Bezpečná vzdialená aktualizácia IoT senzora na báze ESP32

[1]*Martin CHLEBOVEC,* [2]*Miloš DRUTAROVSKÝ*

[1,2] Katedra elektroniky a multimediálnych telekomunikácií, Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, Slovenská republika

[1]martin.chlebovec.2@student.tuke.sk, [2]milos.drutarovsky@tuke.sk

***Abstrakt* — Článok opisuje možnosti bezpečnej aktualizácie firmvéru IoT senzora na báze ESP32 v prostredí ESP-IDF, pričom kladie dôraz na prenos aktualizácie cez zabezpečený prenosový kanál, overenie integrity aktualizácie, zabezpečenie bootovacieho procesu a šifrovanie flash pamäte. Kombinácia týchto metód zaručuje vysokú mieru bezpečnosti pred najbežnejšími typmi útokov, ktoré sú cielené na IoT zariadenia.**

***Kľúčové slová* — ESP32, OTA, digitálny podpis, ESP-IDF, zabezpečený bootovací proces, šifrovanie flash pamäte, firmvér**

I. ESP32 – ESPRESSIF SYSTEMS

ESP32 je mikrokontróler z produkcie čínskej firmy Espressif Systems pre integráciu do IoT (internet vecí) aplikácií [1]. Podporuje WiFi (2,4 GHz) a Bluetooth konektivitu, pričom obe technológie zdieľajú spoločnú plošnú anténu na doske plošných spojov, čím sa výrazne redukuje aj samotná veľkosť modulu. WiFi technológia môže byť použitá pre prenos údajov z IoT senzora, ale aj na prevzatie OTA (Over-The-Air) aktualizácie firmvéru zo vzdialeného úložiska (servera).

Modul s ESP32 je vybavený externou flash pamäťou, najčastejšie s veľkosťou 4 MB. Flash pamäť slúži na uloženie nastavení WiFi adaptéra, firmvéru, softvérového bootloadera (zavádzača) a tabuľky partícií, ktorá rozdeľuje flash pamäť logickým členením na partície. ESP32 je najčastejšie vybavené dvojjadrovým procesorom harvardskej architektúry Tensilica Xtensa L6 s taktom až 240 MHz. Hlavný procesor Xtensa môže realizovať výpočty, komunikovať so senzormi a periférami cez dostupné zbernice.

Zároveň však riadi a obsluhuje aj WiFi/Bluetooth zásobník (stack) pre zabezpečenie konektivity. Za správu a obsluhu WiFi/Bluetooth zásobníka zodpovedá jedno z jadier procesora. u ktorého má WiFi/Bluetooth zásobník maximálnu prioritu. Druhé jadro obsluhuje používateľskú aplikáciu. Obe jadrá zdieľajú spoločnú vyrovnávaciu pamäť.

II. VLASTNOSTI A NÁSTROJE PROSTREDIA ESP-IDF

ESP-IDF je framework z produkcie Espressif Systems pre vývoj IoT aplikácií založený na jazyku C pre platformu ESP32 [2]. Je vyvíjaný na Githube s poslednou produkčnou verziou 4.2. Framework ponúka kompatibilné knižnice a ukážkové projekty pre základné programové implementácie pre obsluhu zberníc, rozhraní ako základ pre vývoj vlastnej aplikácie na ESP32.

Umožňuje vyvíjať aplikácie a komplexnejšie programy s využitím plánovača a operačného systému reálneho času – FreeRTOS [3]. Výhodou plánovača je možnosť spúšťania vlastných programov na ESP32 ako samostatné úlohy, pričom je možné každej úlohe priradiť veľkosť zásobníka. V prípade pretečenia zásobníka sa daná úloha ukončí, avšak neovplyvní ostatné úlohy hlavného programu, ktoré sa naďalej vykonávajú.

Súčasťou frameworku sú aj vývojárske nástroje pre implementáciu bezpečnostných funkcionalít a taktiež Python scripty, ktoré dokážu vykonávať rôzne operácie. Scripty sú spúšťané v konzolovej aplikácii prostredia ESP-IDF s príslušným argumentom pre vykonanie konkrétnej úlohy. Základným Python scriptom v prostredí ESP-IDF je *idf.py*, ktorý dokáže obsluhovať príkazy súvisiace s kompiláciou programu, ale aj pre spustenie podprogramov prostredia. Umožňuje vyvolať Menuconfig (konfiguračné menu), ktorým je možné nastaviť aktuálne otvorený projekt pre potreby vývojára. Dokáže vykonať zápis skompilovaného firmvéru a ďalších skompilovaných obrazov do flash pamäte.

Externá flash pamäť môže byť rozdelená tabuľkou partícií, ktorá sa kompiluje spoločne

s hlavnou aplikáciou na logické partície, ktoré môžu byť aplikačné (s podporou), systémové a taktiež príznakové, ak sa používa viac aplikačných partícií pre nastavenie príznaku pre bootovanie (zavedenie) firmvéru z preferovanej partície.

Pre vzdialenú aktualizáciu firmvéru sa využíva rozdelenie partícií schémou „Factory App, two OTA definitions“, ktorá umožňuje uložiť a bootovať až 3 firmvéry z flash pamäte, pričom partícia Factory je vždy vyhradená pre firmvér nahratý cez fyzické USB-UART rozhranie a slúži zároveň ako „fail-safe“ riešenie v prípade, že sa nepodarí bootovať firmvér z iných – OTA partícií (OTA_0 a OTA_1), ktoré sú použité pre uloženie aktualizovaného firmvéru.

Ďalším z dostupných Python scriptov v prostredí ESP-IDF je *esptool.py*, ktorý slúži pre prácu s flash pamäťou ESP32. Dokáže vykonať priamy zápis do flash pamäte (najčastejšie je spúšťaný scriptom *idf.py*).

Označenie „ESPTOOL“ [4] je zároveň aj označením pre balík Python scriptov dostupných v prostredí ESP-IDF (*esptool.py*, *espsecure.py* a *espefuse.py*). Pre kryptografické operácie je v prostredí ESP-IDF dostupný script *espsecure.py*. Operácie súvisiace s jednorazovo programovateľnými pamäťami eFuses (4x 256-bit) obsluhuje script *espefuse.py*.

III. VZDIALENÁ AKTUALIZÁCIA FIRMVÉRU V PROSTREDÍ ESP-IDF

ESP-IDF obsahuje viacero ukážkových projektov pre vzdialenú aktualizáciu firmvéru. Projekt Native OTA [5] umožňuje vykonať vzdialenú aktualizáciu firmvéru z webservera, ktorý aktualizáciu distribuuje s využitím bezpečného HTTPS (Hypertext Transfer Protocol Secure) protokolu s end-to-end šifrovaním pre architektúru klient-server. Na webserver vykonáva mikrokontróler ESP32 (klient) jednorazovú GET požiadavku pre prevzatie obsahu firmvéru (.bin obraz). Aktualizovaný firmvér je zapísaný do partície OTA_0, alebo OTA_1, odkiaľ je následne bootovaný.

Pre realizáciu zabezpečeného spojenia na strane ESP32 sa vyžaduje certifikát certifikačnej autority v .pem formáte, ktorý sa vloží do preddefinovaného priečinka projektu, kde ho očakáva kompilátor a linker. Certifikačná autorita vydala certifikát pre webserver, na ktorý realizuje pripojenie ESP32. Webserver predkladá klientovi certifikát pri realizácii zabezpečeného spojenia. ESP32 dôveruje certifikátu webservera, keďže ho vydala dôveryhodná certifikačná autorita, ktorej certifikátom klient disponuje. Využíva sa Chain of Trust (reťaz dôvernosti).

Projekt Native OTA sa konfiguruje cez Menuconfig prostredia ESP-IDF. Konfigurovať je možné cestu k cieľovému firmvéru (URL), kde ho distribuuje webserver a taktiež aj údaje súvisiace s konektivitou (SSID, heslo WiFi siete). Native OTA umožňuj vykonávať aj kontrolu verzie firmvéru. Dokáže porovnať verziu aktuálne bežiaceho a stiahnutého firmvéru. V prípade, že sú verzie rozdielne, prepíše sa príznak pre primárne bootovanú partíciu v OTA_DATA partícii a vykoná softvérový reštart pre bootovanie aktualizovaného firmvéru. V prípade totožných verzií sa reštart, ani zmena v OTA_DATA partícii nevykoná.

IV. ZABEZPEČENIE AKTUALIZÁCIE FIRMVÉRU – DIGITÁLNY PODPIS

Keďže projekt Native OTA nemá mechanizmy pre overenie integrity firmvéru a bootloadera, je nutné tieto bezpečnostné funkcionality implementovať dodatočne prostredníctvom nástrojov nato určených, ktoré sú obsiahnuté v Menuconfigu prostredia ESP-IDF. Výhodou funkcionalít je, že ich nie je nutné implementovať do zdrojových kódov aplikácie.

Metóda digitálneho podpisu [6] umožňuje overiť integritu firmvéru. Overený firmvér je bezpečný pre spustenie na vývojovej platforme ESP32. Digitálny podpis vychádza z asymetrickej kryptografie, ktorá používa dvojicu kľúčov – súkromný a verejný. Súkromný kľúč je použitý pre manuálne podpísanie firmvéru vývojárom. Po podpísaní firmvéru ho môže vývojár sprístupniť na OTA serveri, odkiaľ je distribuovaný klientom, ktorí ho sú schopní overiť v prípade, že majú verejný kľúč podpisujúceho k dispozícii.

Verejný kľúč je v procese kompilácie vložený do obrazu softvérového bootloadera. Softvérový bootloader realizuje overenie firmvéru pri jeho bootovaní s možnosťou overenia firmvéru aj v procese samotnej aktualizácie „ON UPDATE“.

Prostredie ESP-IDF využíva kryptografiu na báze ECC (Elliptic Curve Cryptography), ktorá je založená na zložitosti hľadania diskrétneho logaritmu pre náhodný bod eliptickej krivky nad konečným poľom. Pre samotný digitálny podpis sa využíva podpisová schéma ECDSA (Elliptic Curve Digital Signature Algorithm).

Pre implementáciu digitálneho podpisu v prostredí ESP-IDF je možné použiť script *espsecure.py*, ktorý je možné spustiť v prostredí ESP-IDF cez konzolovú aplikáciu frameworku. Príkazom *espsecure.py generate_signing_key private.pem* je možné vygenerovať súkromný kľúč pre eliptickú krivku „NIST256p“. Generovanie sa realizuje s využitím entropie operačného systému, na ktorom je ESP-IDF spustené. Ako alternatívu je možné použiť aj kryptografický

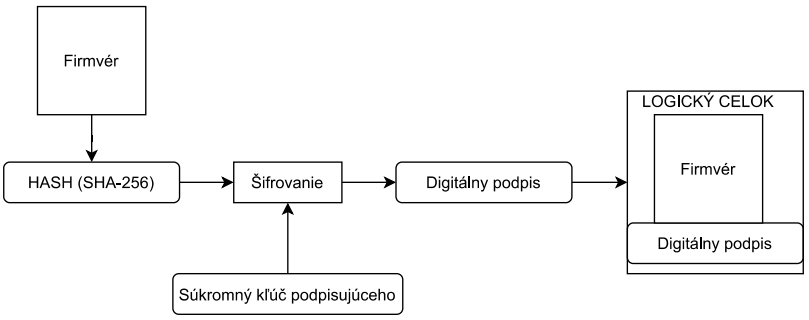
nástroj OpenSSL, ktorý umožňuje generovať kľúč pre identickú eliptickú krivku s označením „prime256v1“. Výstupom scriptu je súkromný kľúč „private.pem“ s dĺžkou 256 bitov v .pem formáte.

Verejný kľúč je možné scriptom *espsecure.py* vygenerovať zo súkromného kľúča príkazom *espsecure.py extract_public_key --keyfile private.pem public.bin*. Aby bolo možné metódu digitálneho podpisu pre aplikáciu založenú na ESP32 použiť, je potrebné v prostredí ESP-IDF v Menuconfigu v časti „Security Features“ zapnúť možnosti „Require signed apps“ a „Verify via Bootloader on startup“, pričom je možné digitálny podpis overiť aj priamo v procese stiahnutia firmvéru, pri akcii „ON_UPDATE“, ktorú je v menu tiež možné vybrať.

Ďalším parametrom, ktorý je potrebné nastaviť je relatívna cesta (vzhľadom na koreňový priečinok projektu) k verejnému kľúču – „public.bin“, ktorý sa v procese kompilácie vloží do obrazu softvérového bootloadera. Podpísanie firmvéru je potrebné realizovať manuálne po kompilácii firmvéru a pred jeho nahrávaním do ESP32 cez USB-UART rozhranie, alebo pred vložením firmvéru na OTA webserver.

Pre podpísanie firmvéru je možné použiť *espsecure.py* a príkazom *espsecure.py sign_data --version 1 --keyfile private.pem --output native_ota.bin native_ota.bin* sa vykoná podpísanie firmvéru „native_ota.bin“ súkromným kľúčom „private.pem“.

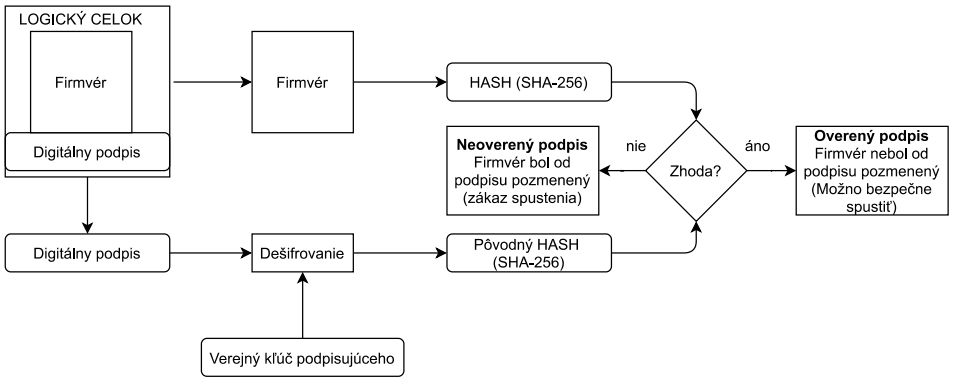
V procese podpisovania firmvéru sa z pôvodného súboru „native_ota.bin“ vypočíta hašovaná hodnota funkciou SHA256 (Secure Hash Algorithm). Odtlačok je zašifrovaný súkromným kľúčom a jeho výsledkom je digitálny podpis, ktorý je vložený do firmvéru za jeho pôvodný obsah. Podpis má dĺžku 68 bajtov, pričom prvé 4 bajty sú slovo tvorené nulami a nasledujúcich 64 bajtov je samotný digitálny podpis. Bloková schéma podpísania firmvéru vývojárom je opísaná na Obr. 1.



Obr. 1 Proces podpísania firmvéru súkromným kľúčom

Verejný kľúč podpisujúceho je zapísaný v obraze softvérového bootloadera. Bootloader kľúčom dešifruje digitálny podpis a získa pôvodnú hašovanú hodnotu SHA256 firmvéru. Následne vytvorí hašovanú hodnotu SHA256 firmvéru a porovná obe hašované hodnoty. V prípade, že sa zhodujú, firmvér je autentický a nebol pozmenený treťou osobou od momentu podpísania.

Firmvér je dôveryhodný a je ho možné spustiť bezpečne na ESP32. Úspešnosť samotného overenia spustí následne funkciu z programu Native OTA, ktorá overuje, či je verzia aktuálne spusteného firmvéru iná, ako verzia stiahnutého firmvéru. V prípade rozdielnych verzií firmvérov sa vykoná prepis v príznakovej partícii OTA_DATA, ktorá definuje primárne bootovanú partíciu s firmvérom a vykoná sa softvérový reštart pre možnosť bootovania aktualizovaného firmvéru z flash pamäte. Proces overenia firmvéru je blokovou schémou opísaný na Obr. 2.



Obr. 2 Proces overenia firmvéru verejným kľúčom

V. ZABEZPEČENIE BOOTOVACIEHO PROCESU – SECURE BOOT

Secure Boot (SB) je metóda zabezpečenia bootovacieho procesu. Kombinuje použitie metódy digitálneho podpisu firmvéru a jeho overenia softvérovým bootloaderom v procese bootovania a aktualizácie pre akciu „ON_UPDATE“. Zároveň je SB [6] rozšírená o overenie softvérového bootloadera na začiatku bootovacieho procesu, ktorú obsluhuje hardvérový bootloader uložený v ROM pamäti.

Pre produkčné aplikácie je vhodné použiť SB s „One-Time Flash“ nastavením, ktoré využíva AES (Advanced Encryption Standard) symetrický kľúč uložený v jednorazovo programovateľnej pamäti eFuse, odkiaľ ho dokáže prečítať a použiť výhradne hardvérový bootloader, keďže eFuse je chránená pred softvérovým prečítaním jej obsahu, či prepisu jej obsahu. V princípe ide o výpočet odtlačku obrazu softvérového bootloadera uloženého vo flash pamäti na ofsete 0x1000 algoritmom SBDA (Secure Bootloader Digest Algorithm).

Vypočítaný odtlačok je následne porovnaný s referenčným, ktorý je uložený vo flash pamäti na preddefinovanom ofsete 0x0, kde ho zapisuje vývojár, ktorý ho vygeneroval lokálne. Výpočet odtlačku a porovnanie na strane ESP32 realizuje hardvérový bootloader. V prípade, že je odtlačok identický, softvérový bootloader je dôveryhodný. Hardvérový bootloader umožní softvérovému bootloaderu pristúpiť k bootovaniu firmvéru a overenia jeho digitálneho podpisu. V prípade, že sú odtlačky rozdielne, fáza bootovanie firmvéru je zakázaná a ESP32 sa reštartuje v nekonečnej slučke a skúša opätovné overenia obrazu softvérového bootloadera.

Algoritmus SBDA využíva AES kľúč (256-bit) z eFuse BLK2, ktorý načíta v reverznej bitovej reprezentácii a obraz softvérového bootloadera z flash pamäte z ofsetu 0x1000. Vygeneruje sa 128-bajtový inicializačný vektor, ktorý sa dosadí pred bootloader. Vykoná sa zarovnanie obrazu bootloadera modulo 128 a následne jeho dorovnanie pre modulo 128 s doplnením hodnôt 0xFF do reprezentácie obrazu.

Na každých 16 bajtov otvoreného textu obrazu bootloadera sa aplikuje bloková šifra AES256 v ECB móde s využitím AES kľúča z eFuse BLK2 s dĺžkou 256 bitov. Výsledný šifrovaný text má reverznú bitovú reprezentáciu. Algoritmus vymení bajt každého 4-bajtového slova šifrovaného textu, vypočíta hašovanú hodnotu SHA512 výsledného šifrovaného textu. Výstupom je 192-bajtový reťazec, ktorý je tvorený 128-bajtovým inicializačným vektorom a 64-bajtovou hašovanou hodnotou SHA512 zo šifrovaného textu.

Vývojár, ktorý lokálne generuje odtlačok má k dispozícii AES kľúč, ktorý predtým vygeneroval a je identický s kľúčom zapísaným v bloku eFuse BLK2. Príkazom *espsecure.py generate_ generate flash_encryption_key secure-bootloader-key-256.bin* je možné vygenerovať AES kľúč „secure-bootloader-key-256.bin“. Pre generovanie sa používa Python funkcia *os.random()*, ktorá využíva viac faktorov operačného systému, na ktorom je spustené rozhranie ESP-IDF pre vyššiu entropiu – náhodnosť kľúča.

SB je možné zapnúť cez Menuconfig, v časti „Security Features“, avšak pre permanentné zapnutie je nutné zapísať potvrdzovací bit do 1-bitovej eFuse ABS_DONE_0. Príkazom *espefuse.py burn_key secure_boot secure-bootloader-key-256.bin* sa vykoná zápis AES kľúča do eFuse BLK2. Potvrdzovací bit do eFuse ABS_DONE 0 je možné zapísať príkazom *espefuse.py burn_efuse ABS_DONE_0*. Potencionálny útočník nedokáže získať kľúč uložený v tejto eFuse, keďže k nej nemá prístup žiadnym softvérovým nástrojom.

K bloku eFuse BLK2 môže pristupovať iba hardvérová funkcionalita SB prostredníctvom hardvérového bootloadera uloženého v ROM pamäti. Táto metóda je efektívna pred spustením modifikovaného bootloadera, ktorý by mohol útočník do ESP32 nahrať cez fyzické USB-UART rozhranie, čím by mohol spustiť bootovanie nepodpísaného (nebezpečného) firmvéru, ktorý by mohol byť podvrhnutý v aktualizácii, alebo nahratý cez fyzické USB-UART rozhranie do ESP32.

VI. ŠIFROVANIE FLASH PAMÄTE – FLASH ENCRYPTION

Flash Encryption (FE) je metóda šifrovania flash pamäte. Umožňuje šifrovať istú časť flash pamäte, alebo celý jej obsah [7]. Každá šifrovaná partícia je v tabuľke partícií definovaná príznakom encrypted. Šifrovaná časť flash pamäte nie je potencionálnym útočníkom spustiteľná (napríklad firmvér), keďže dokáže prevziať cez USB-UART rozhranie iba zašifrovaný firmvér, ktorý nie je bez symetrického AES kľúča spustiteľný, keďže musí byť dešifrovaný.

V otvorenom texte môže byť naďalej distribuovaná aktualizácia firmvéru, ktorý je zašifrovaný v procese zápisu do flash pamäte do aplikačnej partície označenej príznakom encrypted. FE používa symetrický AES kľúč s dĺžkou 256 bitov pre šifrovanie a dešifrovanie obsahu flash pamäte. Kľúč je uložený v jednorazovo programovateľnej pamäti eFuse BLK1 s veľkosťou 256 bitov, ktorá je pre tento kľúč vyhradená. K pamäti eFuse BLK1 dokáže pristupovať iba hardvérový bootloader, ktorý operácie šifrovania a dešifrovania vykonáva.

Všetky aplikačné partície, tabuľka partícií, softvérový bootloader a referenčný odtlačok je vždy šifrovaný bez ohľadu na príznak. Funkcionalitu FE je možné zapnúť cez Menuconfig v časti „Security Features“. Režim „Release“ je vhodný pre produkčné aplikácie, keďže šifrovanie flash pamäte nie je možné v budúcnosti vypnúť z dôvodu, že je príznaková eFuse stavu šifrovania flash pamäte chránená proti prepisu.

Po skompilovaní bootloadera s nastaveným príznakom pre funkcionalitu FE a jeho nahratí do ESP32 je obsah flash pamäte nešifrovaný. Za predpokladu, že je spustená aj funkcionalita SB, hardvérový bootloader overí softvérový bootloader pre možnosť spustenia ďalších fáz bootovacieho procesu. Po úspešnom overení softvérový bootloader na základe príznaku šifrovania flash pamäte načíta hodnotu bloku eFuse FLASH_CRYPT_CNT.

Ak je jej hodnota 0 (ešte nešifrovaná flash pamät), nastaví a aktivuje sa blok šifrovania obsahu flash pamäte, bootloader nastaví 4-bitovú eFuse FLASH_CRYPT_CONFIG na hodnotu 0xF. Operácie samotného šifrovania flash pamäte už vykonáva hardvérový bootloader, keďže softvér nemá prístup k bloku eFuse BLK1, kde je uložený symetrický AES kľúč.

Pre šifrovanie aj dešifrovanie sa využíva AES kľúč zapísaný do eFuse BLK1 s veľkosťou 256-bitov, ktorá je na tento účel určená. Pri väčších partíciách môže trvať šifrovanie až minútu. Softvérový bootloader po ukončení šifrovania nastaví eFuse FLASH_CRYPT_CNT na hodnotu 0x01, čo znamená, že je obsah flash pamäte šifrovaný a opätovné šifrovanie sa pri reštarte a spúšťaní systému nevykoná.

Algoritmus šifrovania flash pamäte využíva blokovú šifru AES256, ktorá pracuje na 32-bajtových blokoch otvoreného textu, využíva dva bloky AES v sérii. AES256 používa symetrický AES v reverznej bitovej reprezentácii. Pre proces šifrovania a dešifrovania flash pamäte sa používajú kryptografické operácie AES opačne. Proces šifrovania je tvorený funkciou AES decrypt a proces dešifrovania flash pamäte funkciou AES encrypt.

Pri šifrovaní sa na každý 32 bajtový blok otvoreného textu aplikuje unikátny AES kľúč, ktorý je odvodený od hlavného AES kľúča vykonaním operácie XOR (exkluzívny súčet) medzi AES kľúčom z eFuse BLK1 a ofsetom bloku otvoreného textu vo flash pamäti. XOR-ovanie konkrétnych bitov kľúča závisí od hodnoty v 4-bitovej eFuse FLASH_CRYPT_CONFIG.

To zaručuje, že sú XOR-ované všetky bity AES kľúča s ofsetom bloku dát. Zmena hodnoty zapísanej v eFuse FLASH_CRYPT_CONFIG by mohla znížiť kryptografickú bezpečnosť šifrovania flash pamäte, keďže by sa operácia XOR nevykonala pre všetky bity AES kľúča. Pre 1. nastavený bit sa vykonáva XOR pre bity 0 až 66, pre 2. bit hodnoty 67 až 131, pre 3. bit hodnoty 132 až 194, pre 4. bit hodnoty 195 až 256.

Prvým krokom pre spustenie FE je vygenerovanie a zapísanie symetrického kľúča do príslušnej jednorazovo programovateľnej pamäte eFuse BLK1. S využitím nástroja *espsecure.py* a príkazom *espsecure.py generate_flash_encryption_key my_flash_encryption_key.bin* je možné kľúč vygenerovať. Funkcia pre generovanie AES kľúča je totožná ako v prípade kľúča pre SB, ktorá vychádza z náhodnosti operačného systému.

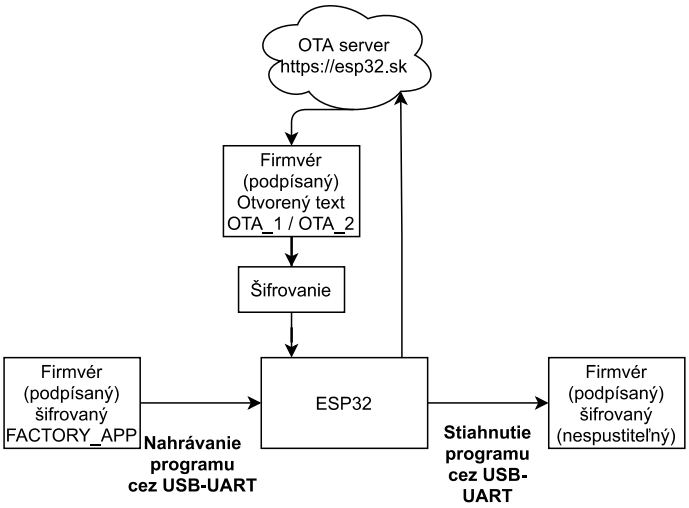
Vygenerovaný kľúč „my_flash_encryption_key.bin“ je následne potrebné zapísať do vyhradenej jednorazovo programovateľnej pamäte eFuse BLK1, ktoré je pre neho určená. Príkazom *espefuse.py - -port PORT burn_key flash_encryption my_flash_encryption_key.bin* sa kľúč zapíše do eFuse BLK1.

Šifrovaný firmvér je možné distribuovať vo vzdialenej aktualizácii v prípade úpravy projektu Native OTA pre zmenu zápisu funkcie firmvéru do flash pamäte na *esp_partition_write()*, ktorá však vyžaduje nastavenie začiatočného ofsetu partície, kde bude firmvér zapísaný. Tento ofset bude použitý aj pre operáciu XOR na ESP32 medzi ofsetom dát a AES kľúčom z pamäte eFuse pre výpočet odvodeného kľúča, ktorým sa následne blok dát šifruje. Z toho dôvodu je možné využívať iba jednu OTA partíciu pre budúce aktualizácie, keďže inak by firmvér nebol dešifrovaný do spustiteľnej podoby správne a zlyhalo by tak jeho bootovanie.

Pre šifrovanie obrazov (firmvér, odtlačok, softvérový bootloader) sa používa príkaz *espsecure.py encrypt_flash_data*, ktorý má viacero parametrov, napr. začiatočný ofset pre správne vykonanie operácie XOR. Python script počíta s nastavením eFuse FLASH_CRYPT_CONFIG na štandardnú hodnotu 0xF pre vykonanie operácie XOR pre všetky bity AES kľúča, ktorý je tiež jedným z parametrov – „my_flash_encryption_key.bin“.

Všetky opísané metódy boli použité v diplomovej práci „Bezpečná aktualizácia firmvéru v senzorovej sieti na báze ESP32“ [8] pre demonštráciu senzorového uzla, ktorý okrem zberu údajov z meteorologického senzora a ovládania digitálneho výstupu na spôsob izbového termostatu vykonáva bezpečnú aktualizáciu firmvéru zo vzdialeného webservera.

Na Obr. 3 je vizualizácia FE pre proces vzdialenej aktualizácie firmvéru, nahratie firmvéru cez fyzické USB-UART rozhranie a pre prevzatie firmvéru, resp. obsahu flash pamäte s príznakom encrypted.



Obr. 3 Metóda šifrovania flash pamäte pre stiahnutie, nahratie firmvéru, prevzatie vzdialenej aktualizácie – Použité v DP pre finálnu aplikáciu pre režim Release [8]

ZÁVER

Opísané metódy zabezpečenia firmvéru, bootovacieho procesu a šifrovania flash pamäte poskytujú vysokú úroveň bezpečnosti pred bežnými typmi útokov cielených na IoT zariadenia. Potencionálny útočník nedokáže na produkčnom mikrokontroléri ESP32 spustiť podvrhnutý bootloader, prípadne firmvér cez fyzické USB-UART rozhranie, alebo cez vzdialenú aktualizáciu. V prípade prevzatia firmvéru z flash pamäte dokáže útočník prevziať iba šifrovaný firmvér, ktorý nie je bez symetrického AES kľúča spustiteľný.

Kľúče k metóde zabezpečeného bootovacieho procesu a šifrovania flash pamäte sú bezpečne uložené v pamätiach eFuses, kde nemá softvér prístup. Zálohu kľúčov má k dispozícii vývojár, ktorý ich používa pre šifrovanie firmvéru a lokálny výpočet referenčného odtlačku bootloadera v prípade jeho úpravy. Útočník tak nedokáže kľúče získať a použiť ich, napríklad pri podvrhnutí firmvéru.

Súčasťou diplomovej práce [8] je detailnejší opis jednotlivých metód zabezpečenia aj s návodom na ich implementáciu v prostredí ESP-IDF pre produkčnú verziu frameworku 4.2. V prílohách diplomovej práce sú zdrojové kódy hlavnej aplikácie na ESP32 a webservera pre demonštráciu IoT senzorového uzla s podporou bezpečnej vzdialenej aktualizácie firmvéru s využitím vlastného webservera pre zber údajov zo senzorového uzla a pre distribúciu vzdialených (OTA) aktualizácií.

POĎAKOVANIE

Táto práca bola podporovaná Agentúrou na podporu výskumu a vývoja na základe Zmluvy č. APVV-18-0373 a Agentúrou na podporu výskumu a vývoja na základe Zmluvy č. 1/0584/20.

REFERENCIE

[1] ESP32 Technical Reference Manual [online]. Espressif Systems [cit. 2020-01-01]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_technical_referenc e_manual_en.pdf
[2] IoT Development Framework - Overview [online]. Espressif Systems [cit 2021-01-20]. Dostupné z: https://www.espressif.com/en/products/sdks/esp-idf
[3] The FreeRTOS™ Reference Manual [online]. Amazon.com, Inc. [cit. 2018-07-01]. Dostupné z: https://www.freertos.org/fr-contentsrc/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf
[4] ESPTOOL (bundle) [online]. Github [cit. 2021-02-22]. Dostupné z: https://github.com/espressif/esptool/
[5] Native OTA [online]. Github [cit. 2021-01-15]. Dostupné z: https://github.com/espressif/espidf/blob/master/examples/system/ota/native_ota_example/main/native_ota_exampl e.c
[6] Secure Boot [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/security/secure-bootv1.html
[7] Flash Encryption [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/flashencryption.html
[8] CHLEBOVEC, M., Bezpečná aktualizácia firmvéru v senzorovej sieti na báze ESP32, Diplomová práca, Košice, Máj 2021, 97 s.