VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Monika Böhmová

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

NÁSTROJ PRO ZABEZPEČENÝ START A AKTUALIZACI FIRMWARE EMBEDDED MIKROKONTROLERŮ

TOOL FOR SECURE START AND UPDATE OF EMBEDDED DEVICE FIRMWARE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — MONIKA BÖHMOVÁ

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. JAN HAJNÝ, Ph.D.

SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**
obor Informační bezpečnost
Ústav telekomunikací

Studentka: Monika Böhmová ***ID:*** 203634
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Nástroj pro zabezpečený start a aktualizaci firmware embedded mikrokontrolerů

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte uživatelsky intuitivní PC aplikaci, která bude sestavovat šifrované binární soubory potřebné pro zabezpečený start a aktualizaci firmware běžícího na mikrokontrolerech. Využijte formát dat, příkazů a strukturu binárního souboru podle nejnovějšího NXP standardu. Výstupem práce bude vytvoření nástroje, který bude podporovat nejnovější generace NXP embedded mikrokontrolérů s ARM Cortex M33 jádry, konkrétně platformy s kódovým označením LPC, i.MX RT a Kinetis. Softwarový nástroj bude napsán v jazyce Python, bude multiplatformní, dostupný pro operační systémy Windows®, Linux® a Apple Mac®.

DOPORUČENÁ LITERATURA:

[1] Elftosb User's Guide [online]. [cit. 2019-09-02]. Dostupné z: https://www.nxp.com/docs/en/user-guide/MBOOTELFTOSBUG.pdf
[2] Getting Started with the MCU Flashloader [online]. [cit. 2019-09-02]. Dostupné z: https://www.nxp.com/docs/en/user-guide/MBOOTFLASHGS.pdf

Termín zadání: 3. 2. 2020 ***Termín odevzdání:*** 8. 6. 2020

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.
Konzultant: Petr Fajmon (NXP Semiconductors Czech Republic, s. r. o.)

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato bakalářská práce se zabývá tvorbou nástroje pro zabezpečný start a aktualizaci firmwaru embedded mikrokontrolerů. Jsou zde popsány kryptografické algoritmy a metody, které byly použity pro zabezpečení. Dále jsou zde vysvětleny části, které následně tvoří celý nástroj. Nástroj byl úspěšně vytvořený, otestovaný a v následujícím roce bude nasazený pro použití.

KLÍČOVÁ SLOVA

Zabezpečený start, aktualizace firmware, mikrokontroler, kryptografie, soubor SB3, Python.

ABSTRACT

This bachelor thesis covers creation of a tool designed for secure boot and update of firmware of embedded microcontrollers. Cryptography algorithms and methods, which were used for security are thoroughly described. In the following sections, independent subparts of which the whole tool comprises are explained. The tool was succesfully created, tested and will be deployed in the following year.

KEYWORDS

Secure boot, update firmware, microcontroller, cryptography, SB3 file, Python.

BÖHMOVÁ, Monika. *Nástroj pro zabezpečený start a aktualizaci firmware embeded mikrokontrolerů*. Brno, 2020, 57 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Nástroj pro zabezpečený start a aktualizaci firmware embeded mikrokontrolerů“ jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autorky

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu práce panu doc. Ing. Janovi Hajnému, Ph.D. za odborné vedení a podnětné návrhy k práci. Také bych ráda poděkovala konzultantovi práce panu Ing. Petrovi Fajmonovi za odborné vedení, konzultace a podnětné návrhy k práci.

Brno ....

podpis autorky

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Práce se věnuje tvorbě uživatelsky intuitivní aplikace, která bude sloužit k sestavování šifrovaných binárních souborů pro zabezpečený start a aktualizaci firmwaru čímž nahradí zastaralý přístup k těmto procesům. Aplikace je určena pro embedded mikrokontrolery s jádry ARM Cortex-M33. Pro vytvoření této aplikace je využit soubor SB3 (Secure Binary version 3), což je nejnovější standard společnosti NXP Semiconductors N.V. (zkráceně NXP). Aplikace bude multiplatformní, bude ji tedy možné využít pro operační systémy Windows, Linux i Apple Mac.

Práce je rozdělena na dvě část. V první části se budeme věnovat aplikaci z pohledu teorie, kdy se zaměříme na vysvětlení použitých systémů a technologií jako je procesor LPC55xx, SRAM (Random Access Memory), PUF (Physical Unclonable Function), TrustZone, CPU (Central Processing Unit), MPU (Memory Protection Unit) a speciální části paměti neboli pojistky zvané fuses. Dále budou uvedeny použité knihovny. Knihovny se využívají pro kryptografii, parsování a serializaci struktur. Také se zaměříme na zabezpečení za použití hešovací funkce SHA (Secure Hash Algorithm), šifrovacího algoritmu AES (Advanced Encryption Standard) a algoritmu pro podepisování ECDSA (Elliptic Curve Digital Signature Algorithm). Při tvorbě aplikace pak budou tyto funkce využity pro dosažení vysokého zabezpečení aplikace, která tak bude odolná proti moderním útokům.

Budeme se zde zaměřovat na dva nejdůležitější body – zabezpečený start a aktualizaci firmwaru. Proto jako další bude popsán zabezpečený start včetně formátů kontejnerů, používaných klíčů, které budou používány pro šifrování, datové formáty sekcí, příkazy, které bude používat soubor SB3 a princip ověřování podpisu obrazu. Poté se zaměříme na bezpečnou aktualizaci firmwaru, kde budou vysvětleny předpoklady a omezení aktualizace, včetně samotného procesu aktualizace firmwaru.

Druhá část pak bude věnována praktické implementaci včetně ukázek kódu. Zde bude popsána architektura aplikace a návrh způsobu tvorby aplikace. Dále se budeme věnovat programovému řešení, kde budou uvedeny již zmíněné ukázky kódu. Prvně bude vysvětlena tvorba generátoru souboru SB3, poté tvorba uživatelského rozhraní a poté testování dílčích částí. Druhá část bude uzavřena zhodnocením bezpečnosti souboru SB3 a doporučením vylepšení zabezpečení.

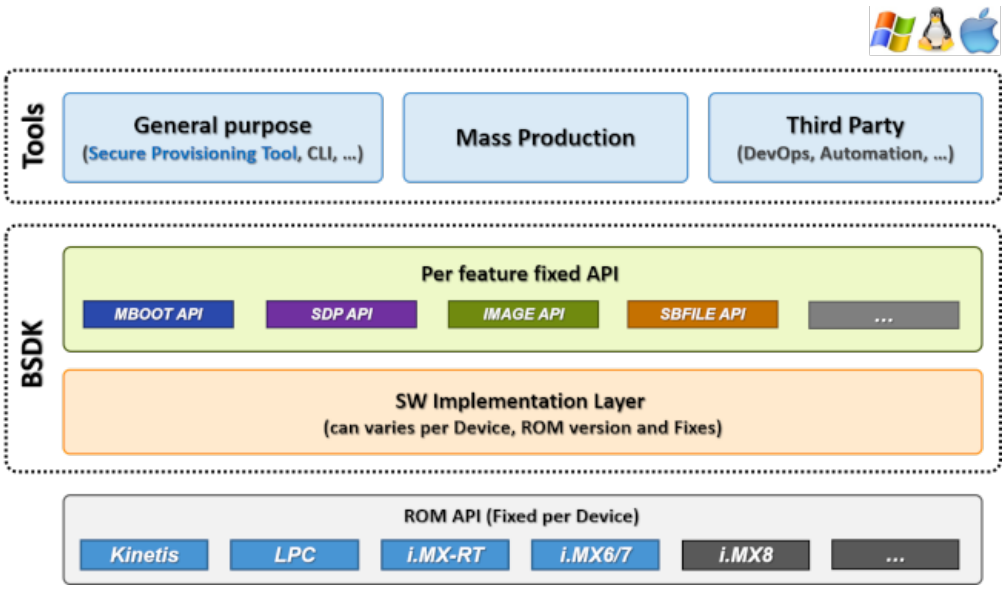
1 Teoretický rozbor

1.1 Systémy a použité metody

Tato práce se zabývá zabezpečeným startem a bezpečnou aktualizací firmwaru pro budoucí generaci embedded mikrokontrolerů s ARM Cortex-M33 jádry interně označovanou K4 [1]. Tato jádra jsou vyráběna společností Arm Holdings specializující se na výrobu polovodičů. ARM Cortex-Mxx je označení pro procesory určené k použití v mikrokontrolerech. Jádro ARM Cortex-M33 vychází z již dříve používaných jader Cortex-M4 a Cortex-M23 a má 3 fázové překrývání strojových instrukcí neboli tzv. „pipelining“.

Dříve byl na architektuře K4 využíván soubor SB2 (Secure Binary version 2) [2], avšak z důvodu nemožnosti pozměnění chování ze strany uživatele nebyl dostatečně vyhovující. V tomto souboru byla kryptografie řešena knihovnou, kterou uživatel nemusel chtít používat, protože knihovně například nedůvěřoval a nemohl snadno provést změnu.

Zabezpečený start by měl být prvním krokem k zabezpečení systému. Pod tímto pojmem je možné si představit proces, který zajišťuje to, že spouštěný software je v ověřené verzi a nedošlo k manipulaci třetí stranou. Z uvedeného důvodu vznikl projekt nazvaný Secure Boot SDK (Software Development Kit). Jedním z hlavních cílů projektu, bylo mimo jiné i vytvoření vylepšené verze souboru s označením SB3. Rozdělení tohoto projektu je zobrazeno na obrázku 1.1.



Obr. 1.1: Struktura projektu Boot SDK [2].

Hlavním rozdílem mezi souborem SB2 a SB3 je koncept oddělení implementace související s kryptografií. Uživatel nyní bude mít možnost zvolit si, zda bude používat výchozí implementaci, nebo použije svou vlastní. To znamená, že uživatel bude moci využít například knihovnu OpenSSL (Open Secure Sockets Layer). K tomuto uživateli bude stačit pouze drobná změna v kódu. Aplikace bude stále fungovat stejným způsobem, jen bude využívána jiná kryptografická knihovna.

Další rozdíl je v zabezpečení. Pro start a aktualizaci firmwaru byl v souboru SB2 využíván algoritmus RSA (Rivest–Shamir–Adleman). Aby bylo dosaženo vyšší bezpečnosti, budou v souboru SB3 používány ECC (Elliptic Curve Cryptography) klíče. Zde se používají afinní souřadnice X a Y bodu ECC. Konkrétně se jedná o algoritmus ECDSA, což je varianta protokolu DSA (Digital Signature Algorithm), která využívá eliptických křivek a používá se k digitálním podpisům.

V současné době se nevyrábí žádný embedded mikrokontroler, který by mohl využívat soubor SB3, jímž se zabývá tato práce. Z pohledu konstrukce architektury se mikrokontroleru, který bude moci soubor SB3 používat, nejvíce přibližuje procesor LPC5500 [3]. Tento procesor využívá technologii ARM Cortex-M33 společnosti ARM Holdings. Ve srovnání s předchozími generacemi má procesor LPC5500 vylepšenou architekturu a pokročilé zabezpečení včetně SRAM PUF a ARM TrustZone.

SRAM PUF [4] vytvořila společnost Intrinsic ID, která se zabývá vývojem softwaru. PUF je fyzicky neklonovatelná funkce, tedy fyzická entita reprezentovaná určitou strukturou. PUF využívá hlubokých submikronových variací, které běžně nastávají při procesu výroby polovodičů. Variace nezáměrně zajišťují, že každý jedinečný transistor má mírně odlišné, avšak náhodné elektrické vlastnosti.

V našem případě je můžeme využít jako jedinečnou identitu daného polovodiče. SRAM PUF je vytvořena na základě chování polovodičové paměti SRAM, která se nachází v digitálních čipech pro odlišení jednotlivých zařízení. Když je SRAM napájená, má každá buňka vlastní preferovaný stav což je výsledkem náhodných rozdílů v prahových napětích. Náhodnost vyjadřujeme ve spouštěcích hodnotách „neinicializované“ paměti SRAM. Odezva pak poskytuje jedinečnou a zároveň náhodnou binární posloupnost.

Jednou z nových vlastností jádra ARM Cortex-M33 je tzv. TrustZone [5], což je způsob rozdělení systému na důvěryhodnou a nedůvěryhodnou část systému. Díky tomuto rozdělení jsou v důvěryhodné části systému uloženy dílčí komponenty jako například zabezpečený start, aktualizace firmwaru nebo šifrovací klíče. Systémové zabezpečení je zajištěno hardwarově vynucenou izolací, která je zabudovaná do CPU. Zavedení TrustZone probíhá před paralelním spuštěním dvou domén a sdílením zdrojů v konfiguraci sady.

Dále je třeba zmínit novou architekturu K4, vycházející z předchozí generace LPC55xx [6]. Architektura K4 je vždy minimálně dvoujádrová a tvoří ji jádra ARM

Cortex-M33 a ARM Cortex-M0+. Cortex-M33 je hlavní jádro, které je určeno pro běh uživatelské aplikace. Naproti tomu jádro Cortex-M0+ není přístupné uživatelům a je využíváno pro NXP bezpečný podsystém. NXP využívá bezpečný podsystém zvaný Sentinel. Bezpečný podsystém zajišťuje dvě hlavní funkce, a to: generování, ukládání, správu tajných klíčů a běh bezpečných kryptografických služeb během bezpečného startu a běžného provozu.

Pro zamezení přístupu uživatelů k bezpečnému podsystému je v embedded mikrokontrolerech použit tzv. MPU [7]. MPU je hardware, který zajišťuje ochranu paměti a bývá implementován v procesorové jednotce CPU. Umožňuje softwaru (obvykle jádru operačního systému) definovat oprávnění přístupu do paměti. MPU monitoruje přístupy do paměti, načítání instrukcí a přístupy z procesoru.

Za účelem nezměnitelnosti části paměti jsou využity pojistky neboli tzv. fuses [6]. Pojistky jsou speciální části paměti. Do pojistek je možné zapsat pouze jednou, poté se již stávají neměnnými. Do pojistek je zapsán heš veřejných klíčů z certifikátů a symetrický klíč algoritmu AES pro dešifrování. Zápis do pojistek je omezen pouze na nuly, čímž se nám snižuje možnost zápisu z obvyklých 256 na pouhých 8. Jelikož je zde ve výchozím nastavení přítomno pouze osm nul. Nulu lze přepsat na jedničku, zpět na nulu je ovšem již přepsat nelze.

Dále je využíváno několik knihoven. Jedna z nich je easy_enum [8], což je implementace typu enum pro jazyk Python (enum je možné používat až od Python 3.4). Enum je výčtový datový typ, který je tvořen konečnou omezenou množinou hodnot. Výčtový typ je speciální hodnotový typ, který vychází z předem určené sady číselných hodnot. Každý jednotlivý prvek výčtového typu je tvořen identifikátorem a hodnotou.

Je zde také využíván modul struct [9], který je součástí standardní knihovny Python (lze používat pouze v Python 3.x). Tento modul slouží pro dvousměrnou konverzi mezi reprezentacemi hodnot v Pythonu a reprezentacemi struktur jazyků C a C++, které v Pythonu můžeme definovat jako bajtové objekty. Konkrétně se zde využívá string Format, který se používá pro určení očekávaného rozvržení při balení a rozbalování dat. Dále metoda pack, která zabalí zadané hodnoty v souladu se stringem Format a poté vrací bajtový objekt. Dále metoda unpack_from, která dle stringu Format rozbaluje zabalená data a vrací n-tici položek. Metoda calcsize, vrací velikost objektu.

Pro kryptografii jsou v souboru SB3 využívány knihovny cryptography, cryprodome a asn1crypto.

Knihovnu cryptography lze využít pro běžné kryptografické algoritmy (např. symetrické šifry, funkce odvození klíčů aj.), ale také pro kryptografické algoritmy vyšší úrovně , jako například hybridní šifry, kde systém asymetrického šifrování kombinuje výhody asymetrického šifrování s rychlostí symetrických šifer [10]. Data se šifrují symetrickým algoritmem, klíč použitý pro symetrické šifrování je zašifrován asymetrickým šifrovacím algoritmem a následně je přiložen k datům.

Jako analyzátor formátu DER je použita knihovna asn1crypto [11]. DER je binární formát pro datové struktury v ASN.1.

Knihovna cryprodome se používá pro parsování a serializaci struktur ASN.1 [12]. Konkrétně se zde využívá balík Crypto.PublicKey, kde například za pomoci metody import_key() můžeme načíst klíč.

1.2 Algoritmy a funkce použité pro zabezpečení

1.2.1 Hešovací funkce SHA

V nástroji je využívána kryptografická hešovací funkce SHA. Hešovací funkce slouží k vytvoření krátkého reprezentantu dat tím, že tvoří heš neboli otisk. Z tohoto otisku pak není možné zrekonstruovat vstupní data, jelikož jakákoliv změna vstupních dat vede ke změně výstupních dat. Hešovací funkce jsou využívány při vytváření podpisů, jelikož podpis zprávy za použití asymetrické kryptografie probíhá jako podpis heše dané zprávy.

Hešovací funkce jsou rychlé a očekává se od nich, že nebude možné z heše najít původní zprávu, tedy budou jednosměrné. Dále se očekává, že nebude možné najít dvě zprávy takové, že hodnoty jejich hešů by se rovnaly, tedy že by došlo ke kolizi.

SHA je rodina hešovacích funkcí (SHA-0, SHA-1, SHA-2 a SHA-3), kde například SHA-256 a SHA-512 jsou zahrnuty do skupiny označené SHA-2. Skupina SHA-2 je prozatím považována za bezpečnou. Rozdíl mezi uvedenými heši je, že SHA-256 má výstupní heš délky 256 bitů a SHA-512 má výstupní heš délky 512 bitů.

Z bezpečnostního hlediska můžeme říci, že SHA-512 je bezpečnější než SHA-256, protože v případě útoku hrubou silou (Brute-Force Attack) je třeba více zkoušek, jelikož SHA-512 má větší délku heše než SHA-256.

1.2.2 Algoritmus AES

Dále je v nástroji využíván algoritmus AES. AES je symetrická bloková šifra, která využívá k šifrování i dešifrování stejný klíč. Standardní velikost klíčů je 128, 192 nebo 256 bitů. AES nahradil algoritmus DES (Data Encryption Standard), který již nebyl dostatečně bezpečný. Narozdíl od algoritmu DES, který využíval Feistelovu

síť, využívá AES tzv. SP-síť (Substitution–Permutation). Jedná se o iterační schéma pro blokové šifry, které představuje sled operací rozptýlení informace v bloku. Můžeme tedy říci že SP-síť rozdělí prostý text do bloků o velikosti minimálně 128 bitů. Následně na blok opakovaně aplikuje substituci (nahrazení znaků) a permutaci (záměna pořadí) čímž se prostý text zašifruje.

Z bezpečnostního hlediska můžeme říci, že AES-256 je bezpečnější než AES-192 avšak je třeba brát v potaz, že dosud nejsou známy žádné metody, které by vedly k prolomení klíče a ani AES-128 nebylo možné prolomit za pomoci útoku hrubou silou (Brute-Force Attack). NSA (National Security Agency) doporučuje používat klíče o velikosti 192 bitů či 256 bitů.

Pro zvýšení bezpečnosti se u blokových šifer používají operační módy například ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback) a další. Tyto módy svazují jednotlivé bloky. V souboru SB2 byl používán mód ECB. Slabinou tohoto módu bylo, že pokud měl dva bloky v prostém textu stejné, měl i šifrované bloky stejné a útočník mohl šifrované bloky permutovat či duplikovat a tím docílit změn v prostém textu. V souboru SB3 bude používán mód CBC, který tento nedostatek řeší. Využívá IV (Initialization Vector), který se generuje pro každý prostý text. Aby byl dostatečně bezpečný, musí být nepředvídatelný. V opačném případě by mohl útočník provést útok prostého textu (Chosen-Plaintext Attack), při kterém si může útočník zvolit, jaký text bude šifrován a získá tak vzorky pro kryptoanalýzu.

1.2.3 Kryptografie s eliptickými křivkami (ECC)

Principy ECC jsou podobné jako u asymetrických systémů. Asymetrické systémy bývají založené na mocnění, u ECC se jedná o násobení celým číslem. Výhodou jsou kratší klíče, vyšší rychlost a menší náročnost na hardware. Velikost klíčů se odvíjí od použité křivky a je udávána v bitech.

Nejpoužívanější křivky pocházejí z dokumentů SEC-1 a SEC-2 (Standards for Efficient Cryptography) [13]. Jako příklad můžeme uvést křivky secp256r1 (velikost klíče 256 bitů), secp384r1 (velikost klíče 384 bitů) a secp521r1 (velikost klíče 521 bitů).

V současnosti jsou nejpoužívanější kryptografické algoritmy pro podpis RSA, DSA a ECDSA. Liší se hlavně v minimální délce klíče. V nástroji je využíván algoritmus ECDSA, který se používá pro podepisování a ověřování podpisu (nelze použít k šifrování) a je připraven na používání všech tří výše uvedených křivek.

V souboru SB2 byl používán algoritmus RSA, který je založen na problému faktorizace a pracuje s klíči o velikosti 2048 bitů. V souboru SB3 již bude využívána kryptografie s eliptickými křivkami ECDSA, která je založena na problému

diskrétního algoritmu, a to za pomoci hledání bodů na eliptické křivce. Pracujeme zde s klíči, které mají menší velikost. Jelikož ECC algoritmy jsou silnější než RSA, můžeme říci, že při porovnání algoritmu RSA s klíčem o velikosti 2048 bitů a algoritmu ECDSA s klíčem o velikosti 224 bitů, budou oba stejně silné [14]. Pokud ovšem použijeme ECDSA s klíčem o velikosti 521 bitů, bude již silnější než nejsilnější RSA.

Abychom mohli použít ECDSA k podepisování, je třeba použít heš, například SHA-256. Dále je třeba vygenerovat soukromý a veřejný klíč. Jak již bylo uvedeno výše, v nástroji můžeme použít klíče o velikosti 256 bitů, 384 bitů či 521 bitů, a to dle zvolené křivky.

1.3 Zabezpečený start

Pro zabezpečený start se na architektuře K4 používá knihovna s interním označením nboot, která umožňuje ověření podepsaného obrazu (prokázání, že nedošlo k pozměnění obrazu) a instalaci podepsaného šifrovaného obrazu firmwaru [6]. Funkce, které knihovna nboot poskytuje, jsou postaveny na dvou formátech kontejnerů (jeden kontejner pro každou operaci).

Jako první uvedeme formát kontejneru podepsaného obrazu, který je určen pro spouštěcí obrazy a ostatní části kódu, které se zde spouští. Pro vygenerování ECDSA podpisu obrazu, je třeba použít heš SHA-512, který se vypočítává z celého obrazu a privátního klíče ISK (Image Signing Key) certifikátu, případně privátního klíče z root certifikátu.

Veřejné ECC klíče jsou uloženy v CTRK (Customer Trusted Root Key) tabulce a jsou svázány k SHA-256 heši v pojistkách neboli fuses. Jak již bylo uvedeno v sekci 1.2, pro tvorbu veřejných ECC klíčů můžeme použít křivky secp256r1, secp384r1, secp521r1. CTRK tabulka může obsahovat až 4 veřejné klíče.

V hlavičce certifikačního bloku je specifikováno, který klíč se použije pro podepsání ISK certifikátu. Nad zřetězením veřejných ECC klíčů je vypočítáván CTRK heš.

Pro vytvoření ISK certifikátu je zapotřebí nejdříve vygenerovat veřejný klíč ISK (k tomu je použito náhodné číslo), poté je přiřazeno sériové číslo, do kterého se přidají ISK verze. Vše je následně podepsáno privátním klíčem ISK.

Druhým formátem je formát kontejneru firmwaru, který lze nahrát neboli SB3. Poskytuje autentizaci, integritu a důvěrnost. Je určen pro bezpečnou aktualizaci firmwaru, bezdrátově nebo přes sériový port.Používá se zde algoritmus ECDSA, využívající již zmíněné křivky secp256r1, secp384r1, secp521r1, algoritmus AES-256, využívající operační mód CBC a heš SHA-256.

Soubor SB3 se dělí do několika bloků. Všechny bloky mají stejnou velikost (typicky 324 bajtů), pouze první blok má odlišnou velikost (1068 bajtů), jelikož má

specifickou funkci a obsahuje odlišné části než ostatní bloky.

První blok (označovaný jako Block 0) je miniaturní podepsaný obraz, ve kterém, se nachází datová oblast podepsaného obrazu (datová oblast obsahuje hlavičku v otevřeném textu, ukazatel na tabulku certifikátů, ukazující do prvního bloku, AES-256 klíč následujícího bloku a SHA-256 heš následujícího bloku), certifikační blok a ECDSA podpis, který používá ISK nad prvním blokem. Veškeré další bloky jsou šifrovány algoritmem AES-256, využívající operační mód CBC[1]. V těchto blocích se nachází počítadlo bloků, datové sekce obrazu firmware, AES-256 klíč následujícího bloku, SHA-256 heš následujícího bloku.

Jak již bylo uvedeno výše, v podepsaných obrazech je možné používat dva ECC klíče – CTRK klíč a ISK klíč. CTRK klíč, je svázaný s hešem a ukládá se do pojistek. ISK klíč, je podepisován CTRK klíčem. V souboru SB3 je možné si klíče zvolit. CTRK a AES jsou povinné klíče, lze pouze zvolit, množství použitých CTRK klíčů. ISK je volitelný klíč.

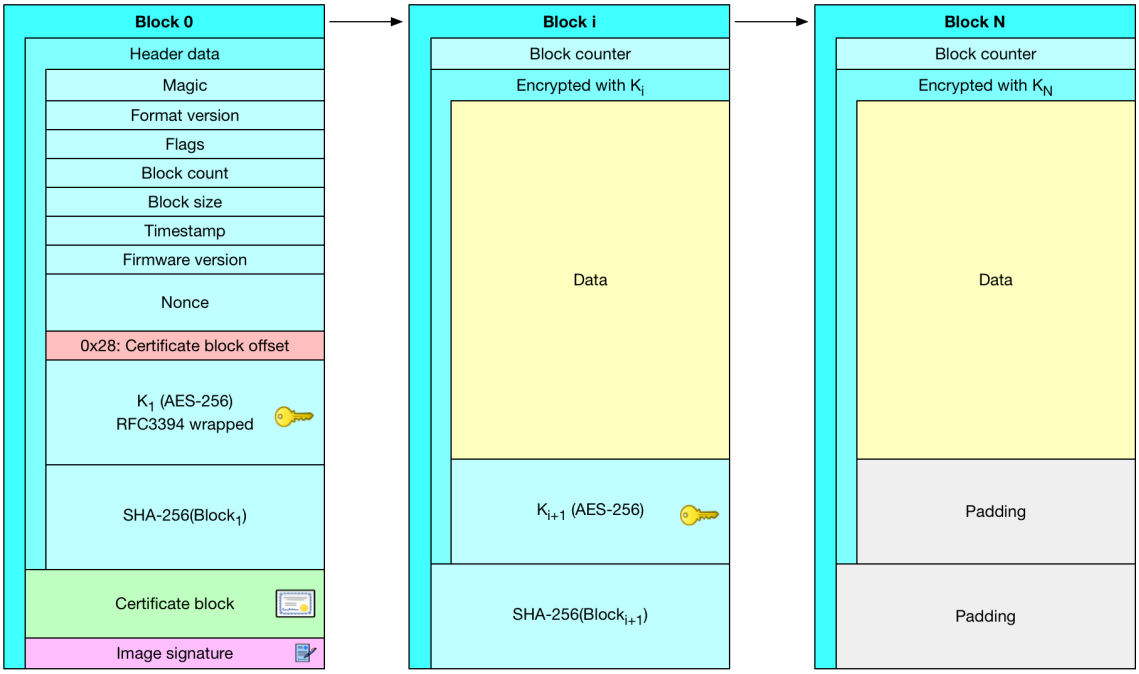
1.3.1 Datový rozsah sekcí

První blok souboru SB3 má tři hlavní části - hlavičku (header), která se skládá z více částí, certifikační blok (certificate block) a ECDSA podpis obrazu (image signature). Tyto části jsou zobrazeny na obrázku 1.2 kde je první blok číslován od nuly a je tedy blokem 0 (na obrázku je označen jako Block 0).

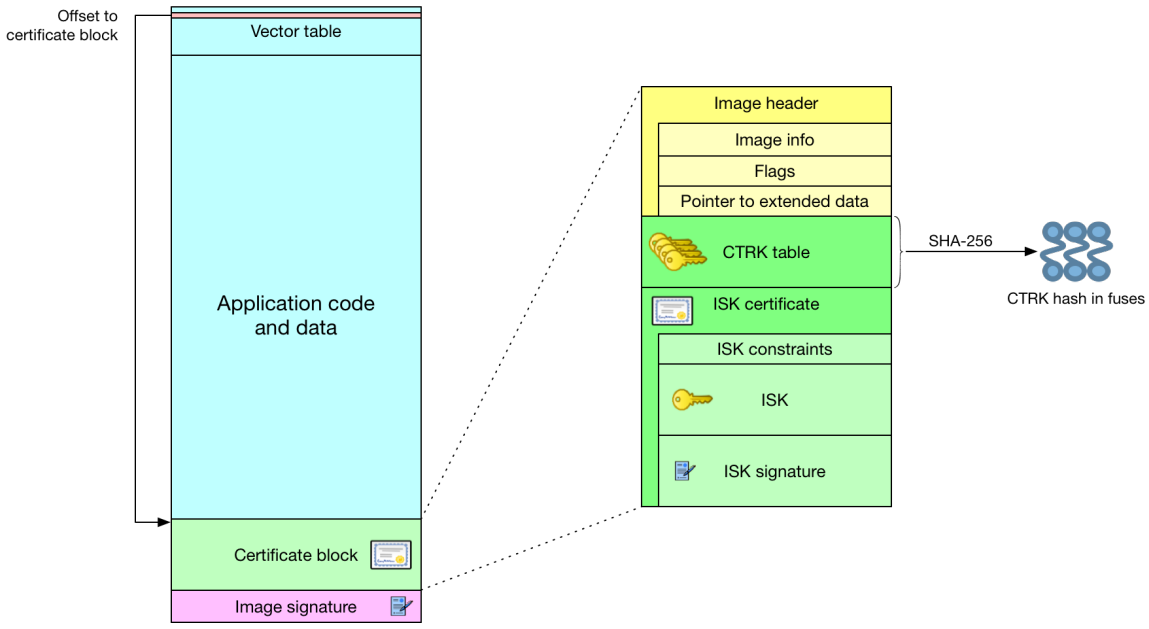
Certifikační blok je tvořen hlavičkou bloku, CTRK tabulkou, která obsahuje veřejné klíče a ISK certifikátem, který obsahuje veřejný klíč, verzi a podpis certifikátu. Tento blok se využívá při podepisování obrazu. Podrobně je certifikační blok zobrazen na obrázku 1.3. V případě souboru SB3 můžeme říci, že jedním blokem podepisujeme blok následující. Tento postup je zobrazen na obrázku 1.2.

Při pohledu na soubor SB3 zjistíme, že v blocích 1-n je umístěna za počítadlem bloků datová oblast. Data se dělí na části o stejných velikostech. V této práci pracujeme s velikostí dat 256 bajtů, ovšem uživatel si může velikost dle potřeby snadno upravit. Tyto části jsou následně umisťovány do vlastního bloku. Podle počtu částí se poté odvíjí i počet bloků. Data v posledním bloku jsou doplněna o výplň, aby i zde měly stejnou velikost. Rozdělení je zobrazeno na obrázku 1.2 v bloku I, označeném jako Block I.

[1] Jelikož zde nelze provést útok prostého textu (chosen-plaintext), kde by byl použit stejný klíč, jelikož se generuje nový klíč pro každý blok, lze zde použít operační mód CBC s IV=0.



Obr. 1.2: Struktura souboru SB3 [6].



Obr. 1.3: Podrobné zobrazení certifikačního bloku [6].

Pro výpočet velikosti datové části můžeme použít následující rovnici:

$$datova\_cast = velikost\_bloku - blok\_rezijnich\_nakladu \tag{1.1}$$

Budeme-li například chtít vytvořit datovou část o velikosti 256 bajtů, tak celková velikost bloku bude 324 bajtů, jelikož blok režijních nákladů neboli overhead má velikost 68 bajtů. Uvedený blok režijních nákladů, je tedy součet velikosti počítadla bloku (block counter) – 4 bajty, klíče bloku K_{i+1} (AES-256) – 32 bajtů a heše bloku (SHA-256) – 32 bajtů.

Jelikož podepisujeme blok předchozím blokem, tudíž u posledního bloku je již místo klíče následujícího bloku (AES-256) a heše následujícího bloku (SHA-256) pouze výplň (padding). Toto rozdělení je zobrazeno na obrázku 1.2, v části označené jako Block N.

Datový tok se rozděluje do sekcí, kde každá sekce má unikátní typ a ID. Délka sekcí je vždy násobkem 16 bajtů. Tato velikost bloku je vyžadována pro šifrování algoritmem AES.

1.3.2 SB3 Příkazy (cmd)

V datovém rozsahu může být prováděno několik akcí, pro které se používá šest příkazů: Erase, Load, Execute, Call, programFuses, programIFR. Každý příkaz má specifickou funkci. Tyto příkazy představují data, se kterými pracují bloky. Příkazy vycházejí z předchozích příkazů, které je možné najít v uživatelské příručce [15]:

- Příkaz Erase vymazává flash paměť, a to dle daného adresního rozsahu. Je třeba, aby byla specifikována adresa, odkud má být provedeno mazání včetně rozsahu. Vymazání se zaokrouhluje na velikost sektoru, který má být vymazán. Zaokrouhlování se vždy provádí směrem nahoru. Velikost příkazu Erase je 16 bajtů.
- Příkaz Load zajišťuje, že data pro zápis následují hlavičku rozsahu. Proto je třeba specifikovat adresu, kam mají být data zapsána včetně dat pro zápis. Pokud je třeba použít výplň musí se po datech připojit tak, že začátek dalšího rozsahu nebo hlavička sekce je zarovnaná na 16 bajtů. Velikost příkazu Load se odvíjí od velikosti dat, se kterými pracuje.
- Příkaz Execute zajišťuje, že jako počáteční adresa bude nastavená adresa směřující do konkrétního umístění v RAM (Random Access Memory), která za pomoci příkazu Execute začne vykonávat aplikaci z této adresy. Tuto adresu je třeba specifikovat. Velikost příkazu Execute je 16 bajtů.
- Příkaz Call zajišťuje, že jako počáteční adresa bude nastavená adresa směřující do konkrétního umístění v RAM. Tuto adresu je třeba specifikovat. Následně

je očekáván návrat na následující výraz, aby mohlo být pokračováno ve zpracovávání souboru SB3. Velikost příkazu Call je 16 bajtů.

- Příkaz programFuses zajišťuje, že počáteční adresa bude nastavená adresa registru pojistek neboli fuses. Příkaz programFuses zajišťuje zapsání specifikovaných slov do registrů pojistek, ty pak bezprostředně následují hlavičku. Adresu, kam mají být data zapsaná je opět třeba specifikovat. Jak již bylo uvedeno v sekci 1.1, do pojistek lze zapsat pouze jednou a poté se stávají neměnnými. Velikost příkazu programFuses se odvíjí od velikosti dat, se kterými příkaz pracuje.
- Příkaz programIFR zajišťuje, že počáteční adresa bude adresa v oblasti IFR (Indexed Flash Region)[2], délka bude v počtu bajtů, pro zápis do oblasti IFR. Data pro zápis do oblasti IFR je třeba specifikovat, včetně adresy oblasti. Data bezprostředně následují hlavičku. Velikost příkazu programIFR se odvíjí od velikosti dat, se kterými pracuje.

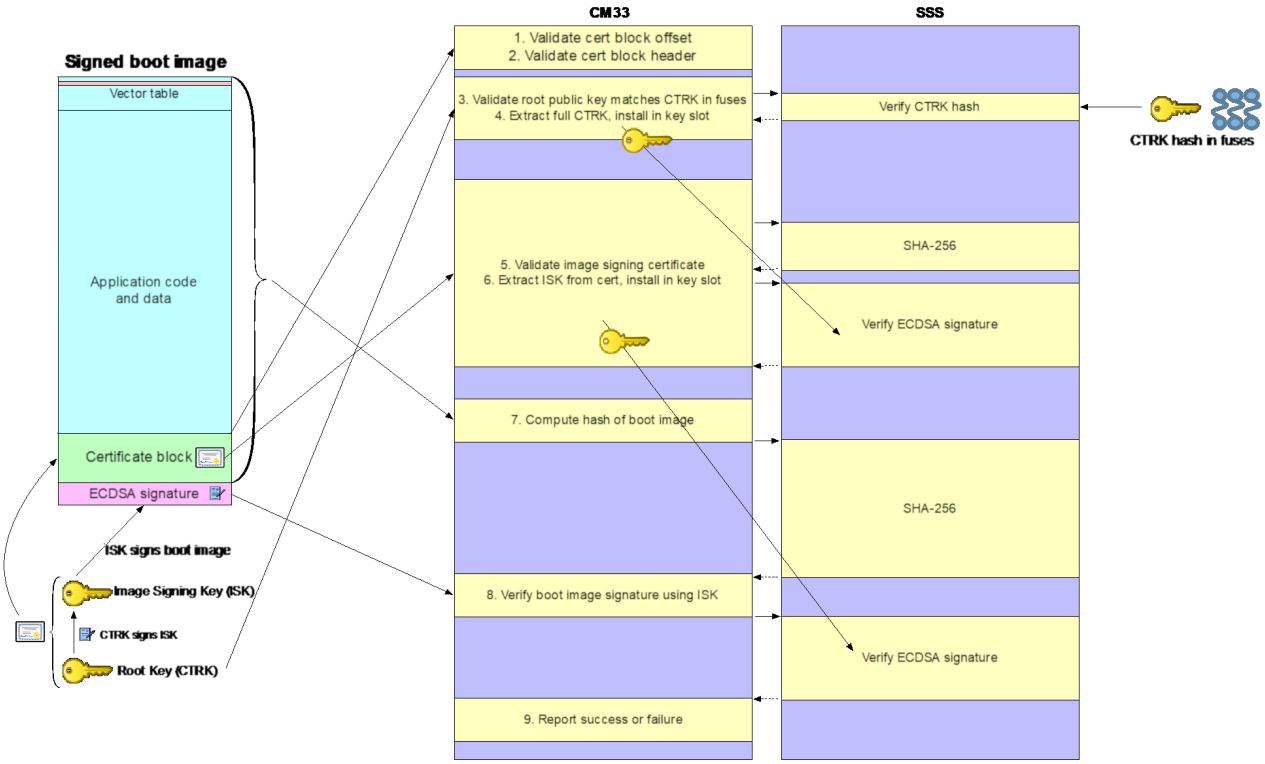
1.3.3 Ověřování podpisu obrazu

Pro ověření podpisu obrazu je dána konkrétní sekvence:

1. ověří se ukazatel na blok certifikátu, který je na adrese (s offsetem) 0x28 ve vektorové tabulce obrazu,
2. ověří se hlavička certifikačního bloku,
3. ověří se, že se kořenový veřejný klíč v bloku certifikátu shoduje s CTRK hešem, uloženým v pojistkách (fusese),
4. extrahuje se úplný kořenový veřejný klíč (CTRK) z bloku certifikátu a nainstaluje se ve slotu klíče v SSS (Security Sub-System),
5. ověří se certifikát pro podpis obrazu (CTRK podepisuje ISK),
6. extrahuje se podpisový veřejný klíč obrazu (ISK) a nainstaluje se do slotu klíče v SSS,
7. vypočte se heš (SHA-512) z celého obsahu obrazu,
8. ověří se ECDSA podpis použitím veřejného podpisového klíče obrazu (ISK), který byl předtím extrahován,
9. nakonec je nahlášen úspěch nebo selhání do jádra Cortex-M33.

Posloupnost ověření podpisu obrazu a vztah k operacím prováděným pomocí SSS jménem CM33 ROM je zobrazen na obrázku 1.4

[2]Oblast IFR je oblast paměti, kterou je například nastavováno chování ROM. Jsou zde například uloženy ROM patche.



Obr. 1.4: Posloupnost ověření obrazu [6].

1.4 Bezpečná aktualizace firmwaru

Architektura K4 vyžaduje, aby byla aktualizace obrazu firmwaru provedena bezpečně [6]. To znamená, že autenticita obrazu může být ověřena před instalací či spuštěním. Je velmi důležité, aby se neoprávněná osoba nemohla vrátit zpět k verzi obrazu firmwaru, která je nižší než aktuální minimální povolená verze. Je třeba zajistit, aby nenastala situace, kdy by vada v procesu aktualizace nebo nově nainstalovaném obrazu ponechala zařízení ve stavu, ze kterého by nemohlo být nikdy obnoveno do „bezpečného“ obrazu.

Rozlišujeme dva typy obrazu, a to validovaný obraz a verifikovaný obraz. Validovaný obraz je obraz, který je autentizován a má akceptovatelnou verzi. Verifikovaný obraz je obraz, který již funguje na zařízení. Verifikace obrazu může probíhat automaticky, kdy se nástrojem či aplikací ověřuje, že aktuálně běžící obraz, dosáhl bodu, ve kterém je funkcionalita obrazu verifikovaná a považovaná za provozní, do takové míry, že může proběhnout aktualizace obrazů firmwarů. Případně může verifikace proběhnout manuálně a to tak, že uživatel za pomoci manuální akce potvrdí, že provoz zařízení je v pořádku.

Jelikož vyžadujeme, aby byl obraz schopný alespoň úspěšně aktualizovat obraz firmwaru tak, aby bylo zařízení obnoveno do nového obrazu, jsou na architektuře

K4 určitá omezení a předpoklady pro dodržení této schopnosti.

Jako příklad můžeme uvést předpoklad, že verifikace obrazu, která ověřuje, zda obraz funguje správně, by měla pokrývat sadu spouštěcích obrazů pro všechny relevantní procesory v systému.

Dalším příkladem je schopnost řídícího jádra startu (na architektuře K4 se jedná o jádro Cortex-M33), která ukládá limitované množství dat, které přetrvávají i po restartovacích cyklech. A samozřejmě také možnost úpravy minimálního čísla verzí, které zprostředkovává bezpečný podsystém.

Za stabilní situaci můžeme považovat situaci, kdy minimální povolená verze firmwaru je označena jako verze *n*. Jsou dostupné dva obrazy (primární a sekundární), které jsou ověřené (validované a verifikované) a jsou s verzí *n*. Je možné mít dva obrazy s rozdílnými verzemi v určitém čase například jako výsledek aktualizovaného obrazu.

Úspěšné ověření obrazu vyžaduje, aby kryptografický podpis byl ověřený přes kořenový neboli root klíč, uložený v zařízení. Verze obrazu musí být minimálně taková, jako je vyžadovaná minimální verze, uložená v pojistkách (fuses).

Aby proběhla aktualizace firmwaru bezpečně, je třeba provést konkrétní sekvenci:

1. při normálním provozu, běží firmware s verzí *n*,
2. přijme se nový obraz firmware verze *n+1* a zapíše se do aktuálně neaktivního umístění obrazu (sem se zapisuje, abychom měli jistotu, že aktuální obraz zůstane dostupný, když by selhal proces aktualizace),
3. v tuto chvíli má systém v primárním a sekundárním umístění dva obrazy s rozdílnou verzí (*n* a *n+1*) a za použití FLW (Flash Logical Window Module) by mělo řídící jádro startu vyměnit umístění obrazů tak, aby umístění primárního obrazu ukazovalo na obraz s číslem nejvyšší verze,
4. na spouštěném či restartovaném zařízení, zkontroluje řídící jádro startu, jestli je nastavena značka pokusu o autentizaci (tedy že proběhl pokus o autentizaci primárního obrazu v předchozím resetovacím cyklu), pokud je tato značka stále nastavená, autentizace selhala a řídící jádro startu se pokusí bootstrapovat[3] systém z obrazu v sekundárním umístění, což probíhá takto:
   řídící jádro startu nastaví značku v tzv. reset-safe paměti neboli paměti, která není ovlivněna provedením restartu. Značka indikuje, že se pokusí o spuštění primárního obrazu, poté řídící jádro startu bootstrapuje systém použitím obrazu nalezeném v primárním umístění, který by měl být nově nainstalovaný firmware verze *n+1*, pak bezpečný podsystém, ověří obraz firmware verze *n+1*. Pokud je ověřen, bezpečný podsystém si ponechá nové číslo verze firmware pro

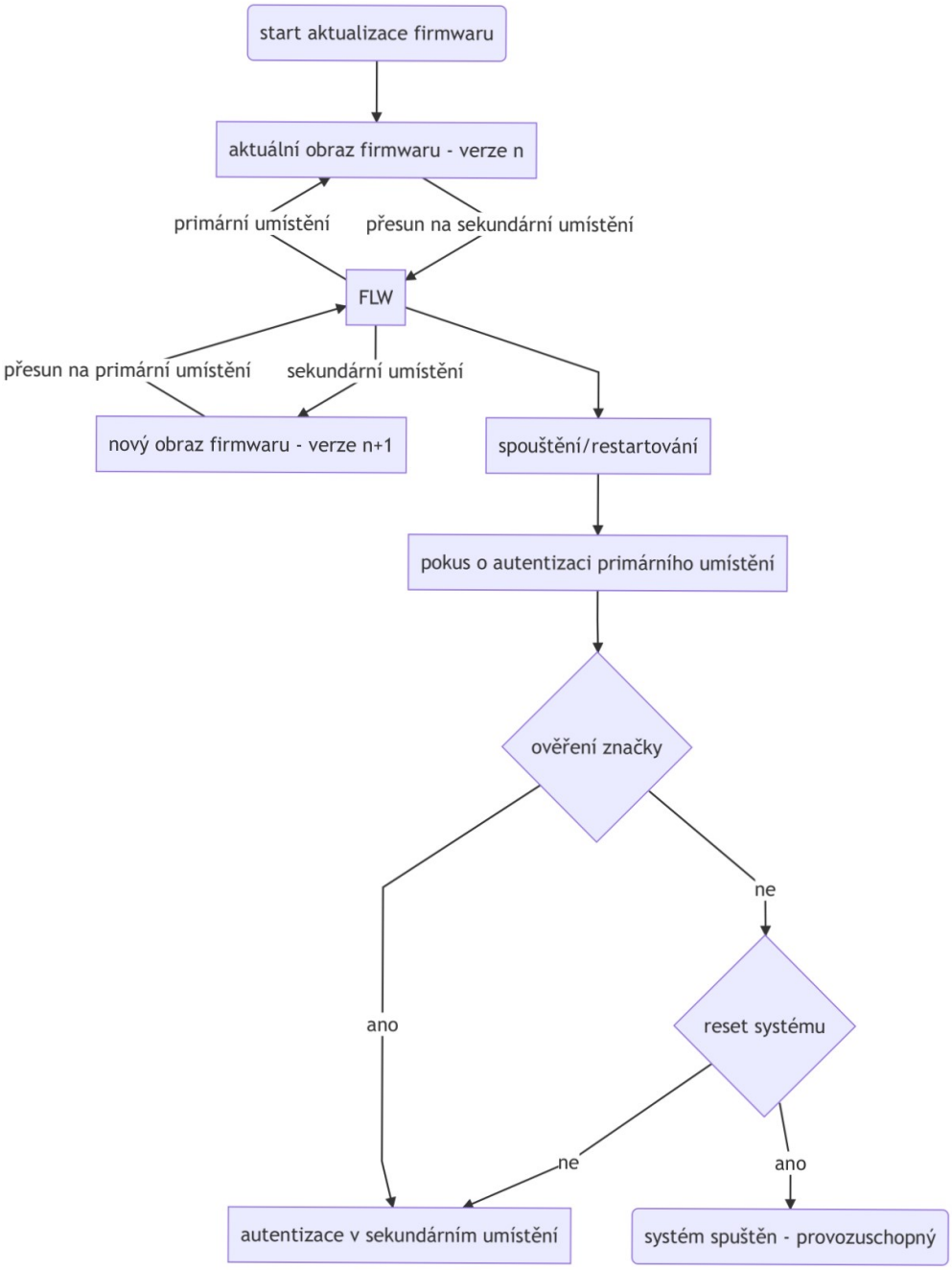
[3]Bootstrapovat znamená, že se za pomoci jednoduchého počítačového programu, BIOSu inicializuje hardware a případně i externí paměťové zařízení, poté se načte program do paměti, čímž je umožněno načtení složitějších programů, například operační systém.

případné pozdější předložení tohoto nového čísla verze do pojistek (pokud je později systém považován jako funkční),

5. pokud ověření selže[4], řídící jádro startu restartuje systém, pokud je ověření úspěšné, řídící jádro startu resetuje značku pokusu o autentizaci a systém má povoleno pokračovat ve spuštění,
6. pokud je sada obrazů provozuschopná, tak je systém již zapnut a běží. Vyšší úroveň softwarové entity v softwaru (typicky uživatelem nainstalovaný software), se potřebuje rozhodnout, zda je obraz dostatečně provozuschopný, aby byl považován za verifikovaný a pokud je považován za verifikovaný, je to oznámeno bezpečnému podsystému, který po přijetí oznámení aktualizuje minimální povolenou verzi firmwaru v pojistkách, na verzi firmwaru, která aktuálně běží.

Zpráva posílaná bezpečnému podsystému pro předložení nové minimální verze firmwaru, nemusí být autentizována, jelikož nové číslo verze bylo převzato z podepsaného kontejneru. Proces aktualizace firmwaru je zobrazen na obrázku 1.5.

[4]Selhání či úspěch ověření signalizuje bezpečný podsystém.



Obr. 1.5: Proces aktualizace firmwaru.

2 Praktická implementace

2.1 Architektura

Jak již bylo zmíněno v sekci 1.1, aby byla zajištěna vyšší bezpečnost a dosaženo vyšší rychlosti, vznikl v rámci této práce generátor souboru SB3, který využívá algoritmus založený na eliptických křivkách. Kód byl napsán v programovacím jazyce Python a pro otestování funkčnosti byly vytvořeny testy za pomoci standardní knihovny pytest. Jelikož repozitář obsahuje i předchozí verze, jsou třídy označovány verzí 3 a příkazy slovem New (nový). Programovací jazyk Python je beztypový, lze tedy do proměnné přiřadit jakýkoliv typ. Dle přiřazené hodnoty Python určuje, jak se má s danou proměnou zacházet. Jelikož soubor SB3, jehož tvorbou se tato práce zabývá, využívá části vytvořené v programovacím jazyce C++, je zde využíván modul struct, který umožňuje konverzi mezi bajtovými hodnotami v Pythonu a strukturami v C++.

V práci byl vytvořen generátor, který rozděluje zadaná data na části. Velikost každé datové části byla stanovena na 256 bajtů. Dále jsou vytvořeny bloky. Počet bloků závisí na velikosti zadaných dat. Pokud bychom měli data o velikosti 1578 bajtů, bylo by vytvořeno 7 bloků, přičemž poslední blok by obsahoval data a výplň nul (padding).

K těmto blokům je následně přidán blok 0, který je oproti ostatním blokům odlišný, a to nejen obsahem, ale i velikostí. Blok 0 obsahuje údaje o obrazu (verze firmwaru, velikost bloku a další), klíč AES-256 následujícího bloku, heš SHA-256 následujícího bloku, certifikační blok a ECDSA podpis obrazu. Celková velikost bloku 0 je 1068 bajtů.

Certifikační blok obsahuje údaje o certifikátu (verze, specifikace použitých klíčů z CTRK tabulky a další), ISK certifikát, obsahující veřejný klíč, podpis certifikátu, verzi a CTRK tabulku, obsahující až čtyři veřejné klíče. K podepsání obrazu (bloku 0) je zde využíván algoritmus ECDSA, soukromý klíč a heš SHA-512.

Ostatní bloky (označené jako bloky 1 - n) mají velikost 324 bajtů, jelikož obsahují pouze počítadlo bloků, data, klíč následujícího bloku a heš následujícího bloku, kromě posledního bloku (označeného jako blok n). Tento blok místo klíče a heše již obsahuje pouze výplň nul o velikosti těchto částí bloku.

Bloky bylo třeba tvořit reverzně, tedy od bloku n do bloku 0, jelikož každý blok obsahuje klíč a heš následujícího bloku. Poslední vytvořený klíč AES-256, který je vytvořen v bloku 1, je poté v bloku 0 zabalen dle doporučení RFC3394[1]. Toto rozdělení je zobrazené na obrázku 1.2.

[1]Konstrukce zabalení klíče (key wrap) slouží k zapouzdření klíčového materiálu. Je využíváno pro zajištění bezpečné přepravy klíče.

V generátoru je možné používat šest příkazů, provádějících specifické akce. Jedná se o příkazy – Erase, Load, ProgramFuses, ProgramIFR, Call a Execute. Informace k příkazům byly již uvedeny v sekci 1.3.2.

Generátor byl navržen tak, aby byl variabilní a bylo možné snadno změnit parametry, například velikost bloků, typ použité křivky atd.

Po dokončení generátoru bylo vytvořeno i uživatelsky přívětivé rozhraní. Cílem rozhraní bylo zjednodušit uživateli pochopení funkčnosti generátoru, bez nutnosti procházet veškeré kódy a snažit se jim porozumět či znát daný programovací jazyk. Uživateli stačí pouze zadat v rozhraní vstupy – cesty k certifikátům, cesty k datům, které má použít příkaz Load nebo ProgramIFR. Dále adresy, které příkazy potřebují pro specifikaci místa, kde mají akce vykonávat, případně délky a samozřejmě i hodnoty (slova), které má použít příkaz programFuses. Příkaz tyto hodnoty zapisuje do pojistek neboli fuses.

Uživatel zde může využít potřebných rozhodovacích vstupů jako například, zda chce či nechce použít ISK certifikát nebo kolik ze čtyř CTRK klíčů bude chtít použít. V případě, že zadané hodnoty nezmění, budou vykonány dle základního nastavení (bude použit ISK certifikát a čtyři CTRK klíče).

2.2 Programové řešení

Jak již bylo uvedeno výše, jako první bylo potřeba vytvořit blok 0, který tvoří několik částí. První vytvořenou částí byla hlavička bloku, ve které je použit již dříve zmíněný modul struct, označený jako FORMAT, čímž definujeme, jakého typu a velikosti jsou daná pole. FORMAT uchovává formát pro modul struct a je velmi důležitý pro dosažení správné funkčnosti funkcí `export` a `parse`. Jako příklad můžeme uvést string – MAGIC ("sbv3"), který identifikuje začátek hlavičky a má velikost 4 bajty, zde byl použit typ char. Dle modulu struct se pro typ char použije označení *s*, při velikosti 4 bajty se jedná o označení 4*s*. Dále je třeba specifikovat endian, který určuje pořadí bajtů (little endian <, big endian >). Jelikož se hlavička nešifruje, je možné vyčíst string MAGIC a dle něho ověřit na jakém místě se nachází začátek hlavičky obrazu či hlavička certifikačního bloku. Následně jsou inicializované položky, které hlavička obsahuje.

Hlavička bloku obsahuje funkce `export` a `parse`. Funkce `export` slouží k serializaci objektu a je zobrazena ve výpisu 2.1. Funkce ošetřuje výjimku, která by mohla vzniknout, pokud by objekt nebyl typu BackendClass[2]. V případě, že je objekt správného typu vrátí se True, jinak se vrátí False a je vyvolána výjimka. Pokud

[2]BackendClass je rodičem kryptografických backendů backend_internal a backend_openssl, čím nahrazuje zápis všech těchto backendů a usnadňuje případné přidání dalších backendů, které jsou využívány pro kryptografii.

nenastane výjimka (tedy vrátí se True), tak je provedeno rozdělení verze hlavičky a verze firmwaru na major a minor, za pomoci metody `split`.

Poté je vrácen bajtový objekt, obsahující hodnoty, které mají být zapsány do hlavičky, včetně rozdělené verze a firmware verze (v souladu s formátem řetězce). Funkce `export` nejdříve spojuje objekt obsahující výše zmíněné parametry hlavičky do jednoho celku a následně jej převádí do formy bajtového pole, a to za použití funkce `pack`.

Výpis 2.1: `Funkce export hlavičky bloku.`

```
def export(self, **kwargs: BackendClass) -> bytes:
    backend = kwargs.get("backend")
    if not isinstance(backend, BackendClass):
        raise Exception("Backend is not included!")

    major_version, minor_version = [int(v) for v in
    self.version.split(".")]
    major_firmware_version, minor_firmware_version = [int(v) for v
    in self.firmware_version.split(".")]

    return pack(self.FORMAT,
                self.MAGIC,
                minor_version, major_version,
                self.flags,
                self.block_count,
                self.block_size,
                self.timestamp,
                major_firmware_version, minor_firmware_version,
                self.image_total_length,
                self.image_type,
                self.certificate_block_offset
                )
```

Funkce `parse` je zobrazena ve výpisu 2.2 a slouží k deserializaci objektu. Kontroluje se zde, zda velikost dat bez odsazení (offsetu) je rovna předpokládané velikosti dat. K tomuto je využívána funkce `calcsize`, která vrací velikost struktury v bajtech. V případě, že nesplňuje tuto podmínku, je vyvolána výjimka. V opačném případě jsou provedeny zbývající instrukce ve funkci.

Z dočasné paměti jsou rozbalena data za pomoci funkce `unpack_from`, přičemž rozbalování začíná na pozici dané zmíněným odsazením. Rozbalená data jsou následně zapsána do nové proměnné, reprezentující objekt typu ImageHeaderV3, který tvoří návratovou hodnotu funkce.

Výpis 2.2: `Funkce parse hlavičky bloku.`

```
    def parse(cls, data: bytes, offset: int = 0) -> "ImageHeaderV3":
        if cls.SIZE > len(data) - offset:
            raise Exception("SIZE is bigger than length of the data
                without offset!")
        (
            magic,
            major_version, minor_version,
            flags,
            block_count,
            block_size,
            timestamp,
            major_firmware_version, minor_firmware_version,
            image_total_length,
            image_type,
            certificate_block_offset
        ) = unpack_from(cls.FORMAT, data, offset)
        ...
        return obj
```

Následně byl tvořen certifikační blok, který se skládá z hlavičky certifikačního bloku (CertBlockHeaderV3), dále CTRK tabulky (CtrkTable) a ISK certifikátu (IskCertificateV3).

Hlavička certifikačního bloku, opět používá modul struct, s označením FORMAT a poté inicializuje položky, které hlavička obsahuje. Hlavička také obsahuje funkci `export`, která rozděluje verze, poté spojuje objekt obsahující parametry hlavičky a následně převádí do formy bajtového pole. Funkce `parse`, která z dočasné paměti rozbaluje data, následně zapisuje data do nové proměnné, reprezentující objekt typu CertBlockHeaderV3, který tvoří návratovou hodnotu funkce. Toto je tvořeno obdobným způsobem jako u hlavičky bloku.

Jako další byla vytvořena CTRK tabulka, která obsahuje pouze veřejné klíče. Je zobrazena ve výpisu 2.3. Uživatel má možnost volit, kolik klíčů bude použit (lze zvolit 1 až 4 veřejné klíče). Byl zde vytvořen parametr MAX_COORDINATE_SIZE, který zajišťuje větší variabilitu. Parametr byl nastaven na hodnotu 66 bajtů.

Funkce `export` obsahuje data z certifikátů, které si uživatel sám zvolí. Při zadávání dat z certifikátů by mohlo dojít k chybě, při které by se použila jedna či více křivek jiného typu. Z tohoto důvodu je zde ošetřena výjimka, která ověřuje, zda jsou všechny křivky stejného typu a pokud by stejného typu nebyly, došlo by k vyvolání výjimky. V opačném případě jsou provedeny následující instrukce. Funkce `export` provede metodu import_key, která vrací souřadnice křivky x a y v bajtovém poli.

Jelikož je v generátoru možné využívat více křivek – secp256r1, secp384r1 nebo secp512r1, byly vytvořeny koordináty včetně výplně nul, aby nebylo třeba dalších

zásahů, pokud by byla využita jiná než vzorová křivka secp256r1. Toto bylo provedeno z toho důvodu, že každá křivka má souřadnice o jiných velikostech a maximální velikost by při fixní velikosti mohla být překročena. Například zde se používá křivka secp256r1 a velikost jedné její souřadnice je 32 bajtů (souřadnice x a y, každá o velikosti 32 bajtů, tedy 64 bajtů celkem).

Výpis 2.3: `CTRK tabulka.`

```
    def export(self, key1: bytes, key2: bytes, key3: bytes,
               key4: bytes, max_coordinate_size: int = 66) -> bytes:

        keys = [key1, key2, key3, key4]
        data = b""
        ecc_key1 = ECC.import_key(key1)
        curve = ecc_key1.curve
        for key in keys:
            if key is not None:
                ecc_key = ECC.import_key(key)
                if ecc_key.curve != curve:
                    raise Exception("Invalid curve in one of the loaded
                        files! All curves must be of the same type.")

        for key in keys:
            if key is not None:
                byte_key = b""
                key = ECC.import_key(key)
                ctrk_point_x = key.pointQ.x.to_bytes()
                ctrk_point_y = key.pointQ.y.to_bytes()
                byte_key += add_zero_padding_ctrk(ctrk_point_x,
                            max_coordinate_size) + \
                             add_zero_padding_ctrk(ctrk_point_y,
                             max_coordinate_size)
            else:
                byte_key = bytes("\x00" * 132, "utf8")
            data += byte_key

        return data
```

ISK certifikát je tvořený ISK verzí (constraints), veřejným klíčem ISK a podpisem ISK.

Podobně jako v CTRK tabulce, jsou ve funkci `export` z certifikátu načteny souřadnice použitých bodů v souvislosti s eliptickou křivkou, v našem případě se opět jedná o křivku secp256r1. Je nutné nejprve získat veřejný klíč, zde je opět velikost jedné souřadnice 32 bajtů (souřadnice x a y). U soukromého klíče je velikost jedné souřadnice také 32 bajtů (souřadnice r a s). Návratovou hodnotu této funkce tvoří bajtové pole, které se skládá z výše uvedených částí.

Po vytvoření částí certifikačního bloku bylo třeba ještě vyřešit možnosti volby počtu CTRK klíčů, proto za pomoci rozhodovací konstrukce if/elif/else je rozhodováno o použití jednoho až čtyř CTRK klíčů. Je zde ošetřeno použití špatné volby, a to vyvoláním výjimky, která uživatele informuje o tom, že je třeba volit pouze čísla od 0 do 3.

Poté funkce `create_cert_block_v3()`, spojuje výstupy jednotlivých funkcí tříd, které jsou reprezentovány třemi proměnnými (header, ctrk_table a isk_certificate), přičemž každá z nich obsahuje bajty. Tyto výstupy jsou poté spojeny a následně funkce vrací již jako certifikační blok. Tvorba certifikačního bloku je zobrazena ve výpisu 2.4.

Výpis 2.4: `Tvorba certifikačního bloku.`

```
    def create_cert_block_v3(self,
                             header: bytes = None,
                             ctrk_table: bytes = None,
                             isk_certificate: bytes = None) -> bytes:
        if header is None:
            header = self._header
        if ctrk_table is None:
            ctrk_table = self._ctrk_table
        if isk_certificate is None:
            isk_certificate = self._isk_certificate
        certificate_block_v3 = header + ctrk_table + isk_certificate
        return certificate_block_v3
```

Součástí generátoru jsou příkazy neboli cmd, které jsou určeny k vykonávání několika akcí, a to především během nahrávání firmwaru. Jedná se konkrétně o šest příkazů: Erase, Load, Execute, Call, programFuses, programIFR. Některé z informací k těmto příkazům byly již uvedeny v sekci 1.3.2.

Nejprve byla vytvořena hlavička příkazů, ve které je opět použit modul struct jako FORMAT. Dále jsou inicializována pole, které hlavička příkazů obsahuje. Poté jsou provedeny funkce `export` a `parse`, které již byly popsány výše. Následují příkazy, kde jsou opět inicializována pole, které příkazy obsahují. Hlavička příkazů, která je zobrazena ve výpisu 2.5 slouží k inicializaci nové hlavičky o určité délce na předem dané adrese v paměti. Je zde funkce určená pro export dat hlavičky a funkce pro parsování, kde jsou ošetřeny výjimky – kontrola velikosti dat a štítku, v kódu označeného jako TAG.

Výpis 2.5: `Funkce export a parse hlavičky příkazů.`

```
    def export(self) -> bytes:
        return pack(self.FORMAT, self.TAG, self.address, self.length,
                    self.cmd)

    def parse(cls, data: bytes, offset: int = 0) -> "CmdHeaderNew":
        if calcsize(cls.FORMAT) > len(data) - offset:
            raise Exception("FORMAT is bigger than length of the data
                without offset")
        obj = cls()
        (tag, obj.address, obj.length, obj.cmd) = unpack_from(cls.
            FORMAT, data, offset)
        if tag != cls.TAG:
            raise Exception("TAG is not valid.")
        return obj
```

Příkaz Load, který je zobrazen ve výpisu 2.6, slouží pro nahrávání firmwaru a kontrolu, zda se nepohybujeme mimo rozsah paměti a zda jsou nahrávaná data ve formě bajtů. Při exportu aktualizujeme velikost – délku dat a přidáváme prefix x00. Za pomoci funkce `parse` je možné přeformátovat data a případně pozměnit odsazení či velikost dat. Je zde třeba zadat adresu a data, která mají být použita k nahrání. Příkaz je označen štítkem – LOAD.

Výpis 2.6: `Funkce export a parse příkazu Load.`

```
    def export(self) -> bytes:
        data = bytes(self._data)
        if len(data) % 4:
            data += b"\x00" * (4 - (len(data) % 4))
        # update header
        self._header.length = len(data)
        # export
        final_data = self._header.export()
        final_data += data
        return final_data

    def parse(cls, data: bytes, offset: int = 0) -> "CmdLoadNew":
        header = CmdHeaderNew.parse(data, offset)
        if header.cmd != EnumCmdTag.LOAD:
            raise Exception("Values are not same.")
        offset += CmdHeaderNew.SIZE
        load_data = data[offset: offset + header.length]
        return cls(header.address, load_data)
```

Příkaz Erase, jak již prozrazuje jeho název, slouží k mazání hlavičky v paměti. Aby se předešlo tomu, že bude zadáno mazání mimo rozsah paměti, kontroluje příkaz Erase, jestli se zadaný rozsah v paměti skutečně nachází. Za pomoci funkce `parse` je opět možné přeformátovat data a případně pozměnit odsazení či velikost dat. Je třeba zadat danou adresu a rozsah. Příkaz je označen štítkem – ERASE.

Příkaz Execute zajišťuje nastavení počáteční adresy do konkrétního umístění v RAM a následně za pomoci tohoto příkazu začíná vykonávat aplikaci z této adresy. Je třeba zadat danou adresu. Příkaz je označena štítkem – EXECUTE.

Příkaz Call signalizuje volání v paměti. Příkaz zajišťuje nastavení počáteční adresy do konkrétního umístění v RAM a poté očekává návrat a je pokračováno ve zpracovávání souboru SB3. Je třeba zadat danou adresu a rozsah. Příkaz je označen štítkem – CALL.

Příkaz programFuses zajišťuje nastavení počáteční adresy do registru pojistek neboli fuses. U tohoto příkazu je třeba zadat nejen adresu, ale i hodnoty. Tyto hodnoty se zapisují do registrů pojistek, což lze provést pouze jednou, poté již hodnoty změnit nelze. Příkaz je označen štítkem – PROGRAM_FUSES.

Příkaz programIFR zajišťuje nastavení počáteční adresy do oblasti IFR. Oblast IFR je oblast paměti, kterou je možné nastavovat chování ROM. Je zde třeba zadat adresu a data, která mají být použita k zápisu do oblasti IFR. Příkaz je označen štítkem – PROGRAM_IFR.

Příkazy opět využívají funkci `parse`, která umožňuje přeformátování dat a případně pozměnění odsazení či velikosti dat.

Po vytvoření příkazů byla vytvořena funkce `add_command`, která je zobrazená ve výpisu 2.7. Tato funkce slouží k tvorbě kolekce příkazů, které se využívají pro start a aktualizaci firmwaru.

Za pomoci cyklu for a metody append jsou všechny příkazy postupně zapisovány na nakonec listu a je vytvořena kolekce všech příkazů, které může uživatel používat. Jakmile je kolekce vytvořena, je ještě do kolekce přidána hlavička sekce. Jelikož hlavička obsahuje délku kolekce příkazů, je nutné přidat ji do kolekce až jako poslední.

Výpis 2.7: `Funkce pro tvorbu kolekce příkazů v generátoru SB3.`

```
    def add_command(self, export: bool = False, **kwargs: object)
        -> None:
        if export:
            collection_of_commands = bytes()
            for command in self._commands:
                collection_of_commands += command.export()
            section_header = CmdSectionHeader(section_uid=1,
                                              section_type=1,
                                    length=len(collection_of_commands))
            collection_of_commands = section_header.export() +
                                     collection_of_commands

            self._commands = collection_of_commands
        else:
            self._commands.append(kwargs.get("cmd"))
```

Následně byly tvořeny již samotné bloky. Tyto bloky jsou odlišné od bloku 0, který se používá pro podepisování právě těchto bloků. Jelikož bloky obsahují heš následujícího bloku a klíč následujícího bloku, bylo nutné je tvořit v opačném pořadí. První je tedy tvořen poslední blok, který obsahuje data a výplň nul, nahrazující klíč a heš, které tento blok již neobsahuje. Poté jsou tvořeny další bloky, které již klíč a heš obsahují.

Data jsou rozdělena na části po 256 bajtech. Zbylá data, která již nedosahují této velikosti, jsou doplněna o výplň nul. Je vygenerován náhodný klíč a spolu s první částí dat se vytvoří data pro šifrování. Tato data jsou pak šifrována algoritmem AES, který využívá operační mód CBC. Provede se funkce `aes_cbc_encrypt`. Blok následně rozdělíme a včetně počítače bloků a 32 bajtů nul zahešujeme. Tyto bajty nahrazují heš následujícího bloku, který v posledním bloku není k dispozici. V dalších blocích je již používán heš následujícího bloku.

Každý vytvořený blok je vždy vkládán na začátek listu za pomoci metody insert, aby výsledkem byla již správně seřazená kolekce bloků. Funkce pro tvorbu bloků je zobrazena ve výpisu 2.8.

Výpis 2.8: Funkce pro tvorbu bloků v generátoru SB3.

```
    def create_blocks(self) -> bytes:
        ...
        for block in reversed(collection_of_blocks):
            if last_block:
                block = add_leading_zeros(block, DATA_SIZE)
                encryption_key = internal_backend.random_bytes(32)
                data_to_encrypt = block + encryption_key
                encrypted_data = internal_backend.aes_cbc_encrypt(
                                 encryption_key, data_to_encrypt)
                encrypted_block = encrypted_data[:256]
                encrypted_aes_key = encrypted_data[256:]
                data_to_hash = pack("<L", block_counter) +
                     encrypted_data + encrypted_aes_key + bytes(32)
                block_hash = internal_backend.hash(data_to_hash,
                             algorithm="sha256")
                current_block = Sb3Block(data=encrypted_block,
                   encryption_key_of_next_block=encrypted_aes_key,
                   hash_of_next_block=bytes(32 * "\x00", "utf-8"),
                                        block_number=block_counter)
                exported_blocks.insert(0, current_block.export())
                last_block = False
            else:
                encryption_key = internal_backend.random_bytes(32)
                data_to_encrypt = block + encryption_key
                encrypted_data = internal_backend.aes_cbc_encrypt(
                                 encryption_key, data_to_encrypt)
                encrypted_block = encrypted_data[:256]
                encrypted_aes_key = encrypted_data[256:]
                hash_of_next_block = internal_backend.hash(
                           exported_blocks[0], algorithm="sha256")
                current_block = Sb3Block(data=encrypted_block,
                   encryption_key_of_next_block=encrypted_aes_key,
                               hash_of_next_block=hash_of_next_block,
                                         block_number=block_counter)
                exported_blocks.insert(0, current_block.export())
            block_counter -= 1
        return b"".join(exported_blocks)
```

Po vytvoření bloků, jsme získali klíč bloku 1, který potřebujeme pro blok 0. Bylo tedy možné přistoupit k zabalení klíče neboli tvorbě wrapu. Funkce `create_wrap`, slouží právě k zabalení tohoto klíče a zajišťuje bezpečný přenos. Zabalení probíhá dle doporučení RFC3394 a je tomu třeba použít soubor s uživatelským klíčem a klíč následujícího bloku. Poté již probíhá zabalení klíče.

Poté byla vytvořena funkce `create_signature`, která je zobrazená ve výpisu 2.9. Tato funkce slouží k tvorbě podpisu obrazu. Je zde použit soukromý klíč, algoritmus pro podepisování – ECDSA a hešovací funkce – SHA-512. Velikost jedné souřadnice je opět 32 bajtů (r a s).

Výpis 2.9: `Funkce pro tvorbu podpisu bloku v generátoru SB3.`

```
def create_signature(private_key: bytes, data: bytes) -> bytes:
    private_key_obj = ECC.import_key(private_key)
    signature = internal_backend.ecdsa_sign(private_key_obj, data,
                algorithm="sha512")
    half_size = int(len(signature) / 2)
    point_r = signature[:half_size]
    point_s = signature[half_size:]
    signature = add_zero_padding(point_r, MAX_COORDINATE_SIZE) +
                add_zero_padding(point_s, MAX_COORDINATE_SIZE)
    return signature
```

Jelikož již byly vytvořené veškeré potřebné části bylo možné vytvořit blok 0. Nejprve jsou inicializované položky, které blok obsahuje a poté za pomoci funkce `export` jsou tyto parametry bloku spojeny do jednoho celku a následně převedeny do formy bajtového pole za použití funkce `pack`.

Jakmile byl dokončen funkční generátor bylo možné začít tvořit rozhraní pro uživatele. Jak již bylo zmíněno na v sekci 2.1, cílem tohoto rozhraní mělo být poskytnutí prostředí uživateli, který bez toho, aniž by musel procházet veškeré kódy, které generátor obsahuje nebo měl rozsáhlé znalosti programovacího jazyka Python, mohl snadno pochopit, jak generátor funguje a snadno jej spustil.

Vytvořené rozhraní má umožnit uživateli zadat cesty k certifikátům a rozhodnout se, zda bude chtít použít ISK certifikát. Jelikož je tedy ISK certifikát volitelný, stačí uživateli pouze změnit use_isk z True na False a ISK certifikát nebude dále použit. Stejným způsobem si uživatel volí počet CTRK klíčů. Zde si může snadno zvolit, zda použije pouze jeden, dva, tři či všechny čtyři CTRK klíče, a to změnou číslice. K tomuto rozhodnutí má uživatel nápovědu ve formě docstringu, kde má uvedeny veškeré možné volby, tedy že pro použití jednoho klíče volí číslici 0, dvou klíčů číslici 1, tří klíčů číslici 2 a v případě použití čtyř klíčů volí číslici 3. Dále se již pracuje s touto hodnotou. Také je zde možné zvolit, zda bude přidána speciální hlavička, kterou mají pouze NXP soubory. Toto je provedeno při změně hodnoty is_nxp z False na True. Ukázka docstringu je zobrazena ve výpisu 2.10.

Výpis 2.10: Ukázka popisu voleb.

```
    """Boot image generator.

    Use 1 - 4 certificates based on used_root_cert parameter value
    in range [0,3], e.g. value 0 equals to using one certificate,
    1 using two certificates, etc.
    Use isk certificate if use_isk is set to True.
    Use NXP container if is_nxp is set to True.
    """
```

Jako poslední položka, kterou si uživatel může změnit jsou příkazy. Zde uživatel zadává adresy, délky, hodnoty a může si nahrát svá data.

Rozhraní poté již vykoná vše potřebné – vytvoří klíče, certifikáty, heše následujících bloků, klíče následujících bloků, vytvoří kolekci příkazů, vytvoří bloky a vykoná funkci main.

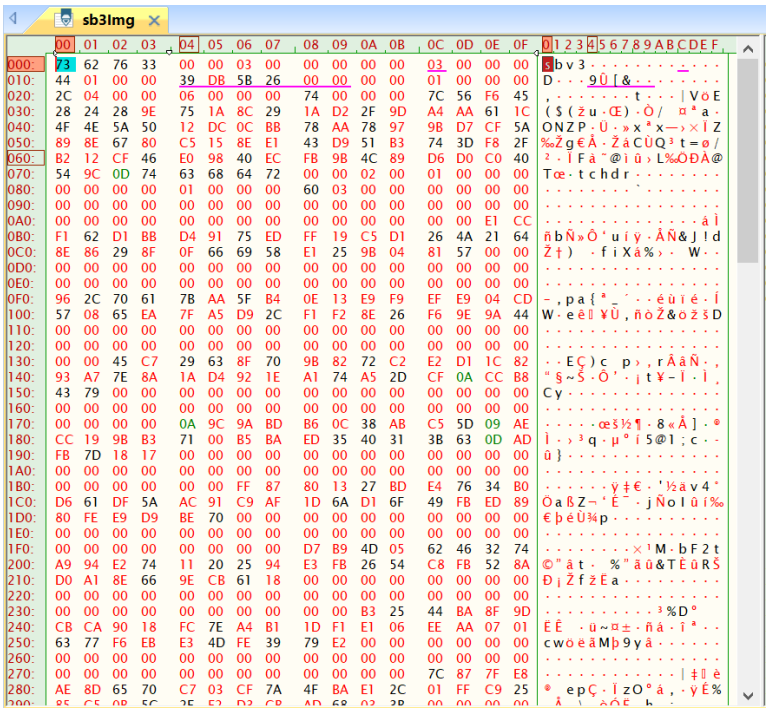
Aby bylo možné porovnat správný výstup generátoru a výstup rozhraní, které jsou v hexadecimální soustavě, byl při tvorbě rozhraní využíván nástroj HexEdit. Takto bylo například zjištěno že v souboru example_output.sb3 se v jedenáctém bajtu nachází hodnota nula a tedy, že neodpovídá hodnota počtu bloků. Tímto bylo zjištěno, že není správně prováděna funkce pro výpočet počtu bloků a bylo možné provést opravu. Což bylo vyřešeno vytvořením funkce `get_number_of_blocks` v souboru image.py. Tato funkce je zobrazena ve výpisu 2.11.

Výpis 2.11: Funkce pro výpočet počtu bloků SB3.

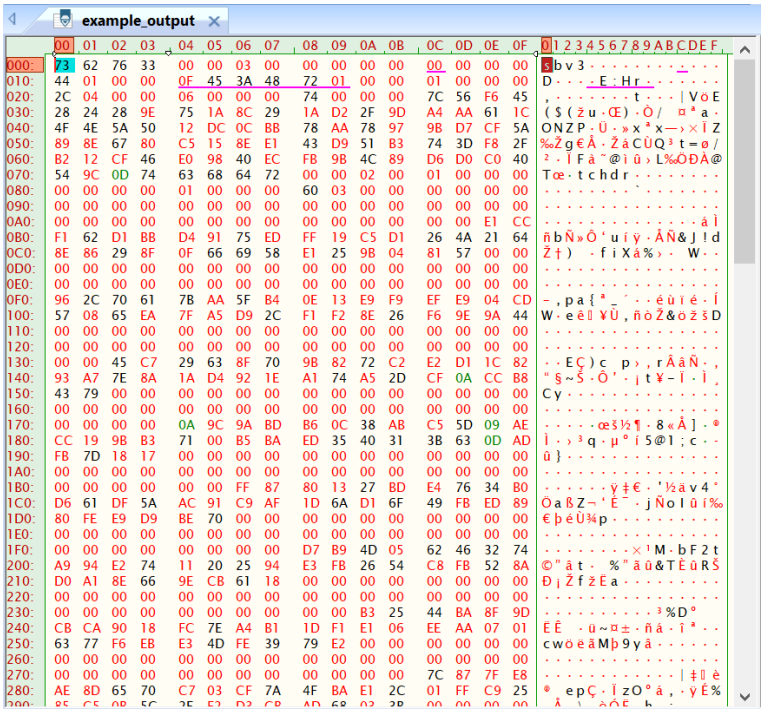
```
def get_number_of_blocks(commands) -> int:
    collection_of_blocks = [commands[i:i + DATA_SIZE] for i in
                            range(0, len(commands), DATA_SIZE)]
    return len(collection_of_blocks)
```

Porovnání výstupu generátoru (sb3Img.sb3) je zobrazeno na obrázku 2.1 a porovnání výstupu vytvořeného rozhraní (example_output.sb3) je zobrazeno na obrázku 2.2. Můžeme zde například vidět string – MAGIC, konkrétně se jedná o slovo "sbv3", které se nachází na první řádce a označuje začátek hlavičky obrazu a slovo "chdr", které se nachází na sedmé řádce a označuje začátek hlavičky certifikátu.

Podtržené části označují rozdíly v porovnávaných výstupech. Veškeré nesoulady mohly být odstraněny díky využití vzorových certifikátů, klíčů a dat, která některé příkazy používají. Nebylo možné ověřit pouze ty části, které používají náhodně vygenerovaná čísla. Proto byla potřeba provést úprava. Jako příklad můžeme uvést klíč, který se používá při šifrování bloků. Standardně se použije náhodně vygenerované číslo, ale pro otestování bylo použito 32 bajtů nul.



Obr. 2.1: Porovnání výstupu generátoru (soubor sb3Img.sb3).



Obr. 2.2: Porovnání výstupu rozhraní (example_output.sb3).

V ukázce je také možné vidět rozdíl v časovém razítku. Jelikož byl každý soubor vytvořen v jinou dobu, jsou časová razítka odlišná. Aby bylo možné ověřit správnou funkčnost, bylo nastaveno časové razítko pro oba soubory stejné.

Pro účely testování zde používáme certifikáty s formátem PEM (Privacy-enhanced Electronic Mail). Formát PEM se využívá z toho důvodu, že obsah souboru je uchováván ve struktuře, kterou popisuje ASN.1 a je serializován kódováním DER. A jak již bylo zmíněno v sekci 1.1, pro formáty DER je použita knihovna asn1crypto. Jako křivku jsme pro testování zvolili secp256r1. Ukázka certifikátu PEM, který byl používán jako vzor je zobrazena na obrázku 2.3.

```
-----BEGIN CERTIFICATE-----
MIICQzCCAemgAwIBAgIJANcSn2v73YZUMAoGCCqGSM49BAMCMH4xCzAJBgNVBAYT
AkNaMRcwFQYDVQQIDA5DemVjaCBSZXB1YmxpYzEPMA0GA1UEBwwGUm96bm92MQww
CgYDVQQKDANOWFAxEzARBgNVBAMMCkx1a3MgWmFqYWMxIjAgBgkqhkiG9w0BCQEW
E2x1a2FzLnphamFjQG54cC5jb20wHhcNMTgxMTA2MDcxNzA4WhcNMTkxMTA2MDcx
NzA4WjB+MQswCQYDVQQGEwJDWjEXMBUGA1UECAwOQ3plY2ggUmVwdWJsaWMxDzAN
BgNVBAcMBlJvem5vdjEMMAoGA1UECgwDTlhQMRMwEQYDVQQDDApMdWtzIFphamFj
MSIwIAYJKoZIhvcNAQkBFhNsdWthcy56YWphY0BueHAuY29tMFkwEwYHKoZIzj0C
AQYIKoZIzj0DAQcDQgAE4czxYtG71JF17f8ZxdEmSiFkjoYpjw9maVjhJZsEgVeW
LHBhe6pftA4T6fnv6QTNVwhl6n+l2Szx8o4m9p6aRKNQME4wHQYDVR0OBBYEFHyB
9SmSNqiL9ZyDHlZwiq+nRyk4MB8GA1UdIwQYMBaAFHyB9SmSNqiL9ZyDHlZwiq+n
Ryk4MAwGA1UdEwQFMAMBAf8wCgYIKoZIzj0EAwIDSAAwRQIgODNB545FUPJANwvf
TtccxBZovTAebtprmXLLGrSz2tICIQCGADlil8qOoe2qed3RK/NWCT2dGN7oSK5m
6yEV0TI+3g==
-----END CERTIFICATE-----
```

Obr. 2.3: Certifikát PEM.

V testu používáme všechny čtyři klíče z CTRK tabulky a ISK certifikát. Testovací soubory, které používají příkazy Load a programIFR jsou v binární podobě. Také adresy, délky a hodnoty byly nastaveny vzorově. Vzorové hodnoty jsou zobrazeny na obrázku 2.4.

```
sb3.timestamp = 643554105
sb3.add_command(cmd=CmdEraseNew(address=0x1384, length=0xFFFF))
sb3.add_command(cmd=CmdLoadNew(address=0x1256, data=loaded_data))
sb3.add_command(cmd=CmdProgFusesNew(0x138498, 0x0, 0x5, 0x1ab, 0x1ab, 0xffffffff, 0xfffffff1, address=0x1384))
sb3.add_command(cmd=CmdProgIfrNew(address=0x1384, data=ifr_data))
sb3.add_command(cmd=CmdCallNew(address=0x1384))
sb3.add_command(cmd=CmdExecuteNew(address=0x1384))
```

Obr. 2.4: Vzorové hodnoty.

V průběhu tvorby generátoru byly tvořeny testy, které měly ověřit funkčnost dílčích částí generátoru. Veškeré testy proběhly úspěšně. Jako příklad můžeme uvézt test tvorby certifikačního bloku, který testuje velikost hlavičky certifikačního bloku, CTRK tabulky a ISK certifikátu. Test je zobrazen na obrázku 2.12.

Výpis 2.12: Test tvorby certifikačního bloku.

```
    def generate_certificate_block(data_dir):
        cert_block = CertBlockV3(
                used_root_cert=0,
                use_isk=True,
                root_cert1=read_file(data_dir, "ec_secp256r1_cert0.pem"),
                root_cert2=read_file(data_dir, "ec_secp256r1_cert1.pem"),
                root_cert3=read_file(data_dir, "ec_secp256r1_cert2.pem"),
                root_cert4=read_file(data_dir, "ec_secp256r1_cert3.pem"),
                signing_public_key=read_file(data_dir,
                "ec_secp256r1_sign_cert.pem"),
                signing_private_key=read_file(data_dir,
                "ec_pk_secp256r1_sign_cert.pem"),
                constraints=0
                )
    return cert_block

    def test_cert_block_v3(data_dir):
    cert_block = generate_certificate_block(data_dir)

    assert len(cert_block._header) == CertBlockHeaderV3.SIZE
    assert len(cert_block._ctrk_table) == CtrkTableV3.SIZE
    assert len(cert_block._isk_certificate) == IskCertificateV3.SIZE
```

Na obrázku 2.5 je zobrazen úspěšný výstup provedeného testu tvorby certifikačního bloku.

```
============================= test session starts =============================
platform win32 -- Python 3.7.5, pytest-5.3.5, py-1.8.1, pluggy-0.13.1 -- C:\Git\bsdk\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Git\bsdk, inifile: pytest.ini
plugins: cov-2.8.1
collecting ... collected 1 item

test_sections_api.py::test_cert_block_v3 PASSED                          [100%]

============================== 1 passed in 1.88s ==============================

Process finished with exit code 0
```

Obr. 2.5: Úspěšně provedený test tvorby certifikačního bloku.

Závěrem této práce byla vytvořena i uživatelská dokumentace, která je přiložena v příloze číslo A.

2.3 Zhodnocení SB3 a doporučení

Aktuálně se nevyrábí žádný embedded mikrokontroler, na kterém by bylo možné soubor SB3 používaný pro bezpečný start a aktualizaci firmware využít či otestovat. Předpokládá se, že soubor SB3 bude na mikrokontrolery nasazen v lednu 2021 a následně bude na mikrokontrolerech využíván uživateli.

Je očekávána životnost generátoru souboru SB3 20 let. Pro tento závěr byly brány v potaz doporučení dle doporučení délek klíčů a srovnání generátoru s doporučeními národního institutu standardů a technologií NIST (National Institute of Standards and Technology) [16].

V souboru SB3 je používána hešovací funkce SHA-256. Toto zabezpečení je aktuálně dostačující. Pokud bychom zvažovali vylepšení, bylo by možné provést využitím hešovací funkce SHA-512, která má výstupní heš větší délky a bude odolnější proti útoku hrubou silou (Brute-Force Attack).

Použití blokové šifry AES s klíčem o délce 256 bitů můžeme považovat aktuálně za dostatečné, jelikož dosud nejsou známy žádné metody, které by vedly k prolomení klíče ani u šifry AES s klíčem o délce 128 bitů, a to za pomoci útoku hrubou silou (Brute-Force Attack).

Dostatečná bezpečnost je prokázána i použitím módu CBC. Jelikož je zde generován nový náhodný klíč pro každý blok, není tedy možné provést útok prostého textu (Chosen-Plaintext Attack).

Pro podepisování je v souboru SB3 používán algoritmus ECDSA a je možné volit křivky secp256r1, secp384r1 a secp521r1. Dle doporučení národního institutu NIST je při použití algoritmu AES-256 doporučeno použití eliptické křivky o velikosti 256 bajtů [16]. Požadavky na zabezpečení jsou tedy splněny, ovšem pro vyšší bezpečnost by bylo vhodné volit křivky větších velikostí. Jelikož jsou všechny tři křivky považovány za dostatečně bezpečné, není třeba v tomto směru provádět žádné vylepšení. Avšak jako vhodné vylepšení můžeme považovat to, že do docstringu v již vytvořeném rozhraní umístíme doporučení o vhodnosti používat křivky s delšími klíči.

Závěr

V této práci byl vytvořen generátor, který vytváří soubor SB3, sloužící pro zabezpečený start a aktualizaci firmwaru na embedded mikrokontrolerech s jádry ARM Cortex-M33.

Vytvořený generátor provádí sestavení jednotlivých bloků ve formě bajtových polí. Základním „stavebním“ prvkem, který aplikace vytváří a následně používá je nultý blok. Tento blok obsahuje množinu dílčích bloků se specifickým účelem, a to podepisovat další bloky. Skládá se z hlavičky obrazu, klíče AES-256 následujícího bloku, heše SHA-256 následujícího bloku, certifikačního bloku a podpisu obrazu. Díky tomuto systému je generován nový IV a tím je zabráněno útoku prostého textu (Chosen-Plaintext Attack).

Certifikační blok, kromě hlavičky obsahuje i CTRK tabulku, která obsahuje veřejné klíče root certifikátu a ISK certifikát, který obsahuje verzi certifikátu, veřejný klíč podepisovaného certifikátu a podpis podepisujícího certifikátu z vybraného root certifikátu z CTRK tabulky.

Je zde možnost zvolit, zda bude použit klíč ISK, a to pouze změnou hodnoty z True (ISK je použit) na False (ISK není použit) či naopak. Dále je možné volit počet použitých root klíčů, které jsou obsaženy v CTRK tabulce. Změna je provedena volbou číslic od nuly do tří (číslice 0 značí volbu jednoho klíče, číslice 3 označí volbu čtyř klíčů). Neplatná volba je ošetřena vyvoláním výjimky.

Pro vykonávání různých akcí, lze v datovém rozsahu použít některý z šesti příkazů – Erase (vymazává flash paměť daného adresního rozsahu), Load (umožňuje zápis zadaných dat), Execute (nastavuje počáteční adresu na konkrétní umístění), Call (nastavuje počáteční adresu na konkrétní umístění v RAM), programFuses (nastavuje počáteční adresu na konkrétní umístění v registru pojistek a zapisuje do pojistek zadané hodnoty) a programIFR (nastavuje počáteční adresu na konkrétní umístění v IFR).

Dále již jen stačí místo vzorových dat vložit data obrazu, ta se rozdělí do bloků a spojí s výše uvedenými částmi a vytvoří blok 0 až blok n. Počet bloků je závislý na velikosti dat. Za pomoci příkazů se vykonají požadované akce.

Bylo vytvořeno i uživatelské rozhraní, které umožňuje snadno pochopit funkcionalitu, aniž by musel uživatel znát programovací jazyk Python a jeho knihovny nebo procházel veškeré kódy.

Pokud by uživatel nechtěl používat výchozí kryptografickou knihovnu, stačí mu již jen použít svoji knihovnu, například OpenSSL. Tato knihovna má například svoji vlastní funkci pro generování náhodných čísel.

Navržený generátor souboru SB3 a uživatelské rozhraní byly úspěšně otestovány. Testování bylo provedeno za pomoci knihovny pytest. Dále bylo provedeno porovnání

výstupů za pomoci editoru HexEdit a můžeme generátor považovat za funkční.

Můžeme tedy konstatovat, že byl vytvořen generátor, který sestavuje šifrované binární soubory, které jsou potřebné pro zabezpečený start a aktualizaci firmwaru, který bude moci uživatel používat na mikrokontroleru.

Literatura

[1] *Getting Started with the MCU Flashloader* [online]. 2018, [cit. 15. 12. 2019]. Dostupné z URL: <https://www.nxp.com/docs/en/user-guide/MBOOTFLASHGS.pdf>

[2] NXP Semiconductors, vnitřní dokumentace. *BSDK - MCU Secure Boot SDK* 2019, poslední aktualizace 22. 11. 2019 [cit. 25. 11. 2019].

[3] *LPC5500 Series* [online]. 2019, [cit. 22. 11. 2019]. Dostupné z URL: <https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc5500-cortex-m33:LPC5500_SERIES>

[4] *Intrinsic ID SRAM PUF Technology* [online]. 2019, [cit. 02. 12. 2019]. Dostupné z URL: <https://www.intrinsic-id.com/sram-puf/>

[5] *TrustZone technology for Armv8-M* [online]. 2019, [cit. 02. 12. 2019]. Dostupné z URL: <https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-for-cortex-m>

[6] NXP Semiconductors, vnitřní dokumentace. *BS3 Design* 2019, poslední aktualizace 5. 11. 2019 [cit. 06. 11. 2019].

[7] *The Memory Protection Unit* [online]. 2019, [cit. 03. 12. 2019]. Dostupné z URL: <https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-for-cortex-m>

[8] *pyEnum* [online]. 2019, [cit. 09. 12. 2019]. Dostupné z URL: <https://github.com/molejar/pyEnum>

[9] *struct — Interpret bytes as packed binary data* [online]. 2019, [cit. 17. 12. 2019]. Dostupné z URL: <https://docs.python.org/3/library/struct.html>

[10] *cryptography* [online]. 2017, [cit. 09. 12. 2019]. Dostupné z URL: <https://cryptography.io/en/latest/>

[11] *asn1crypto* [online]. 2017, [cit. 09. 12. 2019]. Dostupné z URL: <https://github.com/wbond/asn1crypto>

[12] *PyCryptodome's documentation* [online]. 2017, [cit. 09. 12. 2019]. Dostupné z URL: <https://www.pycryptodome.org/en/latest/>

[13] *Standards for Efficient Cryptography Group* [online]. 2010, [cit. 18. 04. 2020]. Dostupné z URL: <https://www.secg.org/>

[14] *Recommendation for Key Management* [online]. 2020, [cit. 25. 05. 2020]. Dostupné z URL: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf/>

[15] *Elftosb User's Guide* [online]. 2018, [cit. 15. 12. 2019]. Dostupné z URL: <https://www.nxp.com/docs/en/userguide/MBOOTELFTOSBUG.pdf>

[16] *Cryptographic Key Length Recommendation* [online]. 2020, [cit. 02. 05. 2020]. Dostupné z URL: <https://www.keylength.com/en/>

Seznam symbolů, veličin a zkratek

AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CPU	Central Processing Unit
CTRK	Customer Trusted Root Key
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
ECB	Electronic Codebook
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
FLW	Flash Logical Window Module
IFR	Indexed Flash Region
ISK	Image Signing Key
IV	Initialization Vector
MPU	Memory Protection Unit
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OpenSSL	Open Secure Sockets Layer
PEM	Privacy-enhanced Electronic Mail
PUF	Physical Unclonable Function
RAM	Random Access Memory
ROM	Read-Only Memory
RSA	Rivest–Shamir–Adleman
SB2	Secure Binary version 2
SB3	Secure Binary version 3
SDK	Software Development Kit
SEC	Standards for Efficient Cryptography
SHA	Secure Hash Algorithm
SP	Substitution–Permutation
SRAM	Static Random Access Memory
SSS	Security Sub-System

Seznam příloh

A Dokumentace

A.1 Formáty kontejnerů

Pro zabezpečený start se na architektuře K4 používá knihovna nboot, která umožňuje ověření podepsaného obrazu a instalaci podepsaného šifrovaného obrazu firmwaru [6]. Funkce, které knihovna nboot poskytuje, jsou postaveny na dvou formátech kontejnerů.

A.1.1 Formát kontejneru podepsaného obrazu

Je určen pro spouštěcí obrazy a ostatní části kódu, které se zde spouští. Pro vygenerování ECDSA podpisu obrazu, je třeba použít heš SHA-512, který se vypočítává z celého obrazu a privátního klíče ISK (Image Signing Key) certifikátu, případně privátního klíče z root certifikátu.

Veřejné ECC klíče jsou uloženy v CTRK (Customer Trusted Root Key) tabulce a jsou svázány k SHA-256 heši v pojistkách neboli fuses. Pro tvorbu veřejných ECC klíčů můžeme použít křivky secp256r1, secp384r1, secp521r1. CTRK tabulka může obsahovat až 4 veřejné klíče. V hlavičce certifikačního bloku je specifikováno, který klíč se použije pro podepsání ISK certifikátu. Nad zřetězením veřejných ECC klíčů je vypočítáván CTRK heš. Pro vytvoření ISK certifikátu je zapotřebí nejdříve vygenerovat veřejný klíč ISK (k tomu je použito náhodné číslo), poté je přiřazeno sériové číslo, do kterého se přidají ISK verze. Vše je následně podepsáno privátním klíčem ISK.

A.1.2 Formát kontejneru firmwaru

Formát kontejneru firmwaru, který lze nahrát neboli SB3 (Secure Binary version 3). Poskytuje autentizaci, integritu a důvěrnost. Je určen pro bezpečnou aktualizaci firmwaru, bezdrátově nebo přes sériový port. Používají se zde algoritmy ECDSA, AES-256, využívající operační mód CBC a heš SHA-256. SB3 se dělí do několika bloků. Všechny bloky jsou stejně velké (typicky 324 bajtů), pouze první blok má odlišnou velikost (1068 bajtů).

První blok (Block 0) je miniaturní podepsaný obraz, ve kterém, se nachází datová oblast podepsaného obrazu (obsahuje hlavičku v otevřeném textu, ukazatel na tabulku certifikátů, ukazující do prvního bloku, AES-256 klíč následujícího bloku a SHA-256 heš následujícího bloku), certifikační blok a ECDSA podpis, který používá ISK nad prvním blokem. Veškeré další bloky jsou šifrovány algoritmem
AES-256, využívající operační mód CBC. V blocích se nachází počítadlo bloků,

datové sekce obrazu firmware, AES-256 klíč pro následující blok, SHA-256 heš následujícího bloku.

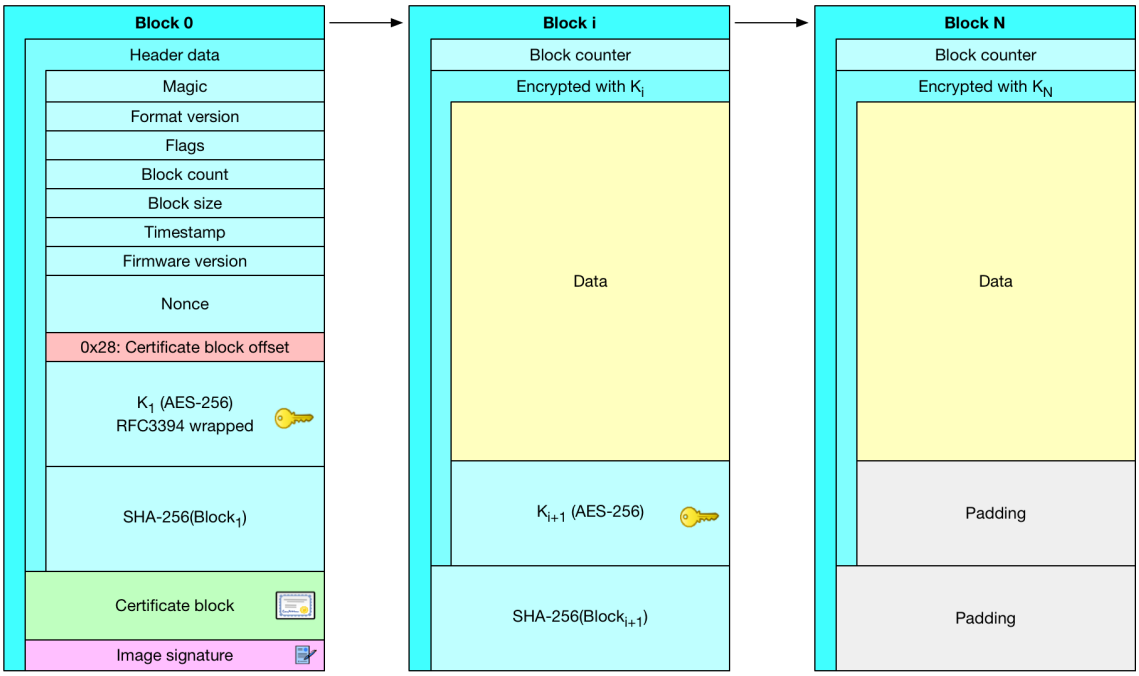
A.2 Klíče

V podepsaných obrazech se používají dva ECC klíče:

- CTRK (Customer Trusted Root Key) – je svázaný s hešem a ukládá se do pojistek. Klíč CTRK je povinný.
- ISK (Image Signing Key) – je podepisován CTRK. Klíč ISK je volitelný.

A.3 Datový rozsah sekcí

První blok souboru SB3 má tři hlavní části - hlavičku (header), certifikační blok (certificate block) a ECDSA podpis obrazu (image signature). Tyto části jsou zobrazeny na obrázku A.1.



Obr. A.1: Struktura souboru SB3 [6].

Certifikační blok tvoří hlavička bloku, CTRK tabulka, která obsahuje veřejné klíče a ISK certifikát, který obsahuje veřejný klíč, verzi a podpis certifikátu. Certifikační blok se využívá při podepisování obrazu. Blok je zobrazen na obrázku A.2.

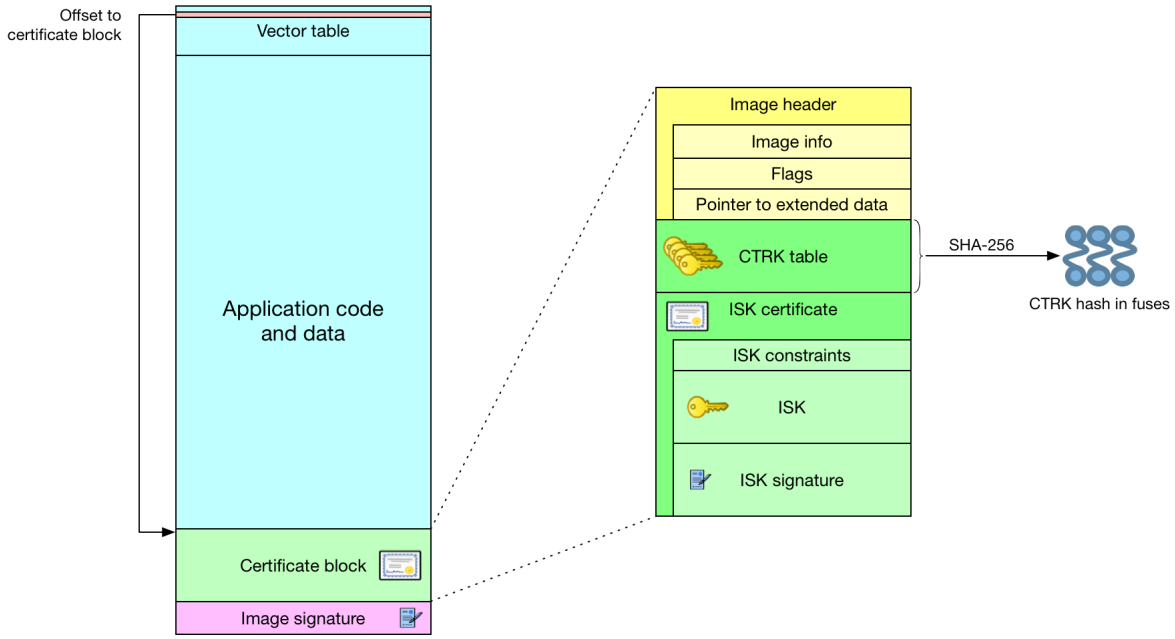
Adresa	Pole	Velikost v bytech	Popis
0x00	magic	4	String "sbv3"– označuje počátek hlavičky.
0x04	version	4	Verze formátu (major=3, minor=0).
0x08	flags	4	Zatím není definováno.
0x0C	block_count	4	Počet bloků.
0x10	block_size	4	Velikost jednoho bloku.
0x14	timestamp	8	Časové razítko – udává kdy byl obraz vytvořen.
0x1C	firmware_version	4	Číslo verze firmwaru.
0x20	image_total_length	4	Celková délka obrazu včetně podpisů.
0x24	image_type	4	Informace o typu obrazu (SB3=6, SB3_NXP=7).
0x28	certificate_block_offset	4	Pozice od začátku hlavičky bloku do bloku certifikátu (umožňuje ověřit podpis přes hlavičku bloku).

Tab. A.1: SB3 pole hlavičky.

Za blokem hlavičky je umístěna datová oblast, která se dělí na části o stejných velikostech. Tyto části jsou poté umisťovány do vlastního bloku. Velikost datové části můžeme vypočítat dle rovnice:

$$datova\_cast = velikost\_bloku - blok\_rezijnich\_nakladu \tag{A.1}$$

Při tvorbě datové části o velikosti 256 bajtů, bude celková velikost bloku 324 bajtů. Blok režijních nákladů (overhead) má velikost 68 bajtů. Blok režijních nákladů, je součet velikostí počítadla bloku (block counter) – 4 bajty, klíče blou K_{i+1} (AES-256) – 32 bajtů a heše bloku (SHA-256) – 32 bajtů.



Obr. A.2: Podrobné zobrazení certifikačního bloku [6].

Adresa	Pole	Velikost v bajtech	Popis
0x00	magic	4	String "chdr"– označuje počátek hlavičky.
0x04	version	4	Verze formátu (major=2, minor=0).
0x08	flags	4	Informace o použití ISK (použit=6, nepoužit=7).
0x0C	used_root_cert	4	Informace o počtu root certifikátů (1 certifikát=0, 2 certifikáty=1, 3 certifikáty=2, 4 certifikáty=3).
0x10	firmware_version	4	Číslo verze firmwaru.
0x14	extended_data_pointer	4	Pozice od začátku obrazu k rozšířeným datům.

Tab. A.2: Hlavička bloku certifikátu.

Jelikož podepisujeme blok předchozím blokem, tudíž u posledního bloku je již místo klíče následujícího bloku (AES-256) a heše následujícího bloku (SHA-256) pouze výplň (padding).

Datový tok se rozděluje do sekcí, kde každá sekce má unikátní typ a ID. Délka sekcí je vždy násobkem 16 bajtů. Tato velikost bloku je vyžadována pro šifrování

algoritmem AES.

```
class CmdSectionHeader():
    FORMAT = ">4L"
    SIZE = calcsize(FORMAT)

    def __init__(self, section_uid=0, section_type=0, length=0):
        self.section_uid = section_uid
        self.section_type = section_type
        self.length = length
        self._pad = 0
```

Sekce datového rozsahu se skládají z jednoho nebo více rozsahů, kde každý rozsah začíná hlavičkou:

```
class CmdHeaderNew:
    FORMAT = ">4L"
    SIZE = calcsize(FORMAT)
    TAG = 0x55aaaa55

    def __init__(self, cmd=EnumCmdTag.NONE):
        self.address = 0
        self.length = 0
        self.cmd = cmd
```

A.3.1 SB3 Příkazy (cmd)

V datovém rozsahu může být prováděno několik akcí, pro které se používají příkazy: Erase, Load, Execute, Call, programFuses, programIFR.

- Příkaz Erase vymazává flash paměť daného adresového rozsahu. Vymazání se zaokrouhluje na velikost sektoru, který má být vymazán.
- Příkaz Load zajišťuje, že data pro zápis následují hlavičku rozsahu. Pokud je třeba použít výplň musí po datech připojit tak, že začátek dalšího rozsahu nebo hlavička sekce je zarovnaná na 16 bajtů.
- Příkaz Execute zajišťuje, že jako počáteční adresa bude nastavená adresa směřující do konkrétního umístění v RAM, která za pomoci příkazu Execute začne vykonávat aplikaci z této adresy.
- Příkaz Call zajišťuje, že jako počáteční adresa bude nastavená adresa směřující do konkrétního umístění v RAM a je očekáván návrat na následující výraz pro pokračování zpracovávání souboru SB3.
- Příkaz programFuses zajišťuje, že počáteční adresa bude adresa registru pojistek, délka bude počet slov v pojistkách pro program. Data zapsaná do registrů pojistek budou bezprostředně následovat hlavičku.

- Příkaz programIFR zajišťuje, že počáteční adresa bude adresa v oblasti IFR (Indexed Flash Region), délka bude v počtu bajtů, pro zápis do oblasti. Data pro zápis do oblasti na dané adrese budou bezprostředně následovat hlavičku.

Příklady příkazů [6]:

- CmdEraseNew:
  tag = 0x55aaaa55
  address = 0x0000_0000
  length = 8192
  cmd = 0x01

- CmdExecuteNew:
  tag = 0x55aaaa55
  address = 0x0000_0000
  length = 0
  cmd = 0x03

- CmdCallNew:
  tag = 0x55aaaa55
  address = 0x0000_0000
  length = 0
  cmd = 0x04

- CmdProgFusesNew:
  tag = 0x55aaaa55
  address = 0x0000_0000
  length = 16 (16 * 4 bajty dat)
  cmd = 0x05

- CmdProgIfrNew:
  tag = 0x55aaaa55
  address = 0x0000_0000
  length = 16
  cmd = 0x06

- CmdSectionHeader:
  section_uid = 1
  section_type = 1
  length = collection_of_commands

A.4 Ověřování podpisu obrazu

Sekvence ověření podpisu obrazu:

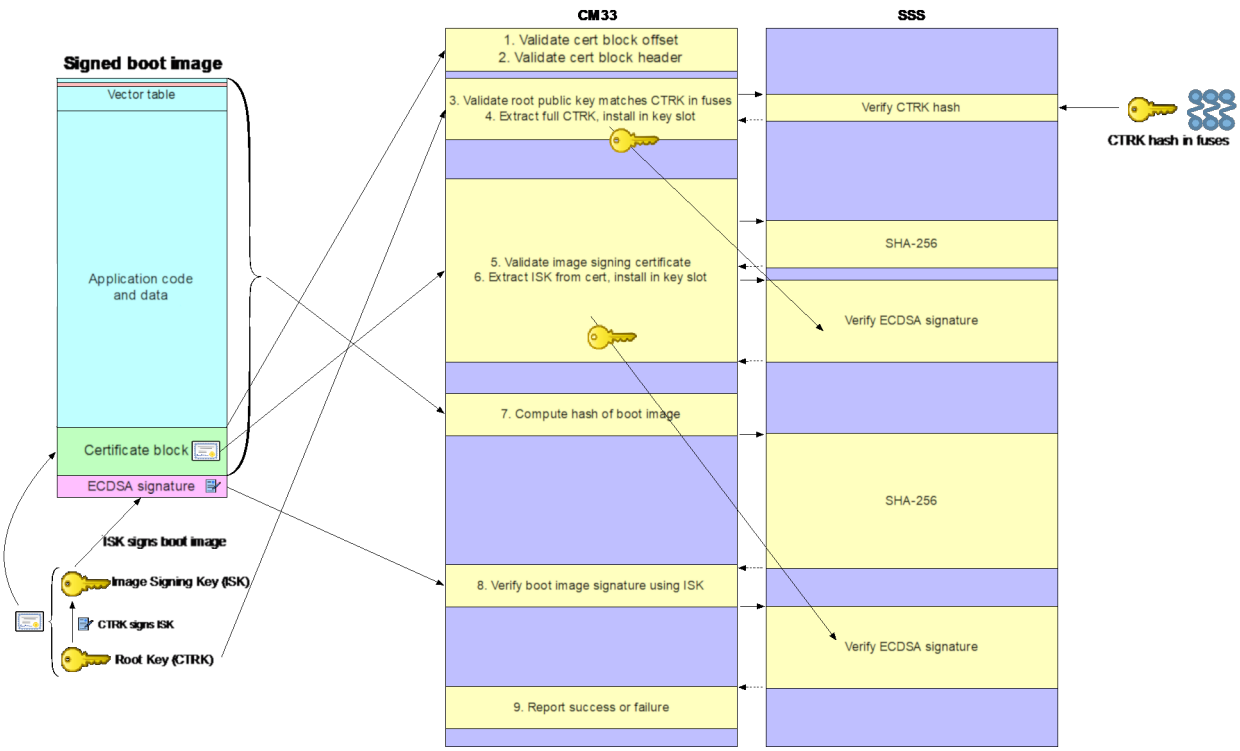
1. ověří se ukazatel na blok certifikátu, který je na adrese (s offsetem) 0x28 ve vektorové tabulce obrazu,
2. ověří se hlavička certifikačního bloku,
3. ověří se, že se kořenový veřejný klíč v bloku certifikátu shoduje s CTRK hešem, uloženým v pojistkách (fusese),
4. extrahuje se úplný kořenový veřejný klíč (CTRK) z bloku certifikátu a nainstaluje se ve slotu klíče v SSS (Security Sub-System),
5. ověří se certifikát pro podpis obrazu (CTRK podepisuje ISK),
6. extrahuje se podpisový veřejný klíč obrazu (ISK) a nainstaluje se do slotu klíče v SSS,
7. vypočte se heš (SHA-512) z celého obsahu obrazu,
8. ověří se ECDSA podpis použitím veřejného podpisového klíče obrazu (ISK), který byl předtím extrahován,
9. nakonec je nahlášen úspěch nebo selhání do jádra Cortex-M33.

Posloupnost ověření podpisu obrazu a vztah k operacím prováděným pomocí SSS jménem CM33 ROM je zobrazen na obrázku A.3

A.5 Bezpečná aktualizace firmwaru

Je vyžadováno, aby byla aktualizace obrazu firmwaru provedena bezpečně. To znamená, že autenticita obrazu může být ověřena před instalací či spuštěním. Je také důležité, aby se neoprávněná osoba nemohla vrátit zpět k verzi obrazu firmwaru, která je nižší než aktuální minimální povolená verze. Je třeba zajistit, že nenastane situace, kdy vada v procesu aktualizace nebo nově nainstalovaném obrazu ponechá zařízení ve stavu, ze kterého nemůže být nikdy obnoveno do „bezpečného“ obrazu.

Rozlišujeme mezi validovaným obrazem a verifikovaným obrazem. Verifikace obrazu může probíhat automaticky, kdy se nástrojem či aplikací ověřuje, že aktuálně běžící obraz, dosáhl bodu, ve kterém je funkcionalita obrazu verifikovaná a považovaná za provozní, do takové míry, že může proběhnout aktualizace obrazů firmwarů. Případně manuální ověřování, kdy uživatel za pomoci manuální akce potvrdí, že provoz zařízení je v pořádku.



Obr. A.3: Posloupnost ověření obrazu [6].

A.5.1 Předpoklady a omezení

Jelikož vyžadujeme, aby byl obraz schopný alespoň úspěšně aktualizovat obraz firmwaru tak, aby bylo zařízení obnoveno do nového obrazu, jsou na architektuře K4 určitá omezení a předpoklady pro dodržení této schopnosti.

Například verifikace obrazu, která ověřuje, zda obraz funguje správně, by měla pokrývat sadu spouštěcích obrazů pro všechny relevantní procesory v systému.

Dalším příkladem je schopnost řídícího jádra startu, která ukládá limitované množství dat, které přetrvávají i po restartovacích cyklech. A samozřejmě také možnost úpravy minimálního čísla verzí, které zprostředkovává bezpečný podsystém.

Za stabilní situaci můžeme považovat situaci kdy minimální povolená verze firmwaru je označena jako verze n. Jsou dostupné dva obrazy (primární a sekundární), které jsou ověřené (validované a verifikované) a jsou s verzí n. Je možné mít dva obrazy s rozdílnými verzemi v určitém čase např. jako výsledek aktualizovaného obrazu.

Úspěšné ověření obrazu vyžaduje, aby kryptografický podpis byl ověřený přes root klíč, uložený v zařízení. Verze obrazu musí být minimálně taková, jako je vyžadovaná minimální verze, uložená v pojistkách.

A.5.2 Sekvence aktualizace firmwaru:

1. při normálním provozu, běží firmware s verzí *n*,
2. přijme se nový obraz firmware verze *n+1* a zapíše se do aktuálně neaktivního umístění obrazu,
3. v tuto chvíli má systém v primárním a sekundárním umístění dva obrazy s rozdílnou verzí (*n* a *n+1*) a za použití FLW (Flash Logical Window Module) by mělo řídící jádro startu vyměnit umístění obrazů tak, aby umístění primárního obrazu ukazovalo na obraz s číslem nejvyšší verze,
4. na spouštěném či restartovaném zařízení, zkontroluje řídící jádro startu, jestli je nastavena značka pokusu o autentizaci, pokud je tato značka stále nastavená, autentizace selhala a řídící jádro startu se pokusí bootstrapovat systém z obrazu v sekundárním umístění, což probíhá takto:
   řídící jádro startu nastaví značku v tzv. reset-safe paměti neboli paměti, která není ovlivněna provedením restartu.
   Značka indikuje, že se pokusí o spuštění primárního obrazu, poté řídící jádro startu bootstrapuje systém použitím obrazu nalezeném v primárním umístění, který by měl být ten nově nainstalovaný firmware verze *n+1*, pak bezpečný podsystém, ověří obraz firmware verze *n+1*. Pokud projde ověření, bezpečný podsystém si ponechá nové číslo verze firmware pro případné pozdější předložení tohoto nového čísla verze do pojistek, pokud je později systém považován jako funkční,
5. pokud ověření selže, řídící jádro startu restartuje systém, pokud je ověření úspěšné, řídící jádro startu resetuje značku pokusu o autentizaci a systém má povoleno pokračovat ve spuštění,
6. pokud je sada obrazů provozuschopná, tak je systém již zapnut a běží. Vyšší úroveň softwarové entity v softwaru, se potřebuje rozhodnout, zda je obraz dostatečně provozuschopný, aby byl považován za verifikovaný a pokud je považován za verifikovaný, je to oznámeno bezpečnému podsystému, který po přijetí oznámení aktualizuje minimální povolenou verzi firmwaru v pojistkách, na verzi firmwaru, která aktuálně běží.

Zpráva posílaná bezpečnému podsystému pro předložení nové minimální verze firmwaru, nemusí být autentizována, jelikož nové číslo verze bylo převzato z podepsaného kontejneru.

B Dokumentace přiloženého CD

- `/bakalarska_prace_bohmova` ..... bakalářská práce
- `/priloha.zip` ..... vrchní adresář CD
  - `bsdk` .....
    - `sbfile` .....
      - `sb3` ..... adresář obsahující kódy
        - `sections.py` ..... soubor obsahující certifikační blok
        - `isk_certificate.py` ..... soubor obsahující ISK certifikát
        - `images.py` ..... soubor obsahující obraz SB3
        - `headers.py` ..... soubor obsahující hlavičku obrazu a certifikátu
        - `ctrk_table.py` ..... soubor obsahující CTRK tabulku
        - `commands3` ..... soubor obsahující příkazy
        - `block.py` ..... soubor obsahující blok SB3
        - `__init__.py` .....
    - `utils` .....
      - `crypto` .....
        - `backend_internal.py` ..... soubor obsahující funkci pro šifrování
  - `examples` .....
    - `sbfile_v3` ..... soubor obsahující testovací uživatelské rozhraní
  - `tests` .....
    - `sbfile` .....
      - `sb3` ..... adresář obsahující testy
        - `test_sections.py` ..... soubor obsahující testy certifikačního bloku
        - `test_sbfile_image.py` ..... soubor obsahující testy obrazu SB3
        - `test_sb3_block.py` ..... soubor obsahující testy bloku SB3
        - `test_isk_certificate.py` ..... soubor obsahující testy ISK certifikátu
        - `test_headers.py` ..... soubor obsahující testy hlaviček
        - `test_ctrk_table.py` ..... soubor obsahující testy CTRK tabulky
        - `test_commands3.py` ..... soubor obsahující testy příkazů
        - `conftest.py` .. soubor obsahující funkci pro specifikaci cest k souborům
        - `common.py` ..... soubor obsahující funkce na vyčtení dat ze souboru
        - `__init__.py` .....
  - `README.md` .....