authenticator
3
user verification, new keypair, attestation

- **authenticate flow** – RP authenticates a user by verifying that the user has access to the private key (which is stored inside the authenticator) that corresponds to the previously registered public key.


Relying Party Server
6 server validation
PublicKeyCredentialRequestOptions
challenge
1
0
5
clientDataJSON, authenticatorData, signature
AuthenticatorAssertionResponse
RP JavaScript Application
WebAuthnAPI
Browser
relying party id, clientDataHash
2
4
authenticatorData signature
Authenticator
3
user verification, create assertion

Authenticator

A cryptographic entity that handles generating and storing keys and performing cryptographic operations.

The private keys never leave the authenticator.

Credential Storage Modality

The keys might be either stored in persistent storage embedded in the authenticator (necessary for client-discoverable/resident credentials that can be used as a first factor), or they might be derived from the credential ID (then the authenticator can support virtually an unlimited number of credentials).

For more information, see 6.2.2. Credential Storage Modality in [1].

Client

An entity that acts as an intermediary between the relying party and the authenticator (typically a **web browser** or a similar application).

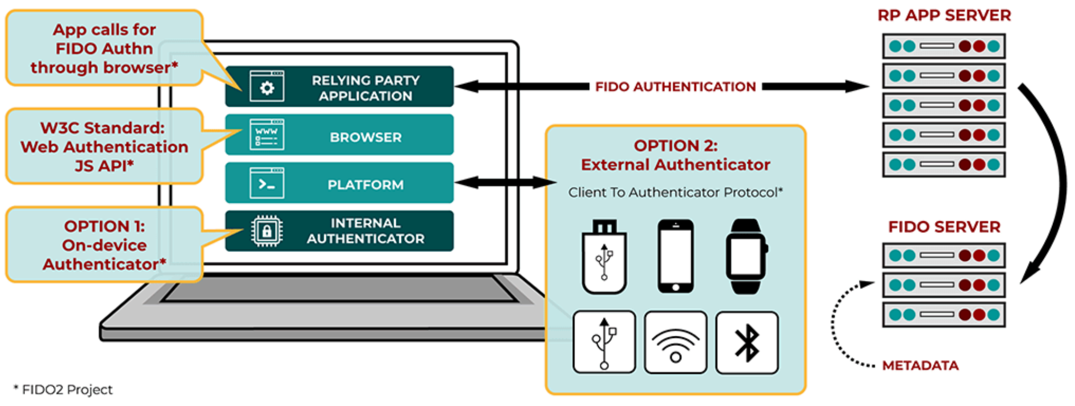
Client Device

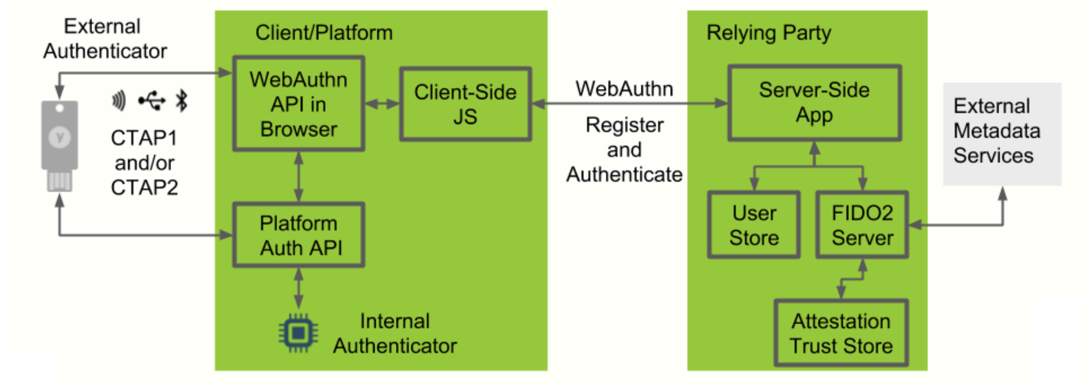
The hardware device on which the WebAuthn Client runs, for example a smartphone, a laptop computer or a desktop computer, and the operating system running on that hardware.

Client Platform

A client device and a client together make up a client platform. A single hardware device may be part of multiple distinct client platforms at different times by running different operating systems and/or clients.

Here are two diagrams that illustrate the above-mentioned concepts (both diagrams depict the same things):


App calls for FIDO Authn through browser*
W3C Standard: Web Authentication JS API*
OPTION 1: On-device Authenticator*
RELYING PARTY APPLICATION
BROWSER
PLATFORM
INTERNAL AUTHENTICATOR
FIDO AUTHENTICATION
OPTION 2: External Authenticator
Client To Authenticator Protocol*
RP APP SERVER
FIDO SERVER
METADATA
* FIDO2 Project



Additional Resources

A nice description of FIDO2 can be found in Chapter 1.2 of [4].

Another description of FIDO2 (in Czech) can be found in Chapter 4 of [3].

User Authentication Specifications Overview – FIDO Alliance

How FIDO Works – FIDO Alliance

WebAuthn Introduction – developers.yubico.com

Examples

There are many hardware security keys from different manufacturers available on the market. Usually, they support communication over USB (either USB-A or USB-C) and NFC.

Below, you can see the two FIDO2 security keys I own and use:

Security Key NFC by Yubico	**GoTrust Idem Key**
https://www.yubico.com/product/security-key-nfc-by-yubico/	https://gotrustid.com/products-idem-key/
USB, NFC	USB, NFC
FIDO2 CTAP1, FIDO2 CTAP2, FIDO Universal 2nd Factor (U2F), FIDO2	FIDO2 CTAP1, FIDO2 CTAP2, FIDO Universal 2nd Factor (U2F), FIDO2
	FIDO2 Security Level 2 **FIPS 140-2 Level 3**

Implementation

This chapter describes the main parts of our FIDO2 authenticator implementation.

Main points:

→ Written in **C**.

→ Runs on the STM3240G-EVAL board with the STM32F407IGH6 MCU.

→ Uses STM32CubeF4 (HAL, LL, and USB Device Library) via STM32CubeMX generator.

→ **The complete source code is available on GitHub here:**
https://github.com/pokusew/fel-krp-project

→ FIDO2 (specifically CTAP2 and CTAPHID) implementation in fido2 dir is based on the SoloKeys Solo 1 project. However, some parts of the original implementation have been **rewritten / refactored / modified / fixed**, so it could be used with the STM32F4 MCU (which has, for example, a different flash memory organization).

Architecture

In this section we describe individual layers (protocol stack) of the firmware that together implement the necessary FIDO2 functionality. The diagram on the right depicts these layers.

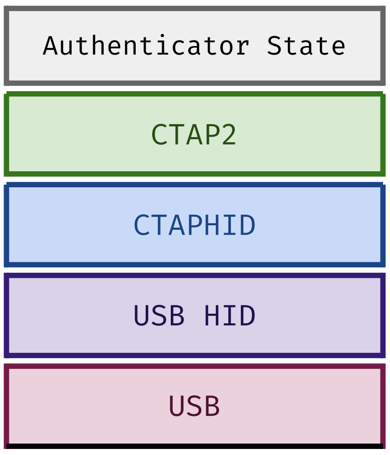


Figure: Protocol Stack

CTAP2

The core part is the implementation of the **CTAP2** protocol. CTAP2 protocol is a high-level transaction-oriented protocol. A CTAP2 **transaction** consists of a *request*, followed by a *response* **message**. CTAP2 transactions are always initiated by the client. Messages are encoded using the concise binary encoding CBOR.

There are two essential transactions – authenticatorMakeCredential, which is implemented in ctap_make_credential, and authenticatorGetAssertion, which is implemented in ctap_get_assertion. They facilitate the register and authenticate flows as described in the Relying party (RP) section.

CTAPHID

Mapping of messages to the underlying USB transport is facilitated by the **CTAPHID** layer. Request and response **messages** are divided into individual fragments, known as **packets**. Packets are the smallest form of protocol data units, which in the case of CTAPHID are mapped into USB **HID reports**. CTAPHID also implements logical channel **multiplexing** so that multiple **clients** (browsers, apps, OS) on the same client device (device) can communicate with the authenticator concurrently.

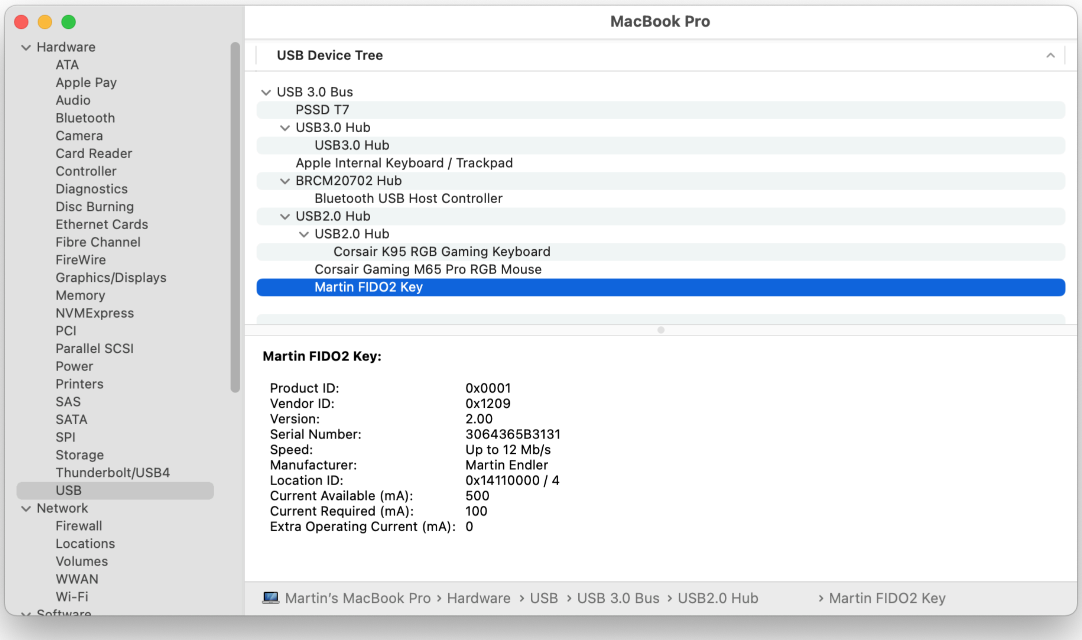
USB

The **USB HID** and **USB** layers are implemented using the STM32Cube USB device library and its *Custom HID* class. It uses the USB_OTG_FS (**USB 2.0 Full Speed**) MCU peripheral.

There are **two endpoints (IN, OUT) with interrupt transfer** (64-byte packet max, poll every 5 millisecond). The descriptors (device, config, interface, endpoints, HID report) are set up according to the CTAPHID specification (11.2.8. HID device implementation).

The USB HID report descriptor plays a key role in the automatic *device discovery*. The CTAPHID protocol is designed with the objective of driver-less installation on all major host platforms. Browsers and other clients use the standard USB HID driver included within the OS. The HID *Usage Page* field within the authenticator's HID report descriptor is set to the value `0xF1D0`, which is registered to the FIDO Alliance (see HID Usage Tables 1.5).

The following screenshot shows the authenticator in the USB Device Tree Explorer in System Information on macOS 11.



```
USB 3.0 Bus
  PSSD T7
  USB3.0 Hub
    USB3.0 Hub
  Apple Internal Keyboard / Trackpad
  BRCM20702 Hub
    Bluetooth USB Host Controller
  USB2.0 Hub
    USB2.0 Hub
      Corsair K95 RGB Gaming Keyboard
    Corsair Gaming M65 Pro RGB Mouse
    Martin FIDO2 Key
```

Martin FIDO2 Key:

```
Product ID:                   0x0001
Vendor ID:                    0x1209
Version:                      2.00
Serial Number:                3064365B3131
Speed:                        Up to 12 Mb/s
Manufacturer:                 Martin Endler
Location ID:                  0x14110000 / 4
Current Available (mA):       500
Current Required (mA):        100
Extra Operating Current (mA): 0
```

Martin's MacBook Pro › Hardware › USB › USB 3.0 Bus › USB2.0 Hub › Martin FIDO2 Key

The chosen Vendor ID (VID) and Product ID (VID) come from the pid.codes project.

pid.codes is a registry of USB PID codes for open source hardware projects. They assign PIDs on any VID they own to any open-source hardware project needing one.

For the initial version of this project, we used their testing VID/PID `0x1209`/`0x0001`. In the future, when we have a more-production-ready authenticator, we might eventually ask for our own PID.

Authenticator State

The top-most block in the diagram represents the authenticator and its internal state.

Persistent Storage

The most complex part is the implementation of **state persistence**. There are multiple pieces of data that need to be persisted. Namely it is:

- the master key (used for storing non-discoverable credentials, see more info in Credential Storage Modality),
- the PIN (or the information that it is not has not been set yet) for user verification (UV) and invalid PIN attempts counter (note that the PIN is not stored in plaintext, instead it is stored as a salted hash),
- the global signature counter,
- and client-side discoverable credentials (formerly called resident credentials, the abbreviation `rk`, which is used in the code, stands for **r**esident **k**ey).

Note: This document uses the following notation: 1 KB (kilobyte) = 1024 B (bytes).

As a persistent storage, we used **a part of the 1024 KB of Flash memory** that is built into the STM32F407IGH6 MCU. By default, it is used for storing the program code (firmware). Because our firmware takes only about the first 104540 B (~ 102 KB), i.e. around 10% of the total capacity, we can use the rest for our application data.

However, flash memory comes with **an important limitation** – one can only program (write) flash bits from 1 to 0. In order to write 1 in place of the already written 0 bits, one must **erase the whole memory sector** (the size of which can be in the range of KBs). This is generic to flash memory technology and is not specific to the STM32.

In case of STM32F407IGH6, its flash memory organization is as follows (also documented in Inc/flash.h):

> A main memory block divided into 12 sectors (with **different** sizes, numbered 0-11):
>
> - `4 sectors of  16 KB: sectors   0-3`
> - `1 sector  of  64 KB: sector      4`
> - `7 sectors of 128 KB: sectors  5-11`

Our program occupies the sectors 0-4 (102 KB fits into the first 128 KB). So, we are only left with the big 128KB sectors for the application data. **Their erasure takes a significant amount of time** (seconds). Also, the total possible number of erasures is limited (**flash memory wear**). Thus, it is not possible to naively rewrite the whole sector when only a few bytes need to be changed (for example, when the global signature counter needs to be incremented).

There are special file systems designed to allow efficient use of such memories. However, for our use case, **we implemented a simple custom solution, adapted from the Solo 1 project** (where they faced the same challenge, but their MCU's sectors were only 1 KB big which still allowed for relatively fast erasures).

The implemented algorithms are conceptually types of **simple wear-leveling algorithms**. With our solution, the need for erasures is greatly reduced. Most of the time, all the changes in the authenticator state can be done without any erasure at all (so it takes no extra time).

Here are the relevant parts of the code that implement the persistence:

- flash memory utils in Inc/flash.h and Src/flash.c
- Inc/memory_layout.h that defines the used sector numbers
- ensure_flash_initialized() in Src/app.c for initialization of the memory layout upon the very first startup and after memory reset (using the app_delete_data debug command)
- persistence of the AuthenticatorState struct (contains the master key + PIN data)
  - authenticator_read_state()
  - authenticator_write_state()
- persistence of the global signature counter
  - ctap_atomic_count() in Src/device.c
  - Note: A special effort has been made to make the counter atomic. In case the flash write is interrupted (e.g., the authenticator is disconnected from the USB and the power is interrupted), the counter should keep its previous value. This is critical for the correct FIDO2 implementation because the counter must never be decremented.
- persistence of the client-discoverable credentials (resident credentials, resident keys)
  - ctap_reset_rk()
  - ctap_rk_size()
  - ctap_store_rk()
  - ctap_delete_rk()
  - ctap_load_rk()
  - ctap_overwrite_rk()
    This is the only part where the wear-leveling algorithm is not implemented yet. Fortunately, it has only a negligible impact, since the ctap_overwrite_rk is only called once when a new client-discoverable credential is being created. This is in contrast with ctap_atomic_count() and authenticator_read_state() that can be invoked even multiple times during a single transaction.

True Random Number Generator

We used the RNG peripheral to generate true random numbers for cryptographic purposes.

The RNG peripheral available in the STM32F407IGH6 MCU is a random number generator, based on a continuous analog noise, that provides a random 32-bit value to the host when read.

The RNG passed the FIPS PUB 140-2 (2001 October 10) tests with a success ratio of 99%.

The relevant parts of the code:

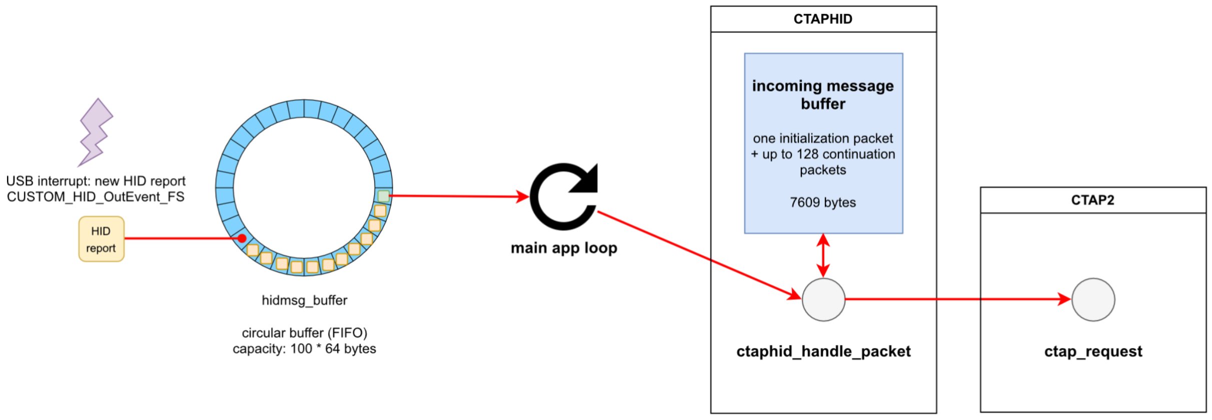
- rng_get_bytes() in Src/rng.c

Data Flow

When a new USB HID report packet (in the code we also call it HID msg) arrives from the host to the authenticator, the function CUSTOM_HID_OutEvent_FS is invoked (within the interrupt context). It takes the report (64 bytes) and puts it in the `hidmsg_buffer` circular buffer (FIFO) that has capacity of 100 * 64 bytes.

The main app loop regularly checks the buffer and if there is an unprocessed message (report), it passes it to the CTAPHID layer's ctaphid_handle_packet function. The CTAPHID layer reconstructs the CTAP2 messages from the individual packets. When a CTAP2 **request** message is ready to be processed it is passed to the CTAP2 layer. After the CTAP2 layer processes the request, the response message is sent (it is split into individual USB HID packets (reports) which are sent one by one by usbhid_send).

The incoming data path (from the host to the authenticator) is visualized in the following diagram:

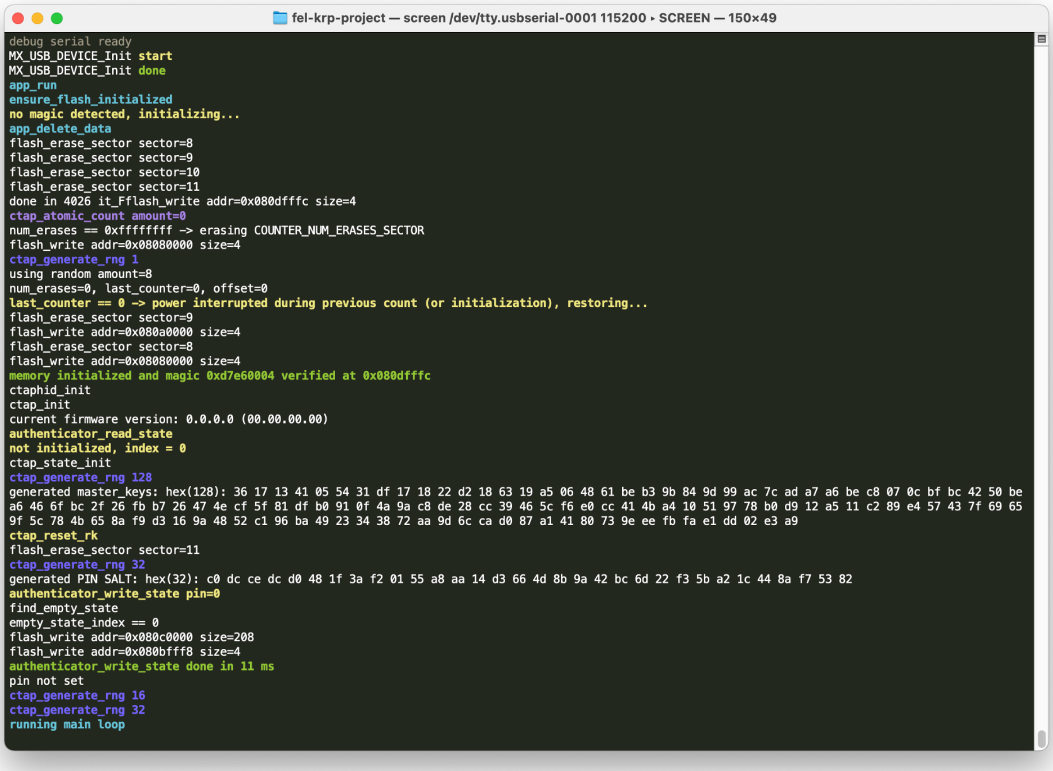


Note that when checking the `hidmsg_buffer` (or taking data from it), the interrupts (or at least the USB interrupt) must be disabled to prevent race conditions (because new data could be written at any time by the CUSTOM_HID_OutEvent_FS that runs in the interrupt context).

Debugging Interface via UART

In order to allow easy debugging (and development of the USB communication stack), I implemented a simple **bidirectional** debugging interface using UART (USART3 peripheral). We wrote a simple write syscall implementation so that the `printf` statements (wrapped in custom `debug/info/error_log` macros) could work as expected. The handling of the incoming debug commands is done in the app main loop here.

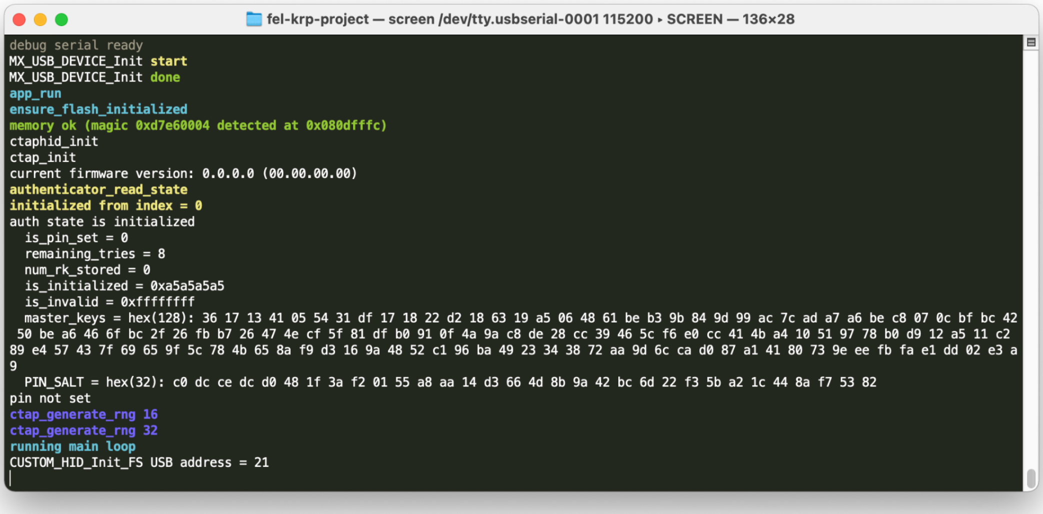
Here is a screenshot of the initial debug logs upon authenticator's first startup when the memory has not been initialized yet:


fel-krp-project — screen /dev/tty.usbserial-0001 115200 ▸ SCREEN — 150×49

```
debug serial ready
MX_USB_DEVICE_Init start
MX_USB_DEVICE_Init done
app_run
ensure_flash_initialized
no magic detected, initializing...
app_delete_data
flash_erase_sector sector=8
flash_erase_sector sector=9
flash_erase_sector sector=10
flash_erase_sector sector=11
done in 4026 it_Fflash_write addr=0x080dfffc size=4
ctap_atomic_count amount=0
num_erases == 0xffffffff -> erasing COUNTER_NUM_ERASES_SECTOR
flash_write addr=0x08080000 size=4
ctap_generate_rng 1
using random amount=8
num_erases=0, last_counter=0, offset=0
last_counter == 0 -> power interrupted during previous count (or initialization), restoring...
flash_erase_sector sector=9
flash_write addr=0x080a0000 size=4
flash_erase_sector sector=8
flash_write addr=0x08080000 size=4
memory initialized and magic 0xd7e60004 verified at 0x080dfffc
ctaphid_init
ctap_init
current firmware version: 0.0.0.0 (00.00.00.00)
authenticator_read_state
not initialized, index = 0
ctap_state_init
ctap_generate_rng 128
generated master_keys: hex(128): 36 17 13 41 05 54 31 df 17 18 22 d2 18 63 19 a5 06 48 61 be b3 9b 84 9d 99 ac 7c ad a7 a6 be c8 07 0c bf bc 42 50 be
a6 46 6f bc 2f 26 fb b7 26 47 4e cf 5f 81 df b0 91 0f 4a 9a c8 de 28 cc 39 46 5c f6 e0 cc 41 4b a4 10 51 97 78 b0 d9 12 a5 11 c2 89 e4 57 43 7f 69 65
9f 5c 78 4b 65 8a f9 d3 16 9a 48 52 c1 96 ba 49 23 34 38 72 aa 9d 6c ca d0 87 a1 41 80 73 9e ee fb fa e1 dd 02 e3 a9
ctap_reset_rk
flash_erase_sector sector=11
ctap_generate_rng 32
generated PIN SALT: hex(32): c0 dc ce dc d0 48 1f 3a f2 01 55 a8 aa 14 d3 66 4d 8b 9a 42 bc 6d 22 f3 5b a2 1c 44 8a f7 53 82
authenticator_write_state pin=0
find_empty_state
empty_state_index == 0
flash_write addr=0x080c0000 size=208
flash_write addr=0x080bfff8 size=4
authenticator_write_state done in 11 ms
pin not set
ctap_generate_rng 16
ctap_generate_rng 32
running main loop
```

You can notice that the authenticator generates a new master key upon its first startup. This master key is used for storing non-discoverable credentials.

On subsequent startups, that initialization is skipped when initialized memory is detected:


fel-krp-project — screen /dev/tty.usbserial-0001 115200 ▸ SCREEN — 136×28

```
debug serial ready
MX_USB_DEVICE_Init start
MX_USB_DEVICE_Init done
app_run
ensure_flash_initialized
memory ok (magic 0xd7e60004 detected at 0x080dfffc)
ctaphid_init
ctap_init
current firmware version: 0.0.0.0 (00.00.00.00)
authenticator_read_state
initialized from index = 0
auth state is initialized
  is_pin_set = 0
  remaining_tries = 8
  num_rk_stored = 0
  is_initialized = 0xa5a5a5a5
  is_invalid = 0xffffffff
  master_keys = hex(128): 36 17 13 41 05 54 31 df 17 18 22 d2 18 63 19 a5 06 48 61 be b3 9b 84 9d 99 ac 7c ad a7 a6 be c8 07 0c bf bc 42
 50 be a6 46 6f bc 2f 26 fb b7 26 47 4e cf 5f 81 df b0 91 0f 4a 9a c8 de 28 cc 39 46 5c f6 e0 cc 41 4b a4 10 51 97 78 b0 d9 12 a5 11 c2
89 e4 57 43 7f 69 65 9f 5c 78 4b 65 8a f9 d3 16 9a 48 52 c1 96 ba 49 23 34 38 72 aa 9d 6c ca d0 87 a1 41 80 73 9e ee fb fa e1 dd 02 e3 a
9
  PIN_SALT = hex(32): c0 dc ce dc d0 48 1f 3a f2 01 55 a8 aa 14 d3 66 4d 8b 9a 42 bc 6d 22 f3 5b a2 1c 44 8a f7 53 82
pin not set
ctap_generate_rng 16
ctap_generate_rng 32
running main loop
CUSTOM_HID_Init_FS USB address = 21
```

There are several debug commands, each one of them is assigned to one **lowercase** letter:

- **`l`** `(lowercase L) – toggle the Blue LED`
- **`t`** – `app_test_time`
- **`g`** – `app_test_rng`
- **`f`** – `app_flash_info`
- **`c`** – `app_test_ctap_atomic_count`
- **`s`** – `app_test_state_persistence`
- **`d`** – **`app_delete_data`**

  This command deletes all data (client-discoverable credentials, global counter) that are stored in the persistent flash memory. After that, Reset is needed upon which the memory will be reinitialized.

The four on-board LEDs are used as status indicators:

- **Green** – main loop running
- **Orange** – turned on USB device has been initialized and assigned an address
- **Blue** – toggle using UART debug char l (lowercase L)
- **Red** – ErrorHandler
  - This indicates that some unexpected fatal error occurred.
    Press the reset button. The device will not work until reset.

Conclusion

In this project, **we successfully implemented** a working FIDO2 USB hardware external authenticator (also called FIDO2 USB security key).

First, we outlined the goal of the project and its motivation. Then, we established the necessary theoretical foundation by describing the Web Authentication and CTAP2 standards, which are together known as FIDO2. We also reviewed existing projects related to our goal.

Then, we proceeded to the actual implementation, which represents the main contribution of this project. We documented our decisions and the most important parts of the implementation. All code is versioned using Git and is publicly **available online on GitHub**. The repository includes build and run instructions. There is also a **CI/CD pipeline** to ensure software quality. Our authenticator supports both client-discoverable credentials and an unlimited number of non-discoverable credentials. Thanks to the full support of client-discoverable credentials and user verification (UV) using PIN, it can be used as a first factor (passkey).

Finally, **we tested** our authenticator **with real websites** with different configurations (1st factor vs 2nd factor). In all cases, **it worked great**, including with Google Account and GitHub.

However, the authenticator is not production-ready. Even though we were able to make it work (with the help of the Solo 1 codebase), there are a lot of details and edge cases that are not correctly handled. The whole FIDO2/WebAuthn specification is very complex and a proper implementation would take much more time.

Future Work

There are a few more steps needed to achieve the ultimate goal of creating a new FIDO2-compliant implementation from scratch that would be thoroughly tested and production-ready.

Currently, our implementation uses a part of the Solo 1 project (the CTAP2 layer) that we need to replace with our own implementation.

Furthermore, we would like to use test-driven development and ensure there is sufficient test coverage. As of now, we have set up a CI/CD pipeline (so at least we can test that project builds correctly), but there are no unit/integration/e2e tests yet.

Ultimately, we would like to attempt to pass the FIDO Authenticator Level 1 (L1) certification.

References

[1] W3C (April 8, 2021). Web Authentication: An API for accessing Public Key Credentials – Level 2.
W3C Recommendation.
https://www.w3.org/TR/webauthn-2/
Note: There are newer working versions of this standard, see the links in the document.

[2] FIDO Alliance (June 21, 2022). Client to Authenticator Protocol (CTAP).
CTAP 2.1 Proposed Standard.
https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html
Note: See https://fidoalliance.org/specifications/download/ for the list of the current published standards. Namely, there is already a newer version available (CTAP 2.2 Review Draft 01).

[3] Matěj Borský (May 12, 2022). FIDO2 Authentication Simulator.
https://dspace.cvut.cz/handle/10467/101916
https://dspace.cvut.cz/bitstream/handle/10467/101916/F8-BP-2022-Borsky-Matej-thesis.pdf

[4] Martin Kolárik (June 4, 2020). FIDO2 KeePass Plugin.
https://dspace.cvut.cz/handle/10467/88264
https://dspace.cvut.cz/bitstream/handle/10467/88264/F8-BP-2020-Kolarik-Martin-thesis.pdf