Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Security Analysis of OnlyKey

Bc. Josef Hušek

Department of Information Security
Supervisor: Ing. Josef Kokeš

June 23, 2022

Acknowledgements

I would especially like to think my supervisor Ing. Josef Kokeš, for having patience with me and for lending me the OnlyKey Original, so that I could study it.

I would also like to thank my family, friends and also my fiancée, for providing me with continued emotional support.

Lastly I would like to thank Tim Steiner and the full CryptoTrust team for creating the OnlyKey, it is indeed an awesome device.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 23, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Hušek, Josef. *Security Analysis of OnlyKey.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Tato práce se zabývá zařízením zvaným OnlyKey, které slouží jako malý osobní autentifikační token. Jeho hlavním účelem je fungovat jako dedikovaný hardwarový správce hesel, včetně podpory pro dvoufázové ověření uživatele. Nejprve se podíváme jaký hardware toto zařízení používá a rozebereme si nějaké obecné informace o zařízení, včetně jeho inzerovaných schopností. Poté se zaměříme na několik open-source kódů, které společně tvoří zázemí umožňující zamýšlenou funkčnost zařízení a pronikneme do toho jak fungují a jak spolu spolupracují. V poslední řadě budeme diskutovat zajímavá fakta, která jsme odhalili během studia tohoto zařízení, a to především z pohledu bezpečnosti uživatele.

Klíčová slova heslo, password manager, autentifikace, dvoufaktorová autentifikace, dedikovaný hardware, OnlyKey, CryptoTrust, PIN, Teensy, cryptography, bezpečnost, soukromí

Abstract

This thesis is about the OnlyKey device, which is a small personal authentication token. Its main function is acting as a dedicated hardware password manager, including support for two-factor-authentication. We will first discuss a bit about what hardware it is based on and some general information about the device including its advertised capabilities. After this we will shift our attention to the several open-source software pieces, which make the device work as intended and delve into how they work and interact with each other. Lastly we will discuss what interesting facts we found during the study of this device, especially pertaining to user security.

Keywords password, password manager, authentication, two-factor authentication, dedicated hardware, OnlyKey, CryptoTrust, PIN, Teensy, cryptography, security, privacy

Contents

List of Figures

Introduction

In today's internet-age society, there is a great need for authentication. All around the Web - and other online services - communicating parties want to make sure that they are actually talking with the intended partner and not an impostor.

There are several ways to solve the authentication problem - the traditional one is to use passwords. In this case the user remembers a password or a passphrase which the other party can verify in some pre-determined way.

This becomes problematic when the user wants to use a lot of different services - it is insecure in practice to use the same password across different services, but at the same time it is infeasible for most people to remember a lot of unique and strong passwords.

Therefore one of the solutions is to store all password somewhere, where only the user can access it. Some people might solve this by writing their passwords down on a piece of paper or in a personal notebook/diary, etc. This solution can work well in theory, as long as the user can guarantee that the list of passwords won't be lost and nobody else will be able to access it (these conditions are difficult to meet though). Even if these are satisfied - it is definitely not a very comfortable solution. Each time the user wants to login to one of their services, they will have to physically take their list, find the correct password and then manually write it on their keyboard.

A more practical and modern approach would be using a special software called a password manager. This is a computer program which stores all the passwords on a hard drive or in the cloud in an encrypted form, requiring another password to gain access and decrypt them. This reduces the amount of passwords needed to remember down to one, while still using strong unique passwords for all the different services. Another advantage can be that - depending on the program used - the manager may allow the user to copy-paste the requested password, or even type it in a selected text field automatically on behalf of the user. A disadvantage of this system is that if a potential attacker gains access to the encrypted database in any way, they will be able

to steal all the passwords. This could happen either by cracking the main password via brute-force or dictionary attacks, or by somehow gaining access to the legitimate users PC (either remotely or locally) and installing some kind of keylogger - a malicious piece of software that records keystrokes. Another way might be bypassing the encryption altogether if, the chosen encryption function is weak, or if there are problems with the implementation which render its security useless.

The majority of these problems can be addressed by using a hardware based password manager instead - a separate device dedicated for personal authentication which preferably can't connect to the Internet at all. This device would then be physically carried by the user and when needed would somehow transfer the needed password(s) to a target platform (PC, smartphone, ...) where the user is trying to authenticate themselves.

Besides passwords and passphrases there are other methods of authentication. These can either be used along with a password, to provide a second/multi-factor authentication or without a password to provide a "passwordless" authentication experience[1]. Usually we talk about 3 categories of authentication - "what you have" (a specific HW device, a phone), "what you know" (a password, a sceurity question) and "what you are" (fingerprints, retina).

In this thesis I am mostly interested in the "what you have" category, since the main topic is a physical device which the user has and carries with him. The device in question is called the OnlyKey and allows among other things password management as well as the option to serve as a second factor security key.

Chapter 1

Hardware and general information

1.1 The Teensy

"The Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port."[2]

Teensy is an independent commercial project created and maintained by Paul Stoffregen. He and his partner Robin Coon share a website[1] that is used to document and sell the various Teensy devices, as well as showcase different projects using the device. The website also hosts among other things a personal blog which is not of interest to us, however there is also a subdomain[2] which contains the user forum and which is a great source of additional information on the Teensy.

Specifically I am interested in Teensy 3.2 (pictured on 1.1 and 1.2), since that is the version that OnlyKey is based on. It uses the "MK20DX256VLH7" 32-bit chip by NXP semiconductors (formerly Freescale semiconductors)[3]. It provides the user with 3 types of memory - 64 kilobytes of RAM, 256 kilobytes of flash and 2 kilobytes of EEPROM. The EEPROM is actually emulated in the last 22528 bytes of the flash memory and wear leveling is performed automatically, to increase its lifespan [4][5]. The RAM serves its traditional purpose of storing currently used data and variables, flash memory is primarily meant to store the user program and the EEPROM should store data that needs to persist between reboots and possible losses of power. The chip provides a security feature designed to protect the memory from being overwritten externally after the proper setup - this technology is called Kinetis Security - its usage in the OnlyKey software will be discussed later.

[1] https://www.pjrc.com
[2] https://www.forum.pjrc.com

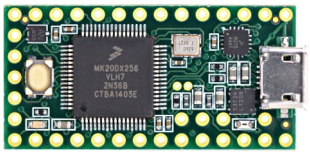

Figure 1.1: The Teensy 3.2 device front (downloaded from the PJRC website[6])

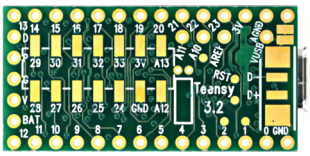

Figure 1.2: The Teensy 3.2 device back (downloaded from the PJRC website[6])

The device also has a separate chip which houses the HalfKay bootloader. This is a proprietary closed-source[7] bootloader which is used for loading the main user program to the Teensy [8] via the USB port, using a small cross-platform application called "Teensy loader" [9]. The Teensy can communicates with a PC (or another host device) using a "Raw HID" protocol [10].

When it comes to software the Teensy is software-compatible with Arduino. This means it primarily uses the Arduino IDE, libraries compatible with Arduino boards and the coding style of Arduino. The usual workflow for Teensy is to first download and install the classic Arduino IDE (which also includes the basic libraries) and then install a special package called "Teensyduino" [11] which among other things contains dependencies necessary for compiling code for the Teensy device and also the Teensy loader application.

1.2 The OnlyKey

The OnlyKey is a personal authentication, encryption/decryption and message signing device which is based on the Teensy. It is developed by CryptoTrust[12] and currently manufactured by Black Vault Labs Llc[13].

The main author of the OnlyKey and its firmware and also the leader of CryptoTrust is Tim Steiner[14][3].

The main websites pertaining to the OnlyKey device are[4]:

- https://onlykey.io/ - The OnlyKey homepage - serves as the product store and houses basic user information as well as an FAQ, contacts, etc.

[3]https://twitter.com/cr7pt0 - Tim Steiner at Twitter

[4]https://www.instagram.com/onlykey.io/ - OnlyKey even has an Instagram account

- https://crp.to/ - the CryptoTrust homepage, we are especially interested in the subdomain, which is the home of the full OnlyKey documentation.
- https://github.com/trustcrypto - CryptoTrust's GitHub page, home of the open-source OnlyKey code.

1.2.1 OnlyKey basic capabilities and usage

One of the two primary goals of the device is to serve as a hardware password manager. When the OK is plugged into a USB port of a supported target device it gets powered on and waits for the user to input their PIN on the provided buttons numbered 1-6. After a successful PIN entry the device is unlocked and can input usernames, passwords and more. This information is stored in numbered "slots" which are activated by pressing the corresponding button of an unlocked OK. Every button actually represents two slots - one gets activated by short press and the other by long press. This way up to 12 slots can be saved. Since Firmware version Beta 7, the user can set up two separate profiles, each using a different access PIN and each being able to store 12 accounts so up to 24 accounts total. The OK actually mimics a Human Interface Device (HID for short), specifically a USB keyboard[15] and types out the password and other info in much the same manner, as a user with a physical keyboard would. This has the advantage of being very platform independent and not needing any special software for its use. The user can also set up a third PIN which serves as a self-destruction PIN - after its entry all user data get wiped from the device. The same thing also happens if the user enters 10 incorrect PINs. Another important purpose of the device is to serve as a universal second factor.

Other usages of the device include using the device to encrypt/decrypt and/or sign messages and/or files using gpg format. It is also possible to sign git commits and to set up passswordless SSH authentication using the provided Agent apps.

1.2.2 OnlyKey variants

As of the time of writing there are four main variants of the OnlyKey device - the OnlyKey Original (OKO), the OnlyKey Color (OKC), the OnlyKey Color developer edition and the OnlyKey Duo (OKD - formerly during development also known as the OnlyKey Go). There are several substantial differences between these versions:

The OnlyKey Original (for about two months after the first announcement known as OpenKey[16]) (pictured on 1.3 and 1.4) was crowd-sourced on Kickstarter and released in 2016[16]. This variant is discontinued and no longer being sold. It is based on Teensy 3.2. It includes Teensy's HalfKay bootloader located on a separate chip[17]. With normal Teensy this bootloader gets triggered by pressing the on-device button. On the OKO I can simulate the same

trigger by bridging the two small contact points in two corners of the device using a small metal wire (1.5)[18]. It has a single-colored orange LED. The latest officially supported OnlyKey Firmware is Beta 7[18], which is also the first version of the OK firmware that was also released in a digitally signed form. I had this device physically available to me during the work on this thesis.

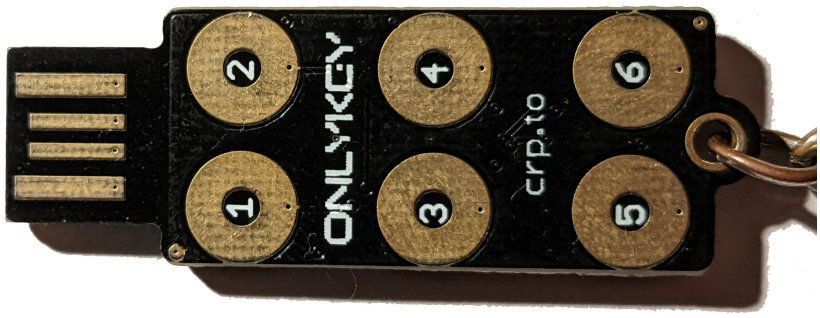

Figure 1.3: The OnlyKey Original from the front

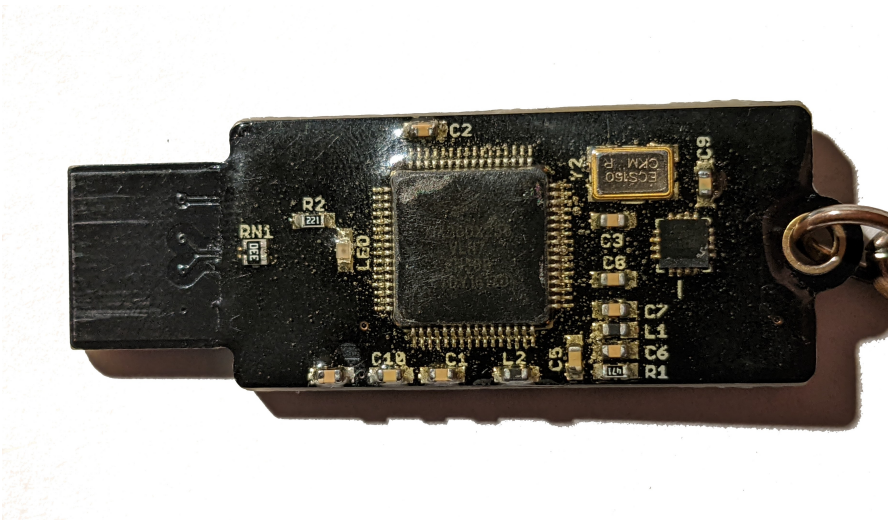

Figure 1.4: The OnlyKey Original from the back

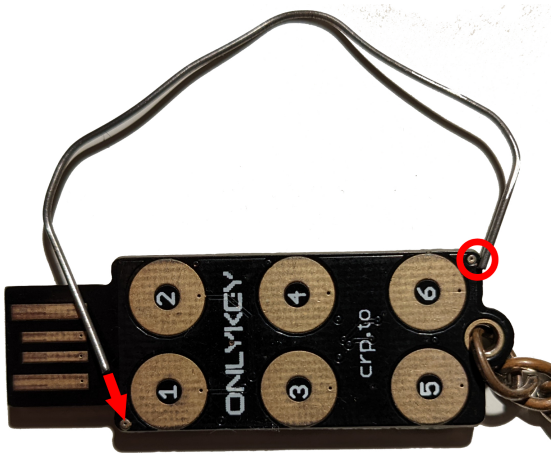

Figure 1.5: The contact points which are used to jump to bootloader

The OnlyKey Color (pictured on 1.6 and 1.7) was announced and released in 2017[16]. This device is still currently being sold (at the time of writing). It is also based on Teensy 3.2 and is visually very similar to the Original. The most obvious difference and also the reason for its name is that the built-in LED is now capable of emitting different colors and brightness levels.

However from a security standpoint a much more important difference is the absence of the separate hardware HalfKay (HK) bootloader[17]. Instead the device contains an open-source bootloader by Black Vault Labs Llc. (the OnlyKey manufacturer). I will refer to this bootloader as the BVL bootloader or BVLB, for short. The BVL bootloader, unlike its HalfKay counterpart, gets executed on the main chip, just as the firmware does. The BVLB binary is also stored in the same flash memory as well - it begins on program address 0 and therefore gets executed every time the device is powered on. It has functionality designed to recognize installed OK FW, to which it jumps and passes control, provided it passes an integrity check - discussed in more detail in2.3.2. It is also capable of re-installing/updating the OK firmware, while verifying it has a valid cryptographic signature by CryptoTrust - again will be discussed in more detail in2.3.2.

There is also a special developer edition which isn't sold through the CryptoTrust website (anymore), but can apparently be acquired by communicating directly with the OnlyKey developers[17]. This version again features the HalfKay bootloader and would therefore allow for the installation of unsigned or custom firmware binaries (for example for testing purposes). There also seems to be some indication that this was originally the device sold under the OK Color moniker, before being replaced by the one without the HK bootloader - my reasoning behind this assumption is explained in 2.3.1. Unfortunately I only had access to the commercially available OKC, but I partially

made up for this issue by simulating the OKC development version using the OKO (more on this topic in 2.3.3.

Figure 1.6: The OnlyKey Color with the supplied cover removed from the front

The OnlyKey Duo (picture on 1.8) was also crowd-sourced, both on Kickstarter[19] and also on Indiegogo[5]. Shipping of the pre-ordered units started in March 2022. I do not have access to this device and this thesis doesn't spend much time discussing it.

There are however several important differences of this modernized version. The device only actually has two buttons instead of six. Button three is emulated by pressing in the middle and therefore pressing both button one and two at the same time. Numbers four through six are simulated by long-pressing numbers one through three respectively. Since with older OK devices

[5]https://www.indiegogo.com/projects/onlykey-duo-the-best-protection-for-your-devices/

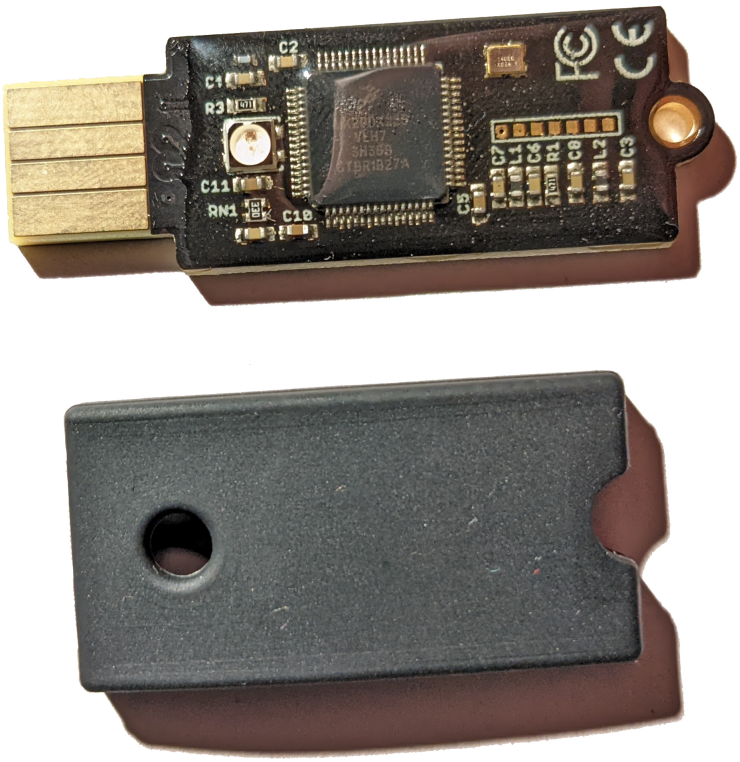

Figure 1.7: The OnlyKey Color with the supplied cover removed from the back

long presses were used for a second slot for each button it would seem now I can only store 6 slots per profile. This is true, however to make up for it, the device has 4 profiles instead of 2 and so the total number of slots remains unchanged at 24. The way to access the separate profiles has changed however - instead of using 4 different login PINs, the user only has one PIN and after successfully entering it, can switch between different profiles by holding button 2 for 5+ seconds[19]. Also the device has a much smaller footprint and not only features USB-A on board but also includes an USB-A to USB-C adapter which at the same time serves as a plastic protective cover[19].

1.2.3 OnlyKey security features

The OnlyKey online documentation claims many security features and properties[21]. I will go through the claimed properties one by one and comment on all of them, while analyzing some of them in detail. The paragraphs in quotation marks are direct citations from the documentation[21].

First is the Security Features Overview:

Figure 1.8: Official 3D render of the OnlyKey Duo available from the OnlyKey product page [20]

- ”Firmware verification - The bootloader verifies the firmware signature on the OnlyKey. The firmware is only loaded onto the OnlyKey if the firmware is correctly signed by CryptoTrust.” - This is done by the BVL bootloader and only makes sense with OK types which do not feature the Teensy HK bootloader. The HK bootloader is agnostic of any OK software and so doesn’t perform any such signature verification - therefore it can be used to load unsigned/custom firmware. The signature verification takes place during FW installation/updating, not during device bootup.
- ”Firmware integrity checking - The bootloader verifies the firmware hash each time the device is used. The device only starts the firmware if the firmware has not changed.” - This is also achieved using the BVL bootloader - on each bootup, the BVL computes a sha256 hash of the installed FW and compares it to a stored hash.
- ”Tamper resistant / chemical resistant hardware - The device is coated with a chemical resistant coating that is resistant to chemical removal.

Visible damage is done to the device by attempting to access coated electronics (Tamper evident).” - A simple visual inspection does confirm, that there is some kind of coating on the device. It seems durable, however any experiments attempting to test how resistant exactly it is, will not be explored in this thesis.

- ”Protected key operations - Encryption / decryption operations are only allowed after user authentication via PIN.” - This is true, I did confirm it by studying the FW source code - will be discussed in more details later.

- ”Read/write-protected secure flash - OnlyKey utilizes Kinetis flash security to securely lock all data residing on OnlyKey.” - This is a feature supported by the main Kinetis chip. It uses different values which the user program can set in the FSEC registr, and which then can allow or forbid certain actions regarding the internal memories - like not allowing them to be wiped, or read from[22].

- ”Offline secure processor - The data stored and processed on the OnlyKey is completely isolated from the connected computer. Data can only be written to the OnlyKey or wiped. Physical user touch is required to authorize authentication.” - This is true, I did confirm it by studying the FW source code - at no point does the host device directly read anything from the OK device. The device has to willingly send any communication data. For sign-in credentials to be written out and/or cryptographic operations to be performed, the user always has to at least touch a button on the device.

- ”Secure encrypted backup and restore - Backups are securely encrypted with a user’s passphrase (25+ characters) or a user’s PGP key.” - This is true, I did confirm it by studying the FW source code - it is also possible to choose a setting which disables future backup mode (PGP/-passphrase) and value (the actual key/phrase) changing. The backup function is discussed more in 2.1.2.4.

- ”True random number generation - To guarantee random keys the patent pending method of random number generation utilizes a combination of hardware entropy and user touch entropy. The user touch entropy is completely unpredictable random data that is affected by many variables such as the conductivity of the user’s skin, how long they press button, how long they delay between button presses, temperature and humidity.” - The source code does contain the described functionality. Whether the RNG could truly be designated a True RNG should be explored using both RNG statistical test suites and extensive testing - this is beyond the scope of this thesis. Note that I do believe the RNG output is random

enough for what its purpose in the OK is - which is primarily generating nonce numbers. More about the RNG generator can be read in the OK docs[21].

After the overview the page talks about Hardware Security, which I did not explore. It also links to several papers/studies in which other similar HW solutions were broken with different HW based attacks.

1.3 OnlyKey alternatives

Other solutions using a hardware token include the YubiKey, SoloKeys, Google Titan, etc. Most of them do not match the OK in the combined functionality and security - it is usually not possible to enter a numerical PIN on them, instead they often rely only a on a single physical touch. Other ones also might not support all the features that OK does.

Chapter 2

OnlyKey Software

All the official OnlyKey software is open-source. The only exception is the HalfKay bootloader in the OKO, however that is not provided by TrustCrypto, but is present on the Teensy by default.

I will now establish several terms for the purposes of this thesis:

- A blank OK - is an OnlyKey device without any version of the OK firmware installed.
- An uninitialized OK - is an OnlyKey device with a fresh installation of any OK firmware version - no PINs or user credentials are set up.
- An initialized OK - is an OnlyKey device which has had its FW "initialized" - this means, that at least one profile PIN has been created - and the corresponding profile can therefore be logged into - but doesn't necessarily mean any user credentials or other data were saved on the device.

2.1 OnlyKey App

The OnlyKey App (The App) is an open-source cross-platform javascript based application, with the source code located on the Trustcrypto GitHub[23]. The already built application is also provided there to download from the Releases page. For most people it is the most straight-forward way to setup and use the OnlyKey device. It is capable of setting up a blank OnlyKey, a fresh uninitialized OK and further configuring a previously initialized one, as well as re-installing the device's firmware. The app is not necessary for the actual usage - meaning typing in saved credentials, etc. - of an already initialized device.

When I started writing this thesis the most up-to-date version of the OnlyKey app was 5.3.3, which is therefore the version I will be using for my

testing purposes. Since then at least one newer version was released, version changelog is also available on the GitHub releases page[24].

2.1.1 OnlyKey first setup

Before describing the OK setup and usage with the help of the OK App we need a few more bits of information about the OK usage and firmware. From the perspective of user login there are basically four main states of an initialized OK:

- A locked OK waiting for PIN entry to perform normal user login
- An unlocked OK in normal usage mode
- A locked OK waiting for PIN entry to switch into "config mode"
- An unlocked OK in config mode

Config mode exists since FW version Beta 4[25] and is provided as an extra security measure. Config mode has to be enabled, for some features to become usable - usually changing values of important preferences. For PIN checking, it uses the same code as the normal login and so is equally as secure. The added benefit lies in the situation of an attacker stumbling onto an unlocked OK device, connected to a computer - they will of course be able to extract all the credentials saved on the unlocked profile, however the features protected by config mode will still be unavailable to them - since a PIN entry is required again. If PD mode second profile is enabled and the user attempts to enter config mode from the first STD profile, but when prompted enters the second profile PIN, the device locks itself. If the user enters the SD PIN while being prompted for a PIN to enter config mode, the device does still perform the factory reset.

There are actually two different editions of the OK FW available, depending on the user's needs. One is called the STD (Standard) edition (formerly also the US edition) and the other one is the IN_TRVL (in travel) edition.

The IN_TRVL edition has limited capability[26]. Only one profile can be used (allowing the storage of up to 12 user accounts) and most importantly, the user data isn't encrypted. The reason for its existence is that some legal jurisdictions can limit or outright ban the usage of encryption and it might therefore be illegal to use OK with the STD edition firmware within such countries[6] (hence the name of the IN_TRVL edition). This limited edition also doesn't support a config mode and so quite a few other features become inaccessible - I will always mention this, when I discuss the features in question.

[6] https://www.gp-digital.org/world-map-of-encryption/ - a world map showing some of the cryptography-related jurisdictional facts.

When the STD edition FW is used, all features are available. A second profile is available to be used, and it is possible to choose between a ”standard” (STD mode) profile, or a ”plausible deniability” (PD mode) profile or no second profile at all. The PD mode makes it so that when the second profile is logged-into, it behaves as if the IN_TRVL edition firmware was installed. In such a situation the first standard profile should be undetectable and the user can therefore ”plausibly deny” the usage of a device featuring cryptography, which might be restricted. Instead he can claim to be using the IN_TRVL edition.

We will now go through the OK setup using the newly installed OK app. After running it, the App's welcome screen is displayed as can be seen in figure 2.1. A dialog box indicating a newer version of the app can also appear (2.2), since the app features the possibility of in-app auto-updating.

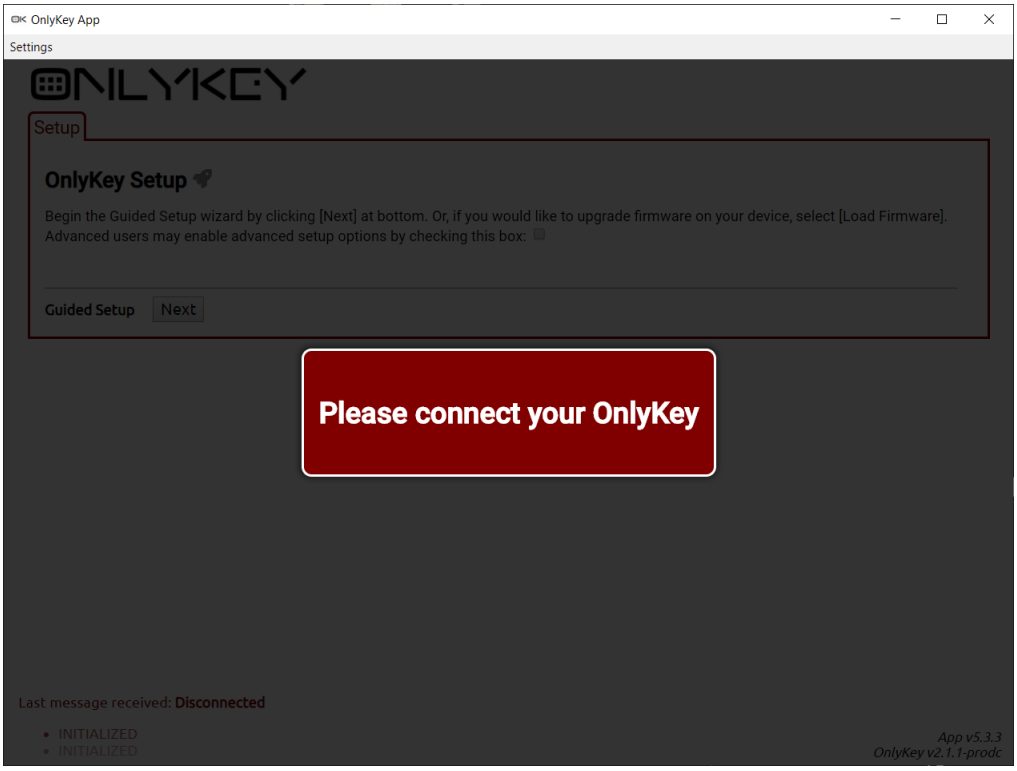

Figure 2.1: The OnlyKey App welcome screen

Take note of the lower left corner of the App window - it displays system messages sent by the OnlyKey device using the ”Raw HID” communication[10]. The App sends requests to the device and then reacts to the receivd reply.

When an uninitialized OnlyKey is plugged in, the app guides the user through the initialization process. On the first screen we can either choose to

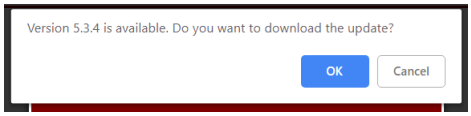

Figure 2.2: The dialog box informing about new update

re-install the OK FW, or to initialize the currently installed FW. Before initializing we can also tick a checkbox, which will trigger "advanced setup". On the second screen we are informed about how to setup the first PIN - the user has to tick a checkbox, declaring their understanding that user information is not recoverable, if the PIN is lost. After ticking the checkbox, the desired PIN is entered on the OK device's keypad. Then we click next, enter the PIN again (to prevent typos) and click next one more time.

After this we are presented with the option to setup the second PIN. The process is nearly the same, with a few key differences. Unlike with the first profile, we can choose to skip the step altogether, if we do not require a second profile (it can be activated later). Secondly if we chose the option to use the advanced setup, we have a choice between the already mentioned standard (STD) second profile or plausible deniability mode (PD) second profile.

Note, that if the IN_TRVL edition is used, a second profile isn't available - however the app doesn't reflect this and acts as if the user can setup the second profile normally. It doesn't really matter whether we choose simple setup, or advanced setup and click on STD/PD mode for the second profile - if the user does attempt to setup the second profile the first PIN actually just gets overwritten by the second one - however the app doesn't provide any indication of this happening.

After that the process repeats one more time and the user can setup (or skip) the self-destruct PIN, which can be used to quickly delete all user data.

The second-to-last step is backup settings. From the users perspective, The OK supports two methods of backup encryption - with a backup passphrase and with a private OpenPGP key. If we chose the advanced setup, we can now choose which of these methods to use and also whether we want to be able to change the method/key/phrase later. If we use the simple setup, only passphrase is available (however all these settings can still be changed later).

The last step is offering us to restore data from a previously made OnlyKey secure backup. This option can be safely skipped, when setting up a new device.

In the case of IN_TRVL edition FW, the backup functionality is unavailable, because it requires cryptography - however the app once more doesn't reflect this. If we attempt to proceed with the backup-settings-setup the app just stays on the same screen. If we chose advanced setup, the setting of the (in)ability to change backup method later actually gets reported in the App as successful, even though this is clearly not relevant in the IN_TRVL edition.

The way to end the setup with the IN_TRVL edition is therefore either to click the ”exit” button, or to unplug and re-plug the OK[26].

It should be noted that only the first step - setting up the primary profile - is mandatory, all the rest is optional, and also possible to setup later (with the exception of the PD second profile - that feature is only available if activated during setup). Also I want to emphasize, that if the IN_TRVL edition is used, the app doesn't change in any way - the setup process looks identical, even though some of the features can't actually be activated - this doesn't seem very user-friendly. Also there is no difference between using the simple and advanced setup in that case, since all the extra features provided by advanced setup are cryptography dependent and unavailable (even though the App still shows them).

2.1.2 OnlyKey usage

After the device has been initialized, the OK app prompts the user to enter their PIN on the OK as seen in 2.3. At this point the user can enter any of the set up PINs, including the self-destruct PIN.

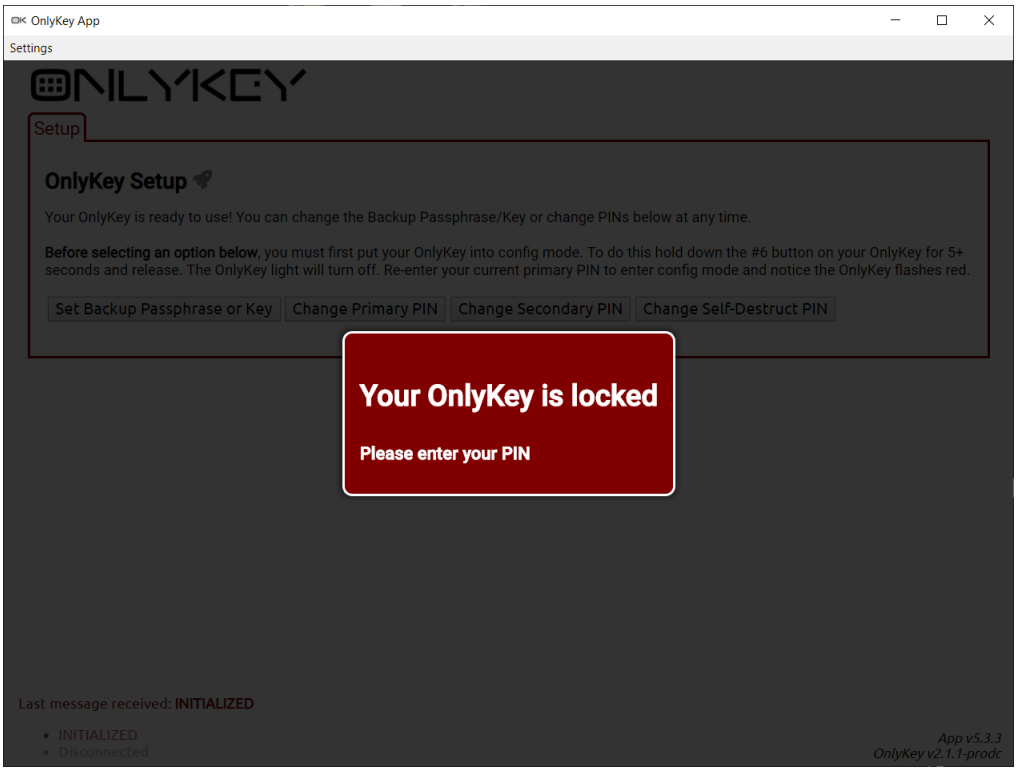

Figure 2.3: User PIN requested

When one of the correct (non-self-destruct) PINs is entered the app first

shows the ”Slots” tab.

2.1.2.1 Slots

The Slots tab can be seen in (2.4). It lists the ”Labels” assigned to each slot. Labels are just user-provided names for the different saved account credentials. A slot number indicates which button it corresponds to, and the letters ”a” and ”b” differentiate between short-press and long-press respectively.

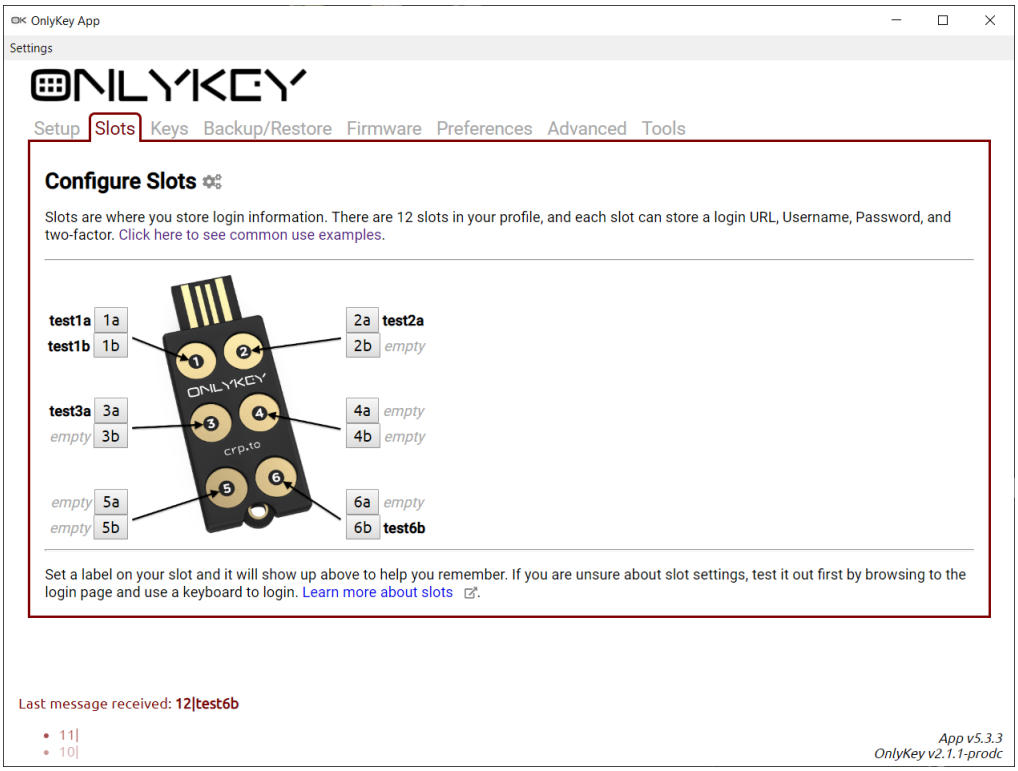

Figure 2.4: Slots tab

Clicking any of the slots brings up the slot configuration window (2.5), allowing the user to assign the required values.

The available values are:

- Label - as was already mentioned, the labels don't really serve any functional purpose and are their for the user's convenience. It is worth noting that even in a STD profile, this value does not get encrypted and therefore definitely shouldn't contain sensitive information.
- URL - if the slot is used to save credentials for a specific website this value can be used to also input the websites URL into the browser, so the user doesn't even have to navigate to the website manually (the enter

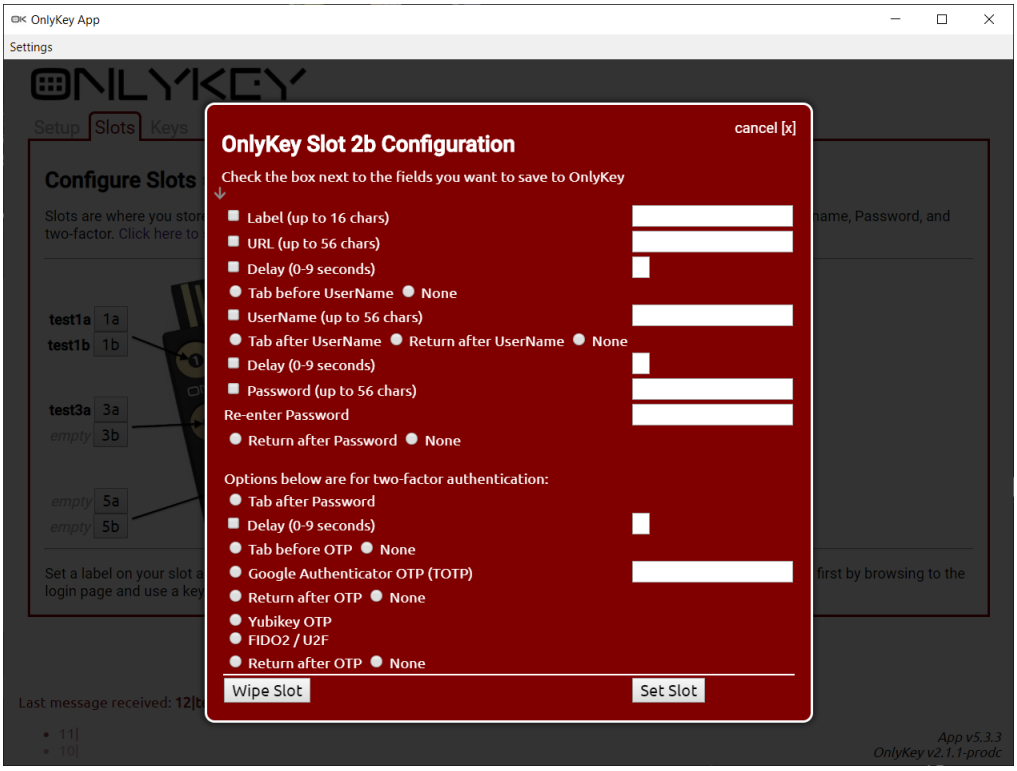

Figure 2.5: Slot configuration window

key is always automatically pressed by the app after typing out the URL value).

- Delay - this value can provide a 0-9 second delay between typing out the URL and the username value. This is almost always useful, since even with really fast internet connection the loading of the website will not be instantaneous.
- Before username character - after the delay either nothing or TAB can be pressed - this is useful in cases where the web element of the target webpage selected by default is not the username input field.
- Username value - self explanatory, contains the login username/e-mail or any such user identification the target webpage uses.
- Before password character - after the username input the app can press either nothing, the enter key or the TAB key to move to a password input field.
- Delay 2 - again its possible to add a 0-9 second delay, before the password is written out. This is useful for example if the webpage works the

following way: the user provides their username, presses enter, and then has to wait for additional content to be loaded before entering their password.

- Password and re-enter password - traditionally, the password has to be provided two times, so that the the user is protected against storing a string containing a typo.
- After password character - we can choose to let the OK press enter to confirm the login.

For the classic 1-factor password authentication the already listed options are sufficient. The following options are available to enable the use of 2-factor authentication, which supports several different protocols:

- TAB presses and delays can again be setup between password input and the input of the second factor.
- Time-based one-time password (TOTP - sometimes referred to as Google's TOTP) - this is a time-based second factor which normally uses a smartphone application called Google Authenticator. In this case the OnlyKey replaces this app and its functionality. An official Arduino library called simply "TOTP library" is used for this functionality[27]. In order to use it the user needs to copy the secret code from their Google account (or another account that supports this type of authentication) and enter it in the app. The mechanism of generating the one time passwords is hashing together a pre-shared secret value and a current timestamp. Since the OK is a passive device, which needs to be powered by the host device, it cannot hold its internal clock when not in use. This is therefore one of the features that requires OK software to work properly - either the OK App installed on the host machine, or the OK Webapp (only needs a compatible web browser) have to be opened. They automatically detect the connected unlocked OK device and notify it of the current time. Until the OK is diconnected or locked it will be able to generate the one time codes.
- YubiKey's OTP - This is an OTP second factor authentication method developed by Yubico, the company that makes the YubiKey[28].
- U2F/FIDO2 also known as universal second factor[29].
- After 2nd factor character - we can again choose to let the OK press enter to confirm the login.

Except for Google's TOTP the second factors do not work while using the IN_TRVL edition or PD mode.

2.1.2.2 Setup

The window can be seen in 2.6. From this screen we can change all the PINs and also the backup passphrase - the process is always pretty much identical to the first-time setup. All of these action require the switch to config mode, which makes them inaccessible from both the IN_TRVL edition and PD mode second profile. If the IN_TRVL edition is used, this implies that the SD PIN has to be setup during the first-time setup, if the user wishes to use it, otherwise it can't be setup later. In such a situation if the user wants to perform a factory reset, they will have to input a wrong PIN 10 times. The IN_TRVL edition however, also doesn't support the secure backup function - therefore before performing the reset, the user will have to manually let the device type out all the credentials in clear text and back them up manually.

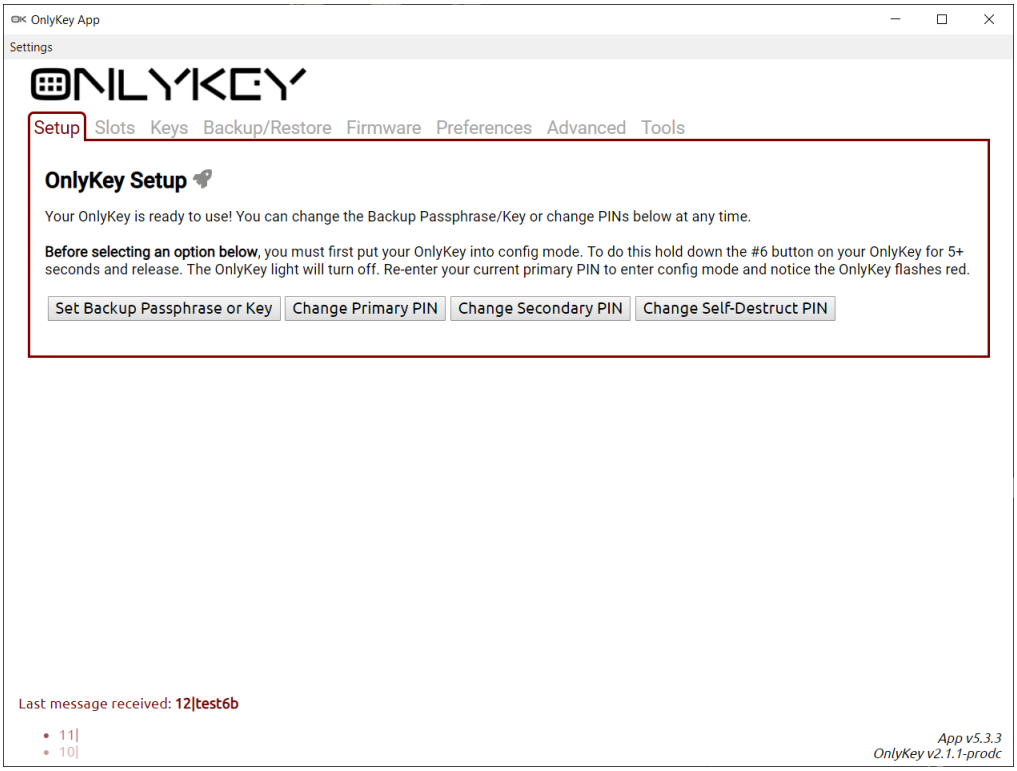

Figure 2.6: Setup tab

2.1.2.3 Keys

The window can be seen in 2.7. From this screen we can load RSA or ECC keys onto the OK, to be used for message/file encryption/decryption/signing. We can also wipe the already saved keys. The set as backup key will change the secure backup mode from using a passphrase to using the provided OpenPGP

key. All key loading/wiping requires config mode to be enabled first, and therefore these options are unavailable in the IN_TRVL edition and in PD mode.

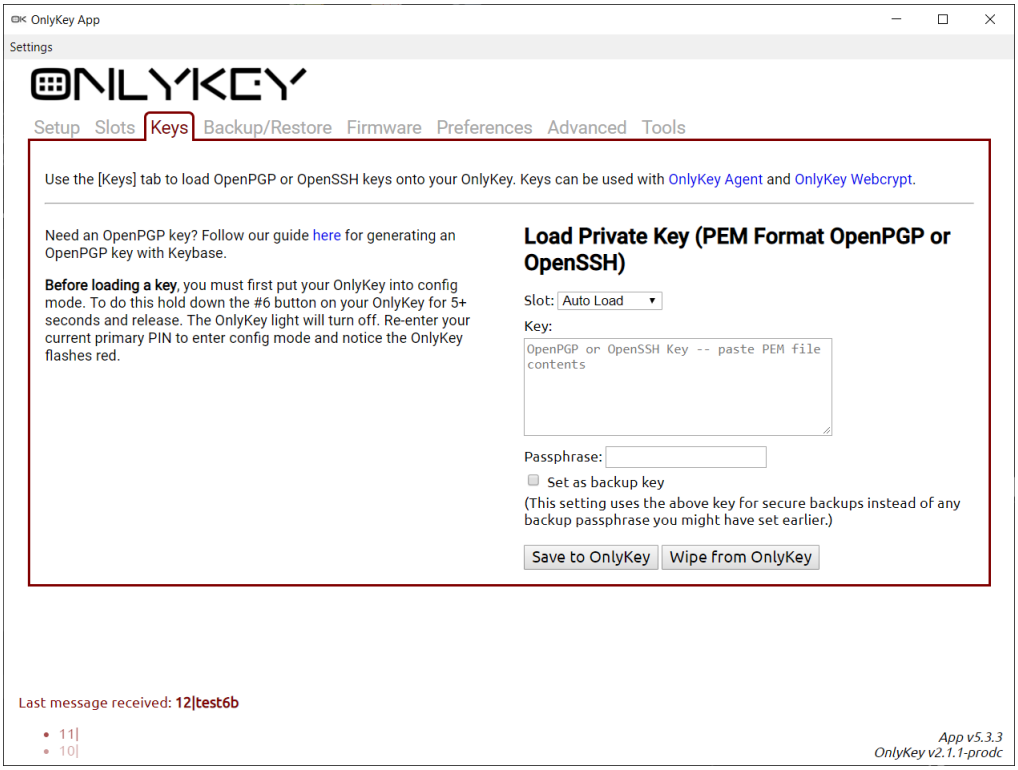

Figure 2.7: Keys tab

2.1.2.4 Backup/Restore

The window can be seen in 2.8. It has instructions on how to have the device write out a backup - it is not necessary to use the supplied text field, any text editor will do. It is also possible to restore a previously made backup here, from a saved text file - for this the device has to be switched into config mode. If IN_TRVL edition or PD mode is used, none of the options work, since the backup functionality isn't available.

2.1.2.5 Firmware

The window can be seen in 2.9, it gives us the option to re-install or update the OnlyKey FW. It works in conjunction with the BVL bootloader and will only load a FW binary that has been digitally signed by CryptoTrust. Before using this function the device has to be switched into config mode and therefore

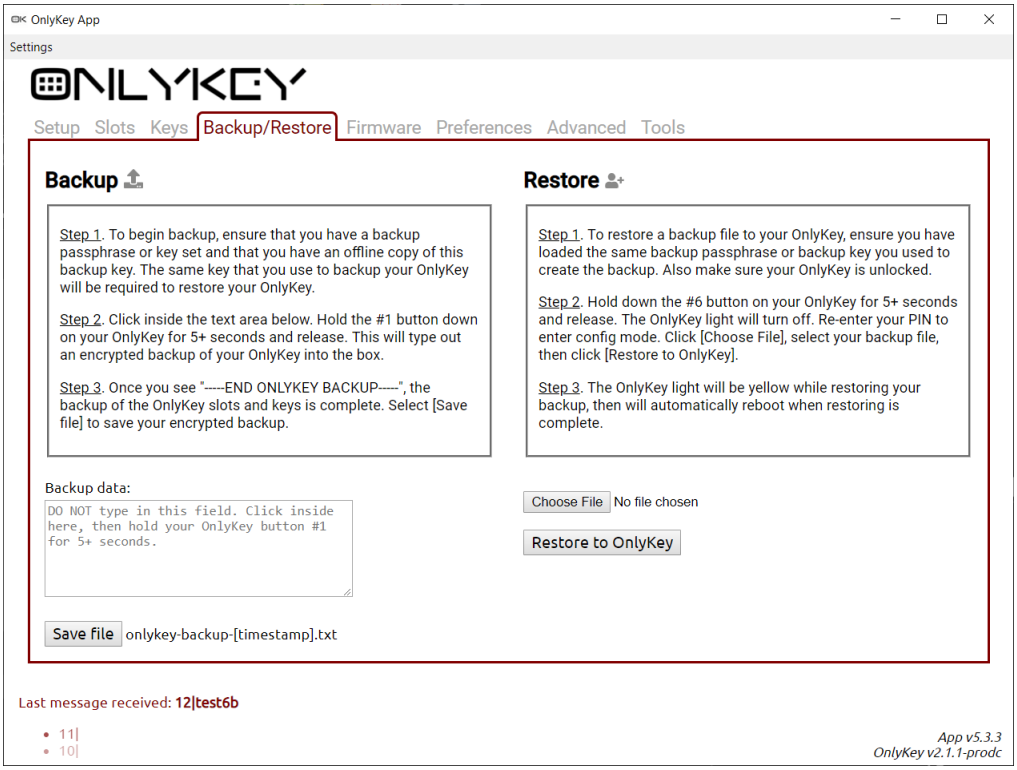

Figure 2.8: Backup/Restore tab

this cannot be used with IN_TRVL edition or PD mode. However, the config mode is only necessary here to protect the OK from being re-installed by a potentially unsafe PC. If the legitimate user has physical access though, it is possible to bridge the two contacts in the corners of teh device for 3 seconds, and the device will jump to BVL and wait for FW install instructions from the App (the App reacts accordingly automatically). This can be done from any screen, even with a locked OK, so PIN knowledge is unnecessary for it.

2.1.2.6 Preferences

The window can be seen in 2.10, and understandably it allows the user to change different options. It is important to note, that the displayed settings in the preferences tab do not actually reflect the preferences already setup on the OK device, they only serve as a way to set the options authoritatively, not to display their current value - in other words, the OK device never actually lets the App know what the option settings currently are, it just lets it change them. Now let us look at each of these preferences:

- Keyboard Type Speed - allows the user to change the typing speed of the OK. This can come in handy, since the default value of 4 is relatively

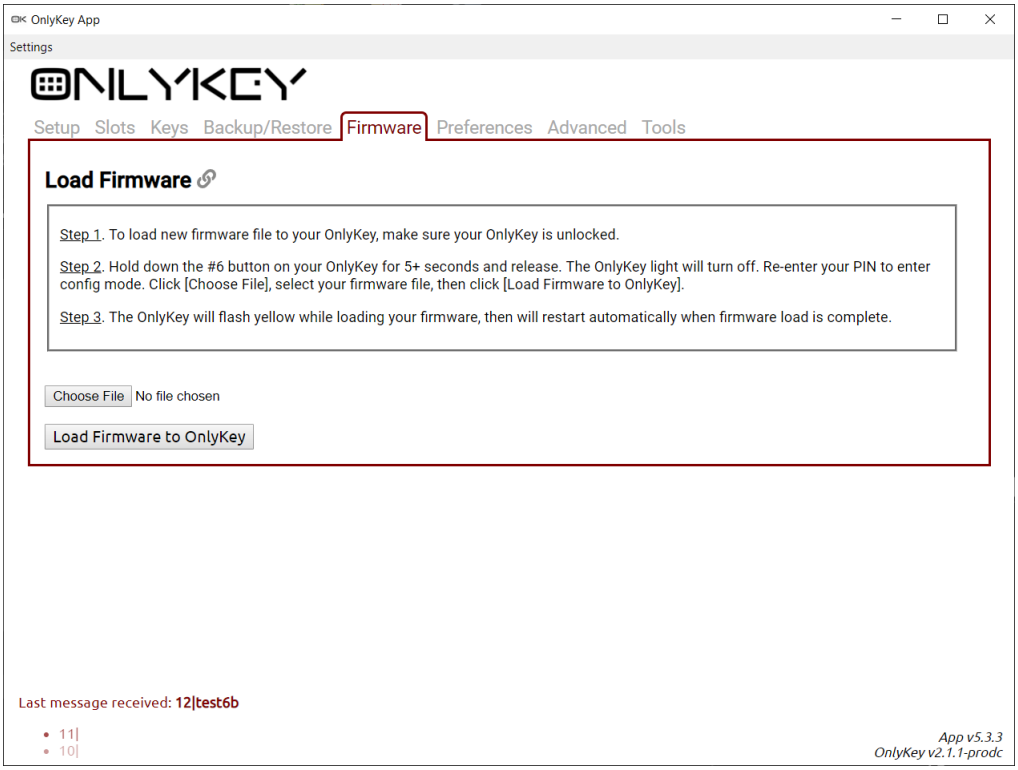

Figure 2.9: Firmware tab

slow, but the higher values might cause problems on some platforms or applications, which are not capable of interpreting keystrokes so fast. Especially when letting the device typeout a backup, it can be useful to set it to a higher speed, since the text is relatively long. Doesn't require config mode.

- Keyboard Layout - this needs to be set to the layout which the PC/host device uses, so that the OK can send the correct corresponding keystroke codes. Doesn't require config mode.

- Indicator Light (LED) Brightness - allows for changing the brightness level of the colorful LED on the OK Color (doesn't work with the monochrome LED of the OK Original). Doesn't require config mode.

- Sysadmin Mode - if sysadmin is enabled the OK can be used to a much greater effect through the slot values - it can press pretty much any keyboard keys and their combinations, therefore allowing things like launching scripts, etc., but as the documentation points out: "Once you enable this feature you will no longer be able to set slot values without first putting OnlyKey into config mode. This adds an extra layer of

security for system administrators."[30]. Actually not only slot setting is affected, also all the other preferences settings, which previously didn't require config mode to be changed, now do require it. Requires config mode to be turned on.

- HMAC User Input Mode - the OK documentation says: "OnlyKey supports HMAC challenge-response. By default, user input (button press) is required on OnlyKey to perform HMAC operation. For some use cases such as full-disk encryption no button press may be preferred. With "Button Press Not Required", HMAC challenge-response operations may be performed without user interaction."[30] - requires config mode to be changed.
- Wipe Mode - if turned on then factory-resetting the device (either through 10 failed PINs or through the SD PIN) also wipes the installed OK FW, not just the user data. Can only be turned on, and then cannot be disabled. Requires config mode to be turned on.
- Inactivity Lockout Timer - the OK will lock itself automatically after this time in minutes has elapsed without any user interaction. Seto to 0 to disable. Doesn't require config mode.
- Lock Button - allows the user to designate one of the OK buttons as a Lock button. When pressed the OK locks itself instantly and also attempts to lock the host machine by sending keypresses corresponding to the most common keyboard shortcuts used to lock different operating systems. Doesn't require config mode.
- Derived Key User Input Mode/Stored Key User Input Mode - these options are very similar. OnlyKey supports both automatic generation of keys that may be used for SSH and PGP/GPG with the OnlyKey Agent and import of existing OpenPGP keys using the OnlyKey app. The documentation says "The default setting is "Challenge Code Required" which requires a 3 digit challenge code to be entered on OnlyKey to perform SSH or PGP/GPG operation. This is great for security but for some users a more convenient approach may be preferred. With "Button Press Required", a physical press on any key is all that is required to perform the operation."[30], which applies to both of these options. Both options require config mode to be changed.
- Backup Key mode - similarly to Full wipe, once set this cannot be disabled. It prevents the user from changing the backup key mode (PGP vs passphrase) as well as the actual value. Requires config mode to be turned on.

All the options that require config mode are only available in STD profiles, except for Full Wipe, which can be enabled from an IN_TRVL edition or PD

mode profile, without the use of config mode. The reason behind this will be discussed in 3.1.2.

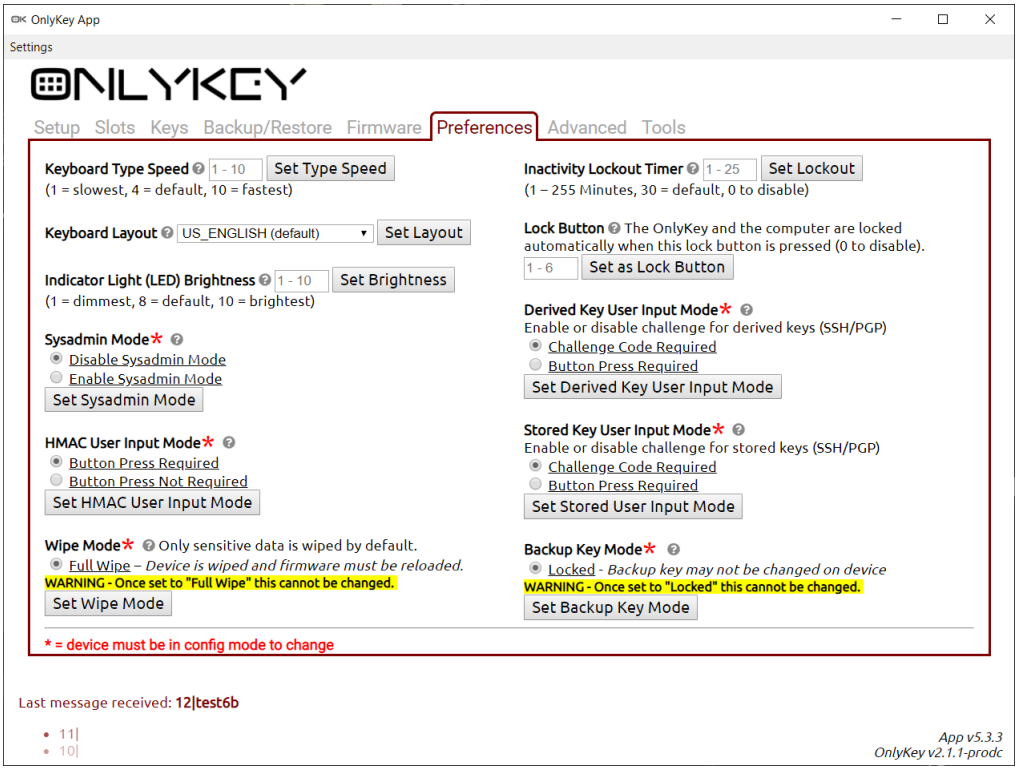

Figure 2.10: Preferences tab

2.1.2.7 Advanced

The window can be seen in 2.11. It is basically an advanced version of the Keys tab and also allows for key loading, however it provides more options to choose from for the user. Private key loading requires config mode. Yubikey Security info is a funtion provided for compatibility with the Yubikey device and its loading does not require config mode - however neither of the options are available in the IN_TRVL edition and PD mode, since they do require cryptography.

2.1.2.8 Tools

The window can be seen in 2.12. This tab actually doesn't actively do anything on its own, it just links to the OnlyKey WebApp, which allows for message and file encryption as well as digital signing. It also links to the OK documentation explaining how to use the OK GPG and SSH Agent. None of these functions support IN_TRVL edition and PD mode, since they require cryptography.

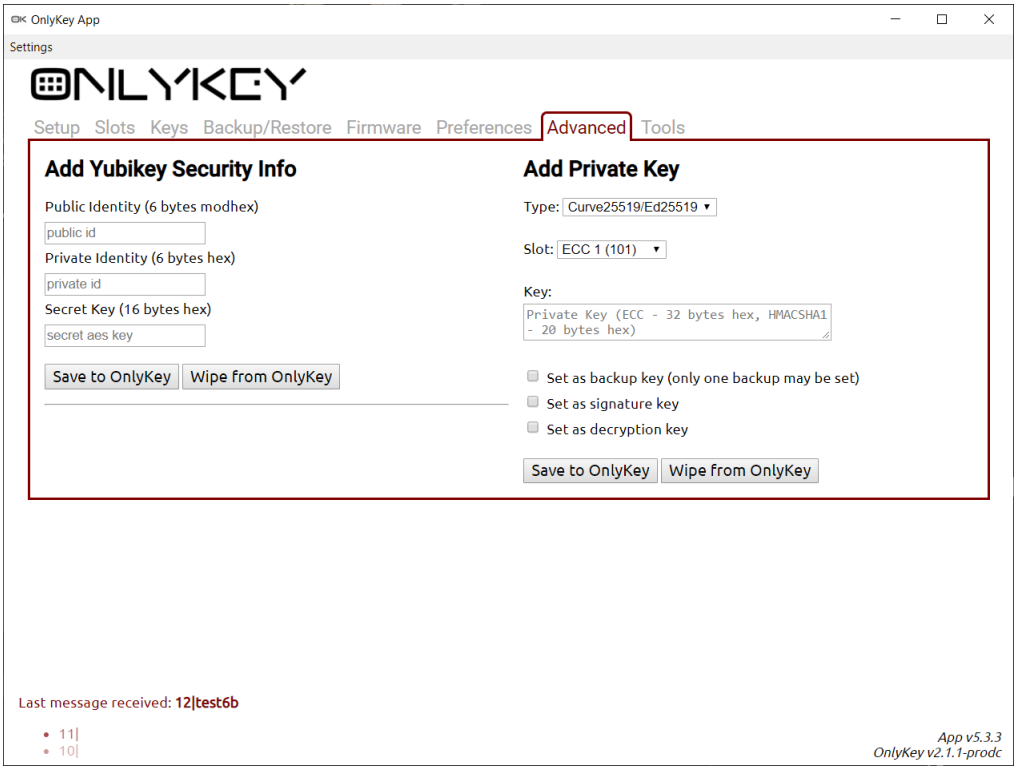

Figure 2.11: Advanced tab

2.2 OnlyKey App alternatives and other software

Apart from using the OnlyKey App there is also the possibility of using a ”Quick setup”, which doesn't require any specialize software, only a generic text input field, such as a text editor. However this way of setting up the device is very limited. The way to use it is described in the OK docs, but it is recommended to use the App[30]. Another option is to use the OnlyKey Python command line interface, which is also hosted on the CryptoTrust GitHub[31] and allows for setting up the OK.

Other official OK software includes the OK WebApp, which can be used for PGP-like message/file encryption and signing - it is available under this webaddress - https://apps.crp.to/.

Lastly there is the OK SSH/GPG agent, also available from the official GitHub page[32]. The documentation says: ”OnlyKey Agent is a hardware-based SSH and GPG agent that allows offline cold storage of your SSH and OpenPGP keys. Instead of keeping keys on a computer, OnlyKey generates and securely stores your keys off of the computer and you can still easily use SSH and GPG”[33]. Unfortunately this app doesn't work very well with the Windows OS yet. It works well on Ubuntu and I would presume also on other

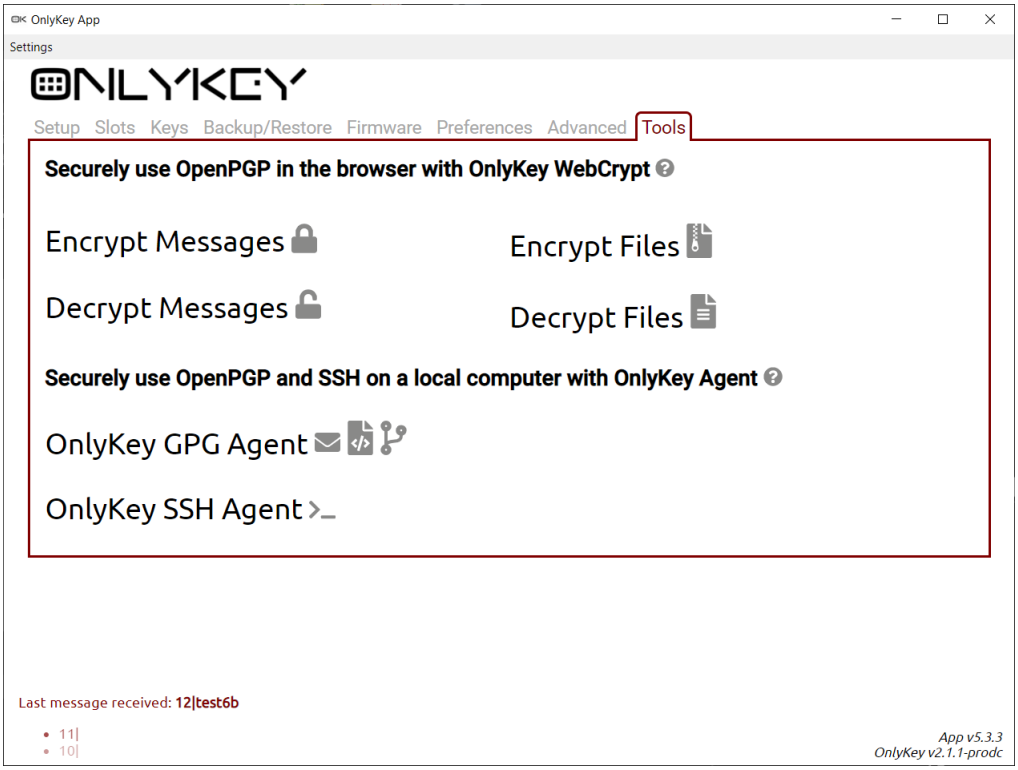

Figure 2.12: Tools tab

Linux distributions and Unix-like systems.

2.3 OnlyKey Firmware and bootloader

In this section we will discuss the OnlyKey Firmware in detail, together with the BVL bootloader, which is nowdays pretty much always paired with it.

2.3.1 Notable FW versions

- OnlyKey Firmware 2.1.1 - this is the firmware version this thesis is most focused on, since it has been the latest released version at the time I started writing it. Since then at least one newer version was released.

- OnlyKey Firmware Beta 3 - in this version the ”full wipe” was introduced. The corresponding changelog has a very relevant description, of why this is useful in case of using the seoncd profile in plausible deniability mode: ”If you are using the plausible deniability feature there is one scenario where an adversary may be able to determine that you were using the plausible deniability feature. This is possible if the adversary

enters 10 incorrect PINs causing your OnlyKey to wipe all data and then they go to reconfigure the PINs and see that they can set both a regular PIN and a PD PIN. Since only the U.S. version firmware allows setting both PINs the adversary would know that your OnlyKey is running the U.S. version firmware. At this point the device is wiped the adversary would not have access to any sensitive information but the adversary would know that your device is capable of encryption which in some areas may be undesirable. To address this issue you can now set the wipe mode of your OnlyKey to Full Wipe. Given the same scenario with Full Wipe set when 10 incorrect PINs are entered the device will completely wipe all information including the firmware from your OnlyKey. No useful information would be available to an adversary concerning what firmware you were running and in order to use the device new firmware must be loaded."[25]

- OnlyKey Firmware Beta 4 - is the first version which has a release directed towards the OK Color - however its not until Beta 7 that we get our first digitally signed release[25]. Together these two pieces of information imply, that originally, the OK Color also sold with the HK bootloader included (which would probably now be the development edition).

- OnlyKey Firmware Beta 7 - this is the oldest digitally signed version - this means, that it is the lowest version to which a device without the HK bootloader can be downgraded[25]. It is worth noting that Cryptotrust is now preparing versions of the FW - or rather of the BVL bootloader - which will feature downgrade prevention[34]. Since FW reinstallation done through the OK App doesn't erase the user data, this expands the potential attack surface - if there are any vulnerabilities in the Beta 7 version (or indeed any version released in between the Beta 7 and 2.1.1), they could be misused, even if they were fixed in the later versions. Also when using this version, I always needed to run the OK App with administrator rights in the Windows OS, for the App to recognize that the device is connected. I am not sure what causes this, but it is not a problem in newer versions - which is definitely good, as any application running with unnecessary rights poses a potential threat.

- OnlyKey Firmware Beta 8 - this version brings the ability to change the user PINs[25]. Closely coupled with this, is also a change in how the encryption scheme works - since the encryption scheme of this (and newer) version is not compatible with the older one, the recommended approach for update is as such: use the secure backup function on an OK running the older FW, erase the OK, update the FW and lastly restore the backup. The backup functionality is prepared in such a manner, that this change doesn't affect it.

2.3.2 Bootloaders and hid devices

As was already alluded to, there are two bootloaders which we need to know about. The closed-source HalfKay (HK) bootloader, which the OKO and development OKC editions use, and the open source Black Vault Labs (BVL) bootloader, which the newer devices (OKC and OKD) use.

The HK is provided by the Teensy authors and sits on a separate chip, stored in a separate memory. It gets triggered by putting the device into programming mode - bridging the two small contacts in the corners of the device, as was discussed in 1.2.2. When this bootloader is used, without using the BVL, the OK Firmware starts on address 0 of the main flash memory and therefore (if it is installed) is the first thing that gets executed, when the device is plugged in. If we want to re-install, or update the OK FW, we have to use an application called "Teensy Loader"[9]. This is a simple graphical utility provided by the Teensy authors. An unsigned, compiled FW can be loaded in this application, and after booting the OK into the HK bootloader it can install the new FW. It is important to note, that this process overwrites the whole flash and even though normally it shouldn't overwrite the (simulated) EEPROM[35] - which includes all the user data - in out case it does. This is done on purpose by the OK FW - it changes the FSEC value of the processor (part of the flash kinetis functionality) in such a way, that every upload will overwrite the EEPROM as well[36]. This is important, because otherwise somebody could create a custom firmware, which would printout the full EEPROM, without needing to enter the PIN - and even if that data would be encrypted, we do not want a potential attacker to be able to do that. Therefore, it is necessary to use the backup function, so that the data can be restored after the update.

When using the BVL, it is located in the same flash memory as the OK FW - this time the bootloader sits on address 0 and executed on every power-on. However, if a signed OK FW has been previously installed, the BVL will check its integrity, using a sha256 hashing function, and if it passes, will transfer the control to it using the `jumpToApplicationAt0x6060` function (everything that is the responsibility of the BVL bootloader is coded within its single source file - `BVL_Bootloader.ino`). If we want to re-install, or update the OK FW, we should use the official OnlyKey App, as was described in 2.1.2. There are actually three situations where the BVL triggers. One is - no FW is installed. Second one - if the user puts the OK into config mode and then uses the Firmware tab from the OK App. And lastly - if the same contacts in the corners of the device, which were used to trigger the HK, are bridged. However this time they have to be held bridged for 3 seconds, before the jump occurs. This is visible in the source code 2.13.

When updating an older OK device with a HK bootloader to newer FW versions, it is necessary to go through the unsigned Beta 7 version - this version actually contains the compiled BVL bootloader within - this can be easily

Figure 2.13: Code that periodically checks whether we should jump back to the BVL bootloader (from `OnlyKey.ino`)

```
//Check for bootloader trigger
//Trigger bootloader to load firmware by PTA4 low for 3 sec
if (!digitalRead(33)) {
 elapsedMillis waiting;
 int jumptobootloader = 0;
 while (waiting < 3000) {
  delay(100);
  jumptobootloader = jumptobootloader + digitalRead(33);
 }
 if (jumptobootloader == 0) {
  eeprom_write_byte(0x00, 1); //Go to bootloader
  //Firmware ready to load
  eeprom_write_byte((unsigned char *)0x01, 1);
  CPU_RESTART(); //Reboot
 }
}
```

confirmed with a hexeditor or even with a text editor, since the bootloader's version messages are visible in the binary. After it is installed it is possible to use the OK App to update to newer versions. This process is also desribed by the OK documentation[18].

The BVL bootloader is open-source as well and the source can be found on the OnlyKey GitHub[34].

The BVL contains a `startup_late_hook` which is a special function that gets called even sooner than the setup function[37]. It is from this function that the FW integrity check is called and if it succeeds the control is passed to the FW.

The GitHub releases page also offers a binary of a BVL Bootloader v2 (version 2), which should provide downgrade prevention[34]. However the commit with which the release is associated doesn't contain the corresponding source code - it still has v1 source code without the downgrade protection and with the v1 version output. The source code for the v2 version seems to be therefore unavailable (so far).

When exploring the different modes/bootloaders/etc, that the OK devices can use, I used a small Python script that I named `monitor_hid_devices.py` - it is available on the attached CD. It monitors all HID devices for new connections and disconnections and shows their vid number, pid number, manufacturer_string, product_string and interface_number - this helped me familiarize myself with how the OK operates, as I could compare the values to ones found on the Teensy website[9] and within the OK App source code - the collection

called `SUPPORTED_DEVICES` from the file `OnlyKeyComm.js`.

2.3.3 Building and debugging the FW

The OnlyKey firmware is written in both pure C and C++ and is actually a Teensy program, which uses the programming style of Arduino sketches. This means the programmer doesn't actually provide a classic `main` function and instead a `setup` and `loop` functions should be written. The `setup` function is automatically called once, every time the device is started. After that the `loop` function is called periodically in a loop by the device.

The OK authors decided to code the FW in a task-based programming style. This means that the top-level functions (meaning the ones programmed by OnlyKey creators, not the Arduino-supplied functions) are represented as tasks which are assigned to a task pool as necessary and some executor periodically goes through the pool and executes them. Specifically an Arduino library called Softtimer is used[38]. The library implements the Arduino `loop` function and inside it takes care of the user-supplied tasks. Therefore the user only creates the Arduino `setup` function and within it supplies at least one task for the library to execute from the `loop` function. Additional tasks can be added as well as removed on demand.

The complete firmware source code is made up of two parts, which are both available from Trustcrypto's GitHub. The first repository is called "OnlyKey-Firmware"[25] and the second one is "libraries"[39]. In the "OnlyKey-Firmware" repository we can also find the version changelog, available on the releases page.

The main repository - OnlyKey-Firmware - contains a directory called OnlyKey, which houses the main OnlyKey.ino file - the .ino extension indicates this is the file representing the "arduino sketch". Other files in this repository but outside the OnlyKey directory are responsible for keymapping for different keyboard layouts, usb constant definitions, etc., these should be placed in the Arduino installation subfolder `hardware/teensy/avr/cores/teensy3`, where they replace a few original Teensy files.

The second important repository making up the complete OK FW is the libraries repository. It contains libraries necessary for the FW compilation and function. There are both external resources and libraries made by the author of the OK.

No official instructions are given as to how the firmware should be built, but there are several helpful posts and discussions between users and the developers about this topic [40][17]. For my own convenience regarding the building and debugging of the FW I devised a way of building it, which may not be the same as the official one - and indeed it likely isn't. Several preprocessor macros (such as `DEBUG`) seem to be definable separately for the two parts (FW/libraries) - once in the `OnlyKey.ino` file and once in the `onlykey.h` file. Therefore the official way likely compiles the two parts separately, and

allows linking them with i.e. different build options using these macros. Since I didn't see much use in this for my work, I decided to simplify the process - I created my own directory called "husek_dip" within the "libraries" directory which would house both my additional debugging/helper code as well as these optional macros, removing them from their original places and removing their duplicity in the process. I can then build the whole firmware, including the libraries at once. My directory contains four files:

- `husek_serial.h` and `husek_serial.cpp` - these files contain functionality related to serial port emulation - see below.
- `husek_declarations.h` - this file contains forward declarations, to fix building errors.
- `husek_flags.h` - this file contains four preprocessor macros, which can be (un)defined to modify the FW functionality. The first macro is called DEBUG_OK and determines whether debug messages are printed through the (simulated) serial port. Originally the macro was called `DEBUG`, but I renamed it, because it collided with a macro in one of the libraries. The second macro is `STD_VERSION`. If it is defined, then the STD edition is built, if not the IN_TRVL edition is built instead. The third macro is `OK_Color` which is supposed to indicate, whether we want to compile for OKC or OKO. It still exists in the newer FW versions, even though the OKO isn't supported by them - I keep it defined even for my OKO testing unit. The fourth and last macro is `FACTORYKEYS`, which isn't actually used in our FW version yet - as is indicated by a comment in `OnlyKey.ino`. In the same file another comment regarding this macro says "// Attestation key and other keys encrypted using CHIP ID and RNG for unique per device".

I build the FW binary using `arduino-builder.exe`, a utility provided with the Arduino IDE. My building batch file (`build.bat`) is supplied on the attached CD, together with the modified version of the OK FW 2.1.1, which I have been using for most of my debugging - all edits I made inside the OK FW source code are marked with a comment saying `HUSEK EDIT`.

The code contains a lot of built in debug messages, which when the `DEBUG` macro is defined are supposed to be printed over the serial port using the Arduino functions `Serial.print` and `Serial.println`. However the Teensy actually doesn't natively act as a serial device [41]. Instead it uses HID protocol. This can be used to emulate the serial communication of the debug messages. At first I didn't realize that one device can have several different "streams" through which it sends data, and therefore it is possible to differentiate between system messages intended for communication with the OK App (or other OK software), and the debug messages. This system is used in the "windows-serial" utility, which is contained in a GitHub repository called

”node-onlykey-fido2”, made and owned by Bradley Matusiak, an advanced OnlyKey user, which maintains several external OnlyKey projects [42]. This utility can be used to detect the connection and disconnection of the OK devices and to print out the HID communication. Before I found out about this, I created my own similar solution - it uses the `hidprint` function from the `okcore.cpp` file to send both the OK supplied and my own debug messages through the same channel as the messages intended for the App. To filter them from the App, the messages have a predetermined structure. I have slightly modified the OK App file `OnlyKeyComm.js` to ignore messages from OK which begin with the substring ”SRL” - those are the ones produced by my serial emulation code. This ensures that the App ignores the debug messages and functions as normally. In turn I have created another small Python script which allows me to listen to the debug messages, while ignoring the system messages and also limiting some specific often-repeating messages so they do not overwhelm the output too much. I called it `my_serial_monitor.py` and it is available on the attached CD. The script uses a bit of code from the OnlyKey command line Python app - it is highlighted by a comment inside the script.

Since for testing with a modified FW I had to use the OKO, which isn't officially supported, I have made several changes to the FW to replace the missing OKC functionality. Code which would light up OKC's colorful LED has been replaced with the classic OKO monochrome LED plus a debug output informing me which color was supposed to light up.

I have sorted the OK libraries into several categories, according to their relationship to cryptography use, and we will go through them one by one. Some of them will be discussed in more detail than others, depending on their relevancy to my thesis.

2.3.3.1 Cryptographic libraries

These libraries contain cryptographic primitives, hashing functions and other related functionality.

- Crypto
  - This library contains both symmetric and asymmetric cryptography, as well as several hashing functions and MAC funtionality.
  - Is used in both the STD edition and the IN_TRVL edition - however the IN_TRVL edition uses fewer files.
  - The OnlyKey uses its Curve25519 cryptography and also its AES-GCM implementation.
  - Made by southern storm software, Pty Ltd.[43].
- mbedtls-2.4.0

  - This library also contains both symmetric and asymmetric cryptography, as well as several hashing functions and MAC funtionality.
  - It is used only in the STD edition.
  - The OnlyKey uses it for its RSA functionality.

- sha256
  - This library contains an implementation of the sha256 hashing function.
  - It is used in both the STD and the IN_TRVL editions.
  - The OnlyKey uses is for the hashing functionality, when making keys out of PINs.
  - Made by Brad Conte[44]

- sha1
  - This library contains an implementation of the sha256 hashing function.
  - It is used in both the STD and the IN_TRVL editions.
  - According to the `licenses.md` from the libraries directory, this library is supposed to be the sha1 implementation by Brad Conte just as the sha256 library, but actually it is not. Instead it is made by Peter "Cathedrow" Knight[45].

- tweetnacl
  - This is a self-contained public-domain cryptography C library, a "tweetable" (compact) version of the NaCl crypto library[46].
  - It is used in the STD edition and also in the BVL bootloader.
  - The BVL uses it for the sha512 hashing function to perform the FW integrity check and also its "Curve25519" asymmetric cryptography for FW signature verification. The onlykey can use its "crypto_box_keypair" to generate a random ECC keypair.

- uECC
  - MicroECC - contains a compact and fast implementation of ECDH and ECDSA cryptography[47].
  - It is used only in the STD edition, in conjuction with the fido2 library.
  - Made by Ken MacKay[47].

2.3.3.2 Crypto-related libraries

These libraries are used in conjunction with the cryptographic libraries, to built upon their functionality.

- fido2
  - This library provides the functionality necessary for the device to act as "universal second factor" within the u2f/fido2 schemes.
  - It is used only in the STD edition.
  - fido2/u2f - developed by Solokeys[48].
- flashkinetis
  - Used to interact with the Flash kinetis functionality provided by the chipset - to lock the flash memory after installing a fresh FW.
  - It is used in both the STD and IN_TRVL editions.
  - The file `licenses.md` from the libraries repo mentions the original can be found at https://github.com/FrankBoesing/Arduino-Teensy3-Flash/ but that website doesn't seem to exist anymore - it gives the 404 error.
- onlykey
  - This library is made by the OK authors themselves, and even though some of the other ones have been modified by them, this one definitely contains most of the OK specific code. It takes care of the normal functioning of the OK device - indeed the distinction between what will be placed in files within this directory and what stayed in the `OnlyKey.ino` file seems a bit arbitrary in certain cases.
  - It is of course used in both the STD and IN_TRVL edition.
- password
  - A generic library used for working with a user login password - in our case its used for handling the PINs.
  - It is used in both the STD and IN_TRVL editions.
  - Made by Alexander Brevig - credited within the main head file `password.h`.
- totp
  - We already mentioned this library in 2.1.2.1, is used to handle the Google TOTP second factor scheme.

  - It is used in both the STD and IN_TRVL editions.
  - Made by Luca Dentella, officially supported by Aruduino[27].

- ykcore and yksim
  - These are open-source libraries providing some of the YubiKey functionality that the OK is compatible with. Specificely this is an adaption of "libyubikey" for Arduino 1.0.x and for for Teensy 3.X.
  - They are used only in the STD edition.
  - Made by Simon Josefsson for Yubico[49].

2.3.3.3 Libraries that are not directly crypto-related

These are all the remaining libraries which are used in the compiled FW, but aren't very important from a security viewpoint.

- husek_dip
  - This directory has already been explained, it contains additional files made by me to help me with the OK study and exploration. Naturally it doesn't exist in the original repository, only in my local debugging copy.
  - It is used in both the STD and IN_TRVL editions.

- T3Mac
  - This library is used to get HW specific values like a MAC address. In our case it is used to get a "chip ID", which is then used to improve cryptography and RNG.
  - It is used in both the STD and IN_TRVL editions.

- tinycbor
  - Used for transforming data into the CBOR binary data representation.
  - It is used only in the STD edition, in conjuction with the fido2 library.

- Adafruit_NeoPixel
  - This library is used to control the colorful LED of the OKO.
  - It is used in both the STD and IN_TRVL editions.

  - "This library is written by Phil 'Paint Your Dragon' Burgess for Adafruit Industries, with contributions by PJRC, Michael Miller and other members of the open source community." - quote from the librarie's GitHub page[50].

- base64
  - This library provides simple base64 encoding and decoding capability.
  - It is used in both the STD and IN_TRVL editions.
  - Made by Brad Conte, just like the sha256 library[44].

- SoftTimer
  - Is used in both STD and IN_TRVL edition.
  - This library is made by Balazs Kelemen[38].
  - As was already explained the purpose of this library is supporting a task based programming style.
  - SoftTimer.h says: "SoftTimer library is a lightweight but effective event based timeshare solution for Arduino."

2.3.3.4 Unused libraries

These are the libraries which are included in the libraries repository, but do not appear to be used anywhere (their header files are not included anywhere and during the building process there don't seem to be any object files created from the respective cpp files). It is possible they are used during the debugging process, or that the authors may have planned on using them but later decided against it.

- WS2812Serial
  - This is a non-blocking WS2812 LED Display library.
  - Made by Paul Stoffregen, creator of the Teensy.
  - https://github.com/PaulStoffregen/WS2812Serial

- justhashtweetnacl
  - This is actually the same tweetnacl library that was already discussed, but everything except hash functions is commented out.
  - Originally I thought it might be used in the ITE, but it is not. My second idea was that it might be used in the BVL bootloader, for performing the FW integrity check, but BVL also requires primitives for signature verification, so it cannot be used there either - actually it doesn't seem to be used anywhere.

- InternalTemperature
  - It was probably supposed to be used to read temperature data from the internal Kinetis Cortex sensor.
  - Originally I thought it might be used for RNG, but it doesn't seem to be used anywhere.
- randombytes
  - Contains random number generation code. It is not used anywhere.

The distinction made between STD and IN_TRVL libraries was made mostly by inspecting "include trees" generated from the source code using GCC and the arduino-builder output, which shows which compiled object files are being linked into the final binary. These files are available for inspection on the attached CD.

2.4 Used Cryptography

There is a lot of cryptography used in the OnlyKey. Currently there is a function called `okcrypto_split_sundae` which is used for user data encryption. A fitting comment in the source code (file `okcrypto.cpp`) says this: "Just like an ice cream sundae, this function mixes the best crypto algorithms, together with multiple keys in order to mitigate side channel attacks against, a single algorithm or key. State is split so that each crypto function only, has access to part of the state."

I will not go into detail about how every kind of data is encrypted and which way PINs are hashed and verified - it is unnecessary, because the OK documentation already goes into great lengths to describe all these processes very well - especially the Advanced paragraph[21]. I have gone through the source code and can verify the claims made on this page within the "About OnlyKey PIN, profiles, key derivation, and encryption" subparagraph and the "How does OnlyKey encrypt backup data" subparagraph.

Chapter 3

What I found

In this chapter we will discuss my findings and observations. These could either be vulnerabilites, documentation/function inconsistencies or other undocumented information that I found, and considered to be interesting for the purposes of this thesis.

CryptoTrust also has an official OnlyKey bug bounty program[51], however none of my findings satisfy the criteria of fitting into one of their thread models. This is good news - it means I haven't been able to find any catastrophically serious vulnerabilities.

Mostly everything described in this chapter that has a practical effect (some of the paragraphs contain observations which might not have any observable effects) has been tested by me on the physical OK device and is repeatable and demonstrable. Especially the two findings from the first and most important section and which follows immediately.

3.1 Attack scenarios and problematic areas

This section describes what I consider to be the most impactful findings, when it comes to security.

3.1.1 The significance of two PINs

The OK should definitely be more clear, about the usage of two profiles with two different PINs. Since the OK Duo came out, where all the 24 slots are available by using only one PIN, it seems the usecase of having each profile contain credentials of different importance/secrecy was not intended to be viable. This is never explicitly stated, and I assumed it would be a viable use case. For example, I could use one of the STD profiles to store logins for all services which have access to money spending and the second STD profile for all the rest - social media, e-mail, etc. This seems like a pretty reasonable situation to me. However, as the OnlyKey stands now, if this STD/STD setup

is used, it is enough to know one of the PINs and we can access all the data. There are actually several ways to achieve this:

- Login to the profile for which we know the PIN, then enter into config mode and change the second PIN to a new one.
- Login to the profile for which we know the PIN, then enter into config mode and change the passphrase to a new one. After this we can have the OK write out the backup, then setup a new OK with the same chosen passphrase and perform a restore. Since the backups do not use any of the PIN information in their encryption, we will have gained access to data from both profiles.
- Brute-force the second PIN by trying all possible combinations. To bypass the limit of 10 tries, simply try 8-9 times, then login with the PIN we know and repeat. It is of course possible that we hit the SD PIN first and this will ruin our attack. As it is now the SD PIN can also be changed through config mode with the knowledge of only one of the PINs - this gets rid of this problem.

The main problem seems to me to be, that config mode is generic for both profiles at once, however it can be entered using any of the two PINs. This is actually contrary to what the App says on the Setup tab - "...Re-enter your current primary PIN to enter config mode..." 2.6, but it doesn't have to be the primary PIN.

Note that none of these are possible when PD mode is used. This is mainly because the PD mode profile cannot be used to enter config mode. Whats more, there is special functionality implemented to stop even the brute-forcing problem, when PD mode is used. It works by counting all failed login attempts in another counter called
textttsincelastregularlogin, which doesn't reset when the PD profile is logged into. If we fail to login 20 times, without logging into the primary profile (it doesn't matter how many times the PD profile was used in between), the primary profile gets silently destroyed, by having its hash wiped - therefore it will become impossible to login to it. Note that the successful logins into the PD profile do not reset this counter, however they also do not increase it.

If the PINs being on the same level and interchangeable is a feature and not considered a problem, I think it should be very clearly communicated to the users. If not, I think there are possible solutions, for example: before changing a profile PIN, always require that the original PIN of that profile is entered first, so that it can't be changed with just the knowledge of the second one. Before changing the passphrase, require either - the original passphrase, both PINs, or at least the PIN, which wasn't used to login for the current session - that way the user can show they know both PINs. Before changing

the SD PIN, again, require the original SD PIN, both PINs, or the PIN, which wasn't used to login for the current session.

Stopping the brute-forcing problem totally (even with the SD PIN caveat fixed) is more difficult and with the current model of usage probably impossible. This is because when both profiles are STD, the OK device has no way of knowing, which PIN the user is attempting to enter, if they fail - this means we cannot have two separate counters. However a shared counter has to be reset by every login, since we cannot just disable one of the profiles, if it hasn't been used for a long time. This is a big difference when compared to the PD mode usecase, where one profile is clearly more valuable and where it is anticipated that the user might be under pressure to give up their PIN and so hiding the STD profile data is more important. An actual solution thus, would require a change in the usage model - for example the user would have to first specify to which profile they are attempting to login - for example by touching the 1 or 2 button on the OK - and only then enter their PIN. This would allow for the existance of two separate counters. However this might be considered too much of a hindrance to a smooth user experience and therefore might not be considered worth it.

3.1.2 PD/STD profile interactions

Since the point of the PD mode is to hide the existence of the main STD profile and pretend, that the device is running the IN_TRVL edition of the firmware, we need to watch out for any ways that could differentiate between them. The OK Docs cleverly notice[52] the fact, that if somebody really wanted to know which FW we are using, they could just input 10 wrong PINs and then go through setup and take note, whether the STD functionality is or is not available. Therefore, to take full advantage of the PD mode, full wipe mode should be enabled - if the attack is attempted then, the FW will be wiped and there won't be any way to know which edition it was. This is the reason why the IN_TRVL edition and PD mode both allow for the full wipe setting to be selected even with their lack of config mode - this is necessary to maintain the illusion that the FW was indeed IN_TRVL, since when the FW gets wiped, we have to know that that edition allows for that setting to be applied.

There is however another interesting fact - if you enable sysadmin mode in the STD1 profile, it enables for the PD profile as well. This option of course shouldn't be available for IN_TRVL edition and therefore gives away the fact that we are using the PD profile - for example, since slot values have to be set through config mode, when sysadmin mode is enabled, this makes it impossible to change slot values in the PD profile. Similarly it will become impossible to change any preferences, since they all also require config mode, after the sysadmin mode is activated. This fact combined with the double PIN-fail counters used with PD mode, can actually get the device into a peculiar state. If the PD counter fills up and the STD profile gets disabled, while the

sysadmin mode was active, the PD mode becomes totally unchangeable, since it cannot be used to disable the sysadmin mode itself.

The solution seems pretty simple - have the sysadmin mode be separate for the two profiles - the PD profile cannot use it at all, and two STD profiles can benefit by having more granular preferences control - it could be useful to enable sysadmin mode on one of the profiles but not the other.

3.2 Minor potentially problematic areas

This section describes findings or observations which have something to do with security, however I wouldn't consider them particularly serious.

- When the BVL bootloader performs the FW integrity verification during bootup it uses a function called `fwintegritycheck` as can be seen in 3.1. The function returns a value of 1, indicating success, if the computed hash and stored hash match. However, it also returns 1, if the stored hash has all bytes set to 0xFF - this is a situation indicating that no hash was written yet. However since it is the BVL bootloader itself that writes the FW hash into EEPROM while installing it, I don't see any reason for this condition to be here. Indeed it only increases the attack surface - if someone found a way to overwrite the FW hash to this value, the integrity check would always pass. However on the whole I do not consider this to be particularly serious. Especially since what the integrity check is actually supposed to do, is prevent unauthorized changes in the installed FW - but the FW is stored in the same flash memory as the BVL bootloader is and considering the properties of the chipset, the EEPROM is actually also part of the same flash memory. It seems to me therefore that if an attacker were to find a way to overwrite the FW, they could also overwrite its hash to the correct value, or for that matter overwrite the bootloader as well and disable the integrity check.

- In the Beta 7 version of the OK FW you can disable backup key value/mode changing through preferences in the PD mode profile, and it also applies it for the primary STD profile. Since this is an irreversible operation performed by a profile which doesn't even support the backup functionality, this is clearly wrong - however it was fixed in the later versions. However until we use the BVL bootloader with downgrade prevention, it is always possible to go down to Beta 7, exploit any potential vulnerabilities and then optionally go back to the newer FW.

- We do know that the OK device has open-source software, however we should note that it is not open-hardware. There are no Arduino or other hardware sketches available to study. This might be problematic

for some users. Also the manufacturers website[13] does not contain much information. Indeed it has been pretty difficult to find anything about Black Vault Labs LLC. They might be the same people who are behind the OK sotware, but we cannot be sure, until they confirm it.

- It is good to realize, that if the host machine is setup to continue supplying power to the USB ports after it has been shut down, an unlocked OnlyKey doesn't have any way of knowing that the machine is no longer running and stays unlocked. Of course with the timeout autolock function it will probably only stay unlocked for a short time, but it is worth noting.

- It is nice for the legitimate user that the two STD profiles are differentiated by a different colored LED light (green for STD1 and blue for STD2), however this also means that if somebody sees your OK glowing blue, it immediately tells them that you have two profiles set up. This is probably not a huge problem, but for added convenience and an even prettier user experience I would suggest to make the colors for both profiles changeable through the preferences menu.

- I am a bit suspicious of the way the secure backup function works. It only restores what was setup at the time it was backed up, and doesn't wipe the data that is currently setup - even within the same slot. Meaning that if we have slot 1a initialized with a username and password and restore from backup that only contains a password for that slot, only the password will get overwritten, while the username will stay as it is. While this is mentioned in the documentation[30], which means it is the expected documented behavior, and even though I wasn't able to come up with a situation where it might be misused, it seems to me to be at the very least a bit non-standard.

- When the PD mode special counter gets triggered only the STD1 PIN hash gets overwritten, and not the actual user data stored in that profile. This is a long shot and probably doesn't matter at all, since if anybody was able to read the data from the EEPROM, they could have done it before attempting to brute-force the main PIN, etc. It is possible that this decision was made to lower the probability of a successful side-channel attack, which might be able to recognize that a failed login attempt was suddenly different (took a longer time to complete, needed more power, emitted differently in the EM spectrum,...) and therefore reveal the PD mode scheme and the existance of the (now defunct) STD profile.

- Since the OK acts as a physical keyboard, it is definitely possible to eavesdrop on the keystrokes, using either a SW keylogger program, installed on the host machine or a HW keylogger installed inside it, sitting

between the outward facing USB hub and the motherboard. However this problem is absolutely not OK specific and pertains to all keyboards. And even if this attack is carried out, the OK has a good advantage over SW password managers in that it only reveals the passwords it types, not all the other ones.

Figure 3.1: The code the BVL bootloader uses to check the FW integrity

```
...
  }
//read 64byte hash from eeprom
  for (int i = 0; i < crypto_hash_BYTES; i++) {
// 0 used for bootloader jump flag, 1 used for fwload flag,
// 2-65 used for fw integrity hash
  temphash[i] = GET_STORED_HASH(i);
  hashsum = hashsum + temphash[i];
  }
  if (hashsum == 16320) {
  //All FFs, default state, no hash written
    return 1;
  } //Check match
  else if (crypto_verify_32(temphash, hashptr) == 0 &&
     crypto_verify_32(temphash+32, hashptr+32) == 0) {
    return 1;
  }
return 0;
```

3.3 Other interesting findings

This section contains purely facts that have for some reason caught my attention and seemed to be interesting enough to be put in this thesis, however I do not claim that they have any security implications at all.

- The `Password::is(char* pass)` function as seen here 3.2 from file `password.cpp` from the password library has a mistake in it. The while cycle will always perform all `MAX_PASSWORD_LENGTH` iterations, or none at all. This is because the second part of the condition continually checks the first character of a non-changing pointer for not being 0. Since it is not changing during the cycle, it will either be a zero in all iterations or something else in all iterations. This seems like it could be a pretty serious mistakes which in some cases might for example read out of bounds

of a supplied buffer - however not in this case, since the buffers that are given here will always be sufficiently large.

- In the same file there is a function called `Password::evaluate()` as seen here 3.3 - at first I thought this function might be vulnerable to a timing sidechannel attack, since it returns false as soon as it finds mismatching characters. However then I realized this function isn't ever used to verify anything as an authentication challenge, but only to verify that the user didn't make a typo while creating their OK PINs - the function compares the result of the two times the user enters their desired PIN and checks if they match. In this context this behavior is not problematic.

- When setting the LED brightness level, the OK App is the one sanitizing the input - the OK device doesn't really check it for boundaries, of course the OK App can be modified to not check the input either. By studying the source code I came to realize that what actually happens with the received 8-bit value (lets call it N) is: `((N)*22 AND 255)+1` - where AND is a bitwise "and" operation. The result of this formula is the actual new brightness level. We can check it for discrete extremes and we find that the actual highest brightness setting achievable is not sending a "10" but sending either a "93" or "221". The lowest brightness can be set by sending either a "35" or a "163" - the difference vs a "1" is actually pretty clearly visible here. We can also send a "0", which turns the LED off.

- The Beta 7 changelog claims - "Better touch sense on OnlyKey buttons using automatic touch sense calibration"[25] - however I didn't find anything reflecting this change in the FW code. It is mentioned again in the 2.1.2 FW changelog - "Better touch sensitivity with touch sense recalibration"[25]. It seems my version of OK App doesn't support this setting yet, however the FW might because there are corresponding EEPROM getter and setter functions in the `okeeprom.c` file from the onlykey library. The getter returns an incorrect macro as an indicator of the values length - it returns `EElen_ledbrightness`, which is supposed to be used for the LED brightness setting, instead of returning `EElen_touchoffset`. It is a mistake, but it doesn't really matter in this case, because both macros evaluate to the same value, which is 1. Just to be sure I went through all the other "pos" and "len" macros which are used to store all the settings and user data in the EEPROM, sorted one value after another in a predefined position. I didn't find any other irregularities.

- The OK allows for the lock button to be set differently for the two profiles. However since the value checking is performed in the OK App, where it can be disabled in code, and not on device, we can send any

number we want. This wouldn't be very interesting, however thanks to how the lock button setting code is written 3.4, it is actually possible to change both lock buttons at the same time from the first profile's preferences. For example we can send a value of 50 (0011 0010 in binary) and this will set the STD1 lock to button 2 and the STD2/PD profile lock to button 3. This only works if there was not lock button set previously (there was a value of zero), since the code adds the new values to the old ones - so if a lock button was already set in the second profile - this way we would increase it by the amount specified, instead of setting it to that value. I found this to be quite interesting, but didn't find any way to actually misuse it.

- There is no way to attack the OK by cutting off power either after an incorrect PIN has been entered or before reaching the maximum length of 10 numerals - the failedlogins actually gets incremented and saved right after pressing the first button, and only gets reset again, if the login happens to be successful.

Figure 3.2: The code that the password library uses to copy the password guess into another buffer

```
//evaluate a string, is it equal to the password?
bool Password::is(char* pass){
        byte i=0;
        while (*pass && i<MAX_PASSWORD_LENGTH){
                guess[i] = pass[i];
                i++;
        }
        return evaluate();
}
```

3.4 Documentation vs. function inconsistencies

In this section I will highlight a few inconsistencies between what the online documentation, the comments in the code or the instructions in the OK App say and what the device actually does. Note that none of these necessarily mean there is a potential security vulnerability, but it is definitely of interest from the security point of view.

The official sources say:

- "Keep in mind that switching from the Standard Edition to the International Travel Edition will disable features not available such as the

Figure 3.3: The code that the password library uses to compare two passwords for match

```
//is the current guessed pass equal to the target pass?
bool Password::evaluate(){
 char pass = target[0];
 char guessed = guess[0];
 for (byte i=1; i<MAX_PASSWORD_LENGTH; i++){
//check if guessed char is equal to the password char
  if (pass==STRING_TERMINATOR &&
   guessed==STRING_TERMINATOR){
//both strings ended and all previous characters are equal
   return true;
  } else if (pass!=guessed ||
              pass==STRING_TERMINATOR ||
           guessed==STRING_TERMINATOR){
//difference OR end of string has been reached
   return false;
  }
//read next char
  pass = target[i];
  guessed = guess[i];
 }
 return false; //a 'true' condition has not been met
}
```

second profile."[26] - This is actually only partially true. Yes, a fresh uninitialized installation of the IN_TRVL edition will make it impossible to set up two profiles. However, if two profiles (either both STD or STD1 + PD) were set up using the STD edition FW, and then we re-install the IN_TRVL FW, both accounts actually remain accessible, under their respective PINs. Of course, since the IN_TRVL edition presumes non-encrypted user data, it won't be able to read user credentials and other data from STD-set-up-profiles. When attempting to read them, the OK produces seemingly random data with random delays, because it is interpreting the data as if it was set up using the IN_TRVL edition. Note that data from a potential PD mode second profile will work normally, since these are also unencrypted.

- "This [IN_TRVL] version of OnlyKey firmware is designed to meet all international requirements in regards to encryption. It does this by not utilizing encryption at all. Because this version does not utilize encryption the device can be used in areas where encryption is forbidden and/or

Figure 3.4: The code for setting up lock buttons

```
okeeprom_eeget_autolockslot(&temp);
if (profilemode) {
        temp &= 0x0F;
        temp += (buffer[7] << 4);
        okeeprom_eeset_autolockslot(&temp);
} else {
        temp &= 0xF0;
        temp += buffer[7];
        okeeprom_eeset_autolockslot(&temp);
}
```

there are mandatory key disclosure requirements. This is particularly useful for international travel where the traveler would like to have secure portable access to accounts"[26] - it is clearly stated that there is "no ecnryption at all", however this is blatantly not true. First - the BVL bootloader is still present, conatains the cryptogtaphic tweetnacl librar, and performs both hashing operations - for FW integrity checking - and digital signature verification - when re-installing FW. But even the FW itself still contains cryptography - yes the user data is not encrypted and the corresponding "profilekey" not generated. However the public keys for the PINs are still generated in the same way - which means using the sha256 hashing function (as was the case in the older versions as well), and since the encryption change from Beta 8 FW there is also an additional step which uses elliptic curve cryptography ("Curve25519"), described in this section - 2.4. These are being used in the IN_TRVL edition as well, which means it does still contain some cryptography.

- "Setting wipe mode to "Full Wipe" ensures that not only is your sensitive data wiped when a factory default occurs but also the firmware is wiped."[30] - Actually, when full wipe is turned on and then a factory-default function called, the FW doesn't immediately get wiped. Instead it wipes the FW hash from EEPROM with all zeroes, so that when the device reboots, the BVL performs and fails the FW integrity check. The BVL then performs the actual flash wipe to delete the FW. This makes sense, because the it would likely be technically impossible for the FW to delete itself, while running. It is worth noting that the full wipe begins with wiping the userdata first, as if it wasn't enabled. This is I think a good design choice, since deleting the data is more important and also if somebody were to perform some kind of side-channel analysis, the differentiation would come only after the user data has already been deleted.

- "While unlocked after a successful PIN entry there is an integrity counter used by the firmware running on OnlyKey. If instructions are skipped over the integrity counter will become corrupt causing the device to restart and lock."[30] - This refers to two variables within the code called `integrityctr1` and `integrityctr2` respectively. These counters are periodically checked to make sure they contain the same value - the way it works that before a sensitive transactional operation is performed the first value gets incremented by one and after the operation is done, the second one is incremented instead. This way in between operations these two variables should always hold the same value. The documentation claims that this process starts after a successful PIN entry - but in fact it begins even before that, while the device is still locked. This is actually probably preferable, it is just strange that the documentation would be so specific in this respect, when its untrue.

- "The data stored on OnlyKey is encrypted with the strongest encryption available (AES-256-GCM) and most importantly is PIN protected. What this means is that if you lose your OnlyKey it is essentially useless without the PIN, nothing can be read from or written to it."[53] - however, as we know, not only AES-256-GCM is used, but also the salsa cipher.

3.5 Secure coding standards

These are a few observations I have made about the secure coding style within the open source OK software.

- OK has a big advantage in that it is not using much dynamic allocation. This helps prevent buffer overflows, double free errors and use after free errors. There is dynamic allocation within the libraries, but only one within OK code `large_temp = (uint8_t *)malloc(17904);` - it is in the `okcore.cpp` file. Weirdly this single dynamic allocation doesnt seem to ever be freed, however since its in the restore function, which will unconditionally perform a restart after the restore is finished, it probably doesnt matter - however it is definitely not recommended to leave unfreed memory.

- A thing that has bothered me a little throughout my study of the FW source code is there are definitely some stylistic programming problems:

  - Relatively heavy use of magic numbers
  - It is pretty common for some functions to be named in a misleading way
  - A lot of global variables

  - A lot of repeating code - both small scale, and relatively sizable code chunks

  While these things don't necessarily pose any danger by themselves, they are something that should be avoided if possible, because they can bring confusion into the codebase, both for the author and especially for any potential readers.

- It is good that the authors didn't attempt to implement their own crypto-primitives and user ones from relatively well known libraries. The use of the crypto seems to be very complex and it is obvious the authors are very proficient in its use.

Conclusion

The OnlyKey device seems to be very well secured, I wasn't able to find and catastrophic vulnerabilities. There are however a few changes that I would suggest, which I allude to when talking about areas that I considered problematic in chapter 3.

Meanwhile I also have a few suggestions/observations for the users, so that they can use their OK as securely as possible. At first it seemed user un-friendly to me, that the App and device allow the user setup PINs that overlap, therefore rendering some of them unusable. Then it occured to me, that it could actually be used to the defenders advantage - for example set the SD PIN to 5556665 and the STD1 PIN to 55566657. This way it is impossible to login into the STD1 profile, because the SD PIN gets triggered sooner and wipes the device. The legitimate user could then still login using the second profile, change the SD PIN or the STD1 PIN (preferably with the modifications I suggested in 3) and then use the STD1 profile as long as needed, before hiding it again. Of course this would not be a very comfortable way of using the device, but it would add another layer of security to one of the profiles.

The lock button function can be used in principally a very similar way. We can guess that a potential attacker that manages to find our device connected and unlocked would first try and print out the 1a slot, tapping the 1 button shortly - if we set this to be the lock button it will lock him out of the device. Alternatively we could cover the "most secret/important" credential with the lock button. Then the slot becomes unusable again, since its button triggers the lock function, however if we need to use it we could un-set the lock button, use the slot and then re-set it again. Again - not a very comfortable way of using the device, but it adds another layer of security to one of the slots.

If the user wants to take advantage of the PD mode I would definitely listen to the OK documentation recommendation and turn on the full wipe mode.

Also even though the OK does protect its users password very well, it

cannot protect the user from creating a weak password - therefore I would recommend generating random long passwords - after all, with OK we do not have to remember them.

Future work

There is still a lot of work that could be done to further study and investigate the capabilities and properties of the OnlyKey. For example a rigorous testing of the RNG mechanism is a whole topic on its own, and so is a security analysis from a secure-hardware point of view - meaning side channel analysis, tamper resistance analysis, fault injection, etc. Also since this thesis didn't really cover the newest addition to the OnlyKey ensemble - the OnlyKey Duo - that is also left to further exploration.

Bibliography

1. BRASEN, Steve. *Accelerating the Journey to Passwordless Authentication* [online]. 2019. Tech. rep. Available also from: `https://www.ibm.com/downloads/cas/G9PM5PN4`. Accessed on 05/9/2022.
2. STOFFREGEN, Paul. *Teensy USB Development board* [online]. [N.d.]. Available also from: `https://www.pjrc.com/teensy/`. Accessed on 02/24/2022.
3. STOFFREGEN, Paul. *Teensy Technical Specs Comparision Table* [online]. [N.d.]. Available also from: `https://www.pjrc.com/teensy/techspecs.html`. Accessed on 02/24/2022.
4. NXP SEMICONDUCTORS, INC. *MK20DX256VLH7 Product Information—NXP* [online]. 2012. Available also from: `https://www.nxp.com/part/MK20DX256VLH7#/`. Accessed on 02/9/2022.
5. NXP SEMICONDUCTORS, INC. *Kinetis K20P64M72SF1* [online]. 2012. Tech. rep. Available also from: `https://www.nxp.com/docs/en/data-sheet/K20P64M72SF1.pdf`. Accessed on 02/9/2022.
6. STOFFREGEN, Paul. *Teensy® 3.2* [online]. [N.d.]. Available also from: `https://www.pjrc.com/store/teensy32.html`. Accessed on 02/20/2022.
7. "FLIGEN"; STOFFREGEN, Paul. *Is the Teensy boot loader source available?* [Online]. 2017. Available also from: `https://forum.pjrc.com/threads/41751-Is-the-Teensy-boot-loader-source-available`. Accessed on 06/3/2022.
8. STOFFREGEN, Paul. *Getting started with Teensy USB development board* [online]. [N.d.]. Available also from: `https://www.pjrc.com/teensy/first_use.html`. Accessed on 06/3/2022.
9. STOFFREGEN, Paul. *Checking HalfKay Is Running* [online]. [N.d.]. Available also from: `https://www.pjrc.com/teensy/check_halfkay.html`. Accessed on 06/3/2022.

10. STOFFREGEN, Paul. *C code for Teensy: USB Raw HID - for building custom USB devices* [online]. [N.d.]. Available also from: https://www.pjrc.com/teensy/rawhid.html. Accessed on 03/6/2022.

11. STOFFREGEN, Paul. *Teensyduino - Add-on for Arduino IDE to use Teensy USB development board* [online]. [N.d.]. Available also from: https://www.pjrc.com/teensy/teensyduino.html. Accessed on 03/6/2022.

12. CRYPTOTRUST. *Products - CryptoTrust* [online]. [N.d.]. Available also from: https://crp.to/p/. Accessed on 05/23/2022.

13. BLACK VAULT LABS LLC. *Products* [online]. 2022. Available also from: https://www.blackvaultlabs.com/products.html. Accessed on 05/12/2022.

14. CRYPTOTRUST. *Team - CryptoTrust* [online]. [N.d.]. Available also from: https://crp.to/t/. Accessed on 05/23/2022.

15. CRYPTOTRUST. *OnlyKey Features — Docs* [online]. 2022. Available also from: https://docs.crp.to/features.html. Accessed on 05/26/2022.

16. STEINER, Tim. *OnlyKey - The Two-factor Authentication and Password Solution by Tim Steiner — Kickstarter / Updates* [online]. 2016. Available also from: https://www.kickstarter.com/projects/timsteiner/openkey-the-two-factor-authentication-and-password/posts. Accessed on 05/28/2022.

17. DONÁT, Takács; STEINER, Tim. *Cannot compile and install beta 7* [online]. 2019. Available also from: https://github.com/trustcrypto/OnlyKey-Firmware/issues/89. Accessed on 06/14/2022.

18. CRYPTOTRUST. *Upgrading Firmware From Beta6 to Beta7 — Docs* [online]. 2018. Available also from: https://docs.crp.to/legacyupgradeguide.html. Accessed on 01/12/2022.

19. STEINER, Tim. *OnlyKey DUO - Portable Protection For All of Your Devices by Tim Steiner — Kickstarter* [online]. 2021. Available also from: https://www.kickstarter.com/projects/timsteiner/onlykey-duo-portable-protection-for-all-of-your-devices. Accessed on 05/28/2022.

20. CRYPTOTRUST. *OnlyKey DUO - Dual USB-C and USB-A Security Key* [online]. 2022. Available also from: https://onlykey.io/products/onlykey-duo-dual-usb-c-and-usb-a-security-key. Accessed on 06/17/2022.

21. CRYPTOTRUST. *About Security — Docs* [online]. 2020. Available also from: https://docs.crp.to/security.html. Accessed on 04/12/2022.

22. HUNTER, Melissa. *Using the Kinetis Security and Flash Protection Features* [online]. 2012. Tech. rep. Available also from: https://www.nxp.com/docs/en/application-note/AN4507.pdf. Accessed on 03/7/2022.

23. CRYPTOTRUST. *GitHub - trustcrypto/OnlyKey-App* [online]. 2022. Available also from: https://github.com/trustcrypto/OnlyKey-App. Accessed on 06/16/2022.

24. CRYPTOTRUST. *Releases - trustcrypto/OnlyKey-App* [online]. 2022. Available also from: https://github.com/trustcrypto/OnlyKey-App/releases. Accessed on 06/16/2022.

25. CRYPTOTRUST. *Releases - trustcrypto/OnlyKey-Firmware* [online]. 2022. Available also from: https://github.com/trustcrypto/OnlyKey-Firmware/releases. Accessed on 06/15/2022.

26. CRYPTOTRUST. *International Travel Edition Guide — Docs* [online]. 2018. Available also from: https://docs.crp.to/ite.html. Accessed on 03/28/2022.

27. ARDUINO. *TOTP library - Arduino Reference* [online]. 2022. Available also from: https://www.arduino.cc/reference/en/libraries/totp-library/. Accessed on 02/15/2022.

28. YUBICO. *Yubico OTP* [online]. [N.d.]. Available also from: https://developers.yubico.com/OTP/. Accessed on 02/11/2022.

29. FIDO ALLIANCE. *FIDO2 - FIDO AllianceFIDO2 - FIDO Alliance* [online]. [N.d.]. Available also from: https://fidoalliance.org/fido2/. Accessed on 05/30/2022.

30. CRYPTOTRUST. *OnlyKey User's Guide — Docs* [online]. 2022. Available also from: https://docs.crp.to/usersguide.html. Accessed on 05/26/2022.

31. CRYPTOTRUST. *GitHub - trustcrypto/python-onlykey* [online]. 2022. Available also from: https://github.com/trustcrypto/python-onlykey. Accessed on 06/16/2022.

32. TRUSTCRYPTO. *trustcrypto/onlykey-agent* [online]. 2022. Available also from: https://github.com/trustcrypto/onlykey-agent. Accessed on 06/1/2022.

33. CRYPTOTRUST. *OnlyKey SSH/GPG agent* [online]. 2021. Available also from: https://docs.crp.to/onlykey-agent.html. Accessed on 05/22/2022.

34. BLACK VAULT LABS LLC. *Releases - onlykey/BVL-Bootloader* [online]. 2021. Available also from: https://github.com/onlykey/BVL-Bootloader/releases. Accessed on 06/12/2022.

35. "BIGPILOT"; STOFFREGEN, Paul. *How do I prevent EEPROM from being overwritten bij Teensy Loader?* [Online]. 2018. Available also from: https://forum.pjrc.com/threads/54446-How-do-I-prevent-EEPROM-from-being-overwritten-bij-Teensy-Loader. Accessed on 06/3/2022.

36. ”CSTACK89”; STOFFREGEN, Paul; ”PICTOGRAPHER”; ”ROBSOLES”; STEINER, Tim (”cr7pt0”); ” ”defragster. *Upload Hex file from Teensy 3.1* [online]. 2015. Available also from: https://forum.pjrc.com/threads/28783-Upload-Hex-file-from-Teensy-3-1. Accessed on 06/3/2022.

37. ”SIGI”; ”DEFRAGSTER”; ”TNI”; STOFFREGEN, Paul. *Blocking AM2321 Library with Arduino 1.8.2 / Teensyduino 1.36* [online]. 2017. Available also from: https://forum.pjrc.com/threads/43080-Blocking-AM2321-Library-with-Arduino-1-8-2-Teensyduino-1-36. Accessed on 04/6/2022.

38. ARDUINO. *SoftTimer - Arduino Reference* [online]. 2022. Available also from: https://www.arduino.cc/reference/en/libraries/softtimer/. Accessed on 02/15/2022.

39. CRYPTOTRUST. *trustcrypto/libraries: Libraries for OnlyKey Firmware* [online]. 2022. Available also from: https://github.com/trustcrypto/libraries. Accessed on 01/11/2022.

40. ”JPATHY”; MATUSIAK, Bradley. *compiling the firmware* [online]. 2017. Available also from: https://github.com/trustcrypto/OnlyKey-Firmware/issues/59. Accessed on 06/13/2022.

41. STOFFREGEN, Paul. *Troubleshooting Common Problems* [online]. [N.d.]. Available also from: https://www.pjrc.com/teensy/troubleshoot.html. Accessed on 03/6/2022.

42. MATUSIAK, Bradley. *bmatusiak/node-onlykey-fido2* [online]. 2020. Available also from: https://github.com/bmatusiak/node-onlykey-fido2. Accessed on 06/4/2022.

43. SOUTHERN STORM. *rweather/arduinolibs* [online]. 2022. Available also from: https://github.com/rweather/arduinolibs/tree/master/libraries/Crypto. Accessed on 05/14/2022.

44. CONTE, Brad. *B-Con/crypto-algorithms* [online]. 2015. Available also from: https://github.com/B-Con/crypto-algorithms. Accessed on 05/16/2022.

45. KNIGHT, Peter. *Cathedrow/Cryptosuite* [online]. 2010. Available also from: https://github.com/Cathedrow/Cryptosuite/tree/master/Sha. Accessed on 05/15/2022.

46. BERNSTEIN, Daniel J.; GASTEL, Bernard van; JANSSEN, Wesley; LANGE, Tanja; SCHWABE, Peter; SMETSERS, Sjaak. *TweetNaCl: Introduction* [online]. 2013. Available also from: https://tweetnacl.cr.yp.to/. Accessed on 04/14/2022.

47. MACKAY, Ken. *kmackay/micro-ecc* [online]. 2022. Available also from: https://github.com/kmackay/micro-ecc. Accessed on 05/7/2022.

48. SOLOKEYS. *solokeys/solo1* [online]. 2021. Available also from: `https://github.com/solokeys/solo1/tree/master/fido2`. Accessed on 05/3/2022.

49. YUBICO. *Yubico/yubico-c* [online]. 2022. Available also from: `https://github.com/Yubico/yubico-c`. Accessed on 06/19/2022.

50. ADAFRUIT INDUSTRIES. *Adafruit/Adafruit-NeoPixel* [online]. 2022. Available also from: `https://github.com/adafruit/Adafruit_NeoPixel`. Accessed on 06/14/2022.

51. CRYPTOTRUST. *OnlyKey Bug Bounty Program* [online]. 2022. Available also from: `https://onlykey.io/pages/onlykey-bug-bounty-program`. Accessed on 06/15/2022.

52. CRYPTOTRUST. *Plausible Deniability Setup Guide — Docs* [online]. 2020. Available also from: `https://docs.crp.to/pdguide.html`. Accessed on 05/24/2022.

53. CRYPTOTRUST. *FAQ – OnlyKey* [online]. 2022. Available also from: `https://onlykey.io/pages/faq`. Accessed on 06/5/2022.

Appendix A

Acronyms

OK OnlyKey

OKO OnlyKey Original

OKC OnlyKey Color

OKD OnlyKey Duo

AES-GCM The Advanced Encryption Standard block cipher in galois-counter-mode

BVL Black Vault Labs (Usually while referring to their OnlyKey bootloader)

SW Software

HW Hardware

FW Firmware

STD The standard edition of the OK FW, or a profile in such an edition, which is not in PD mode - sometimes also explicitly marked as STD1 or STD2.

PD Plausible deniability mode

IN_TRVL / ITE The international edition of the OK FW

HK Teensy's proprietary HalfKay bootloader

PIN Personal identification number - a numeric password

SD PIN Self-destruct PIN

RNG Random number generator/generation

Appendix B

Contents of enclosed CD

`FW_edition_lib_comparison` ..... - a directory containing files which I generated and used to compare which libraries were included and linked in the STD edition FW vs the IN_TRVL edition FW

- `Installers` . - a directory containing the installers necessary to get the OK FW building up and running
- `src` ..... - a directory containing all the source code files
  - `ok_fw_src` ... - a directory which contains my modified version of the OK FW source code files
  - `python_src` .... - a directory which contains my small helper Python scripts
- `Thesis` - a directory containing everything necessary to generate a PDF of this thesis