VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2019 Martin Šulič

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

BEZPEČNÉ ULOŽENÍ ŠIFROVACÍCH A AUTENTIZAČNÍCH KLÍČŮ
SECURE STORING OF ENCRYPTION AND AUTHENTICATION KEYS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Martin Šulič**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Vlastimil Člupek, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: Martin Šulič ***ID:*** 195837
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Bezpečné uložení šifrovacích a autentizačních klíčů

POKYNY PRO VYPRACOVÁNÍ:

V bakalářské práci proveďte analýzu možností bezpečného uložení šifrovacích a autentizačních klíčů na PC a na zařízeních využívaných v internetu věcí. Určete náročnost jednotlivých řešení z hlediska implementace na PC a na zařízeních využívaných v internetu věcí a ohodnoťte jejich kryptografickou bezpečnost. Popište možnosti neoprávněného získání šifrovacích a autentizačních klíčů z PC a ze zařízeních využívaných v internetu věcí. Implementujte vybrané způsoby bezpečného uložení šifrovacích a autentizačních klíčů na PC a na zařízení využívaném v internetu věcí. Ohodnoťte kryptografickou bezpečnost implementovaných řešení. Navrhněte vlastní způsob bezpečného uložení šifrovacích a autentizačních klíčů na PC a na zařízeních využívaných v internetu věcí.

DOPORUČENÁ LITERATURA:

[1] VAN TILBORG, Henk CA; JAJODIA, Sushil (ed.). Encyclopedia of cryptography and security. Springer Science & Business Media. 2014.

[2] LINDELL, Yehuda; KATZ, Jonathan. Introduction to modern cryptography. Chapman and Hall/CRC. 2014.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: Ing. Vlastimil Člupek, Ph.D.
Konzultant:

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

UPOZORNĚNÍ:
Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Táto bakalárska práca sa zaoberá analýzou možností bezpečného uloženia šifrovacích a autentizačných kľúčov na PC, ako aj na zariadeniach využívaných v internete vecí. Ďalej popisuje funkcionalitu a úroveň kryptografickej bezpečnosti vybraných riešení a poukazuje na možné bezpečnostné nedokonalosti. Taktiež poskytuje výsledky meraní výkonu blokových šifier na rôznych architektúrach. Najlepšie z nich sme vybrali ako základ vlastného návrhu bezpečného uloženia šifrovacích a autentizačných kľúčov na PC, a na zariadeniach využívaných v internete vecí. Tieto návrhy sme implementovali v podobe konzolovej aplikácie pre PC a aplikácie univerzálnej platformy Windows pre zariadenia využívané v internete vecí, kde sme uviedli aj náročnosť na výpočtový výkon. Ohodnotili sme aj ich kryptografickú bezpečnosť a analyzovali možné bezpečnostné trhliny.

KĽÚČOVÉ SLOVÁ

Autentizačné kľúče, bezpečnosť, blokové šifry, internet vecí, kryptografia, PC, šifrovacie kľúče.

ABSTRACT

This bachelors thesis is focused on analysis of possibilities of secure storing encryption and cryptographic keys in computers and internet of things devices. It also describes the functionality and level of cryptographic security of chosen solutions and points at possible security vulnerabilities. Next part of the thesis consists of results of test of block cyphers on different architectures. Best of them were chosen as a base for custom solution for secure storing encryption and cryptographic keys. The designs were implemented in form of console application for computers and universal Windows platform application for IoT devices, that includes information about computing power needed to use the proposed solution. At last we conducted vulnerability analysis and evaluated their cryptographic security.

KEYWORDS

Authentication keys, security, block ciphers, internet of things, cryptography, PC, encryption keys.

ŠULIČ, Martin. *Bezpečné uložení šifrovacích a autentizačních klíčů*. Brno, Rok, 77 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Vlastimil Člupek, Ph.D.

Vysázeno pomocí balíčku `thesis` verze 3.03; `http://latex.feec.vutbr.cz`

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Bezpečné uložení šifrovacích a autentizačních klíčů“ vypracoval samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som touto cestou poďakoval Ing. Vlastimilovi Člupkovi, Ph.D. za jeho podnetné návrhy, konzultácie a trpezlivosť pri vedení mojej bakalárskej práce. Taktiež chcem poďakovať svojim rodičom, bez ich pomoci by táto práca nevznikla.

Brno ....

podpis autora

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov

Úvod

Kompromitovanie kľúčového materiálu reprezentuje jeden z najviac zákerných útokov. S legitímnym kľúčom môžu útočníci získať dôveryhodný stav, ktorý im umožňuje obísť každé bezpečnostné opatrenie určené na vylúčenie neoprávnených užívateľov. Bežná ochrana firewallmi, virtuálnymi privátnymi sieťami, či mechanizmami autentizácie sa teoreticky rozpadne pri predložení kompromitovaných, ale zároveň valídnych bezpečnostných poverení (kľúčov). Tajné kľúče sú základom väčšiny kryptografických metód a každá táto metóda sa spolieha na ochranu týchto kľúčov. Hlavne v odvetví internetu vecí je potrebné chrániť kľúčový materiál, pretože tieto zariadenia sa často zhlukujú do väčších celkov, kde spolu komunikujú a pracujú s citlivými údajmi. Kompromitáciou už len jedného zariadenia (alebo kľúča) je ohrozená bezpečnosť celého systému. Netreba teda podceniť zabezpečenie aj tých najprimitívnejších zariadení, hlavne v čase, kedy internet vecí dostáva stále väčší priestor. Je treba brať na vedomie, že ochrana kľúčového materiálu na platforme PC, či v internete vecí je dôležitá, ale netreba zabúdať pozerať sa na bezpečnosť ako celok, pretože aj najmenšia bezpečnostná chyba v systéme môže ohroziť na prvý pohľad dobre zabezpečený kľúčový materiál.

V tejto práci sa zaoberáme hlavne analýzou možností bezpečného uloženia šifrovacích a autentizačných kľúčov na PC a na zariadeniach v internete vecí. Jednotlivé riešenia dôkladne popisujeme a ohodnocujeme po stránke bezpečnosti. Rozoberáme tiež možné útoky na vybrané riešenia, či samotné šifry. Venovali sme sa aj implementačnej, či výpočtovej náročnosti celkových riešení a ich častí na rôznych typoch zariadení s odlišnými architektúrami. Vytvorili sme vlastné návrhy bezpečného uloženia šifrovacích a autentizačných kľúčov na PC a zariadeniach IoT, ktoré sme spolu s časťami vybraných zanalyzovaných možností implementovali ako softvérové riešenie v podobe programu (aplikácie) na jednotlivé platformy. Nakoniec sme ohodnotili bezpečnosť našich implementovaných riešení a popísali ich možné slabé miesta (zraniteľnosti).

1 Analýza možností uloženia kľúčov na platforme PC

Na platforme PC môžeme ukladať celý rad rôznych citlivých informácií. Keďže sú to väčšinou výkonné zariadenia, tak aj bezpečnosť môže byť bezproblémovo na vysokej úrovni. Zabezpečiť vysokú úroveň bezpečnsoti je možné mnohými cestami. Preto sme si jednotlivé riešenia rozdelili na hardvérové a softvérové. Sú to dve veľmi odlišné skupiny, ktoré ale vo výsledku medzi sebou spolupracujú a vytvárajú bezpečnostný celok.

1.1 Hardvérové riešenia

Tieto riešenia často fyzicky bezpečne uchovávajú citlivé údaje na špecializovanom hardvéri. Niektoré z nich, ako napríklad šifrovacie kľúče, môžu aj generovať, či overovať ich pravosť. Dôležitou časťou je aj ochrana pred fyzickou manipuláciou s dátami na danom hardvéri.

1.1.1 Modul dôveryhodnej platformy

Trusted Platform Module (TPM) je kryptografický modul, ktorý zvyšuje bezpečnosť počítača a jeho súkromie. Je to takzvaný bezpečnostný kryptoprocesor (mikrokontrolér). TPM je uznávaný ako medzinárodný štandard ISO/IEC 11889 (verzia 1.2) [1]. Tento modul pomáha bezpečnosti pri ochrane dát prostredníctvom šifrovania a dešifrovania, ochrane autentizačných poverení a dokáže ukladať šifrovacie kľúče, ktoré chránia citlivé informácie [2].

Ochrana kľúčov na platforme Windows pomocou TPM

Platform Crypto Provider môže v TPM vytvárať kľúče, ktoré sú obmedzené iba na isté použitie. Operačný systém môže načítať a používať kľúče v module TPM bez toho, aby ich kopíroval do systémovej pamäte, kde sú zraniteľné voči malvéru. Ak TPM vytvorí nejaký kľúč, tento kľúč je jedinečný a nachádza sa iba v tomto nástroji TPM. Ak modul TPM importuje kľúč, potom Platform Crypto Provider môže tento kľúč používať výhradne v danom nástroji TPM, ale samotný nástroj nemôže vytvárať ďalšie kópie kľúča alebo umožniť používať kópie inde. Práve takáto ochrana je výhodou oproti softvérovým riešeniam ochrany kľúčov, ktoré môžu byť terčami útokov pomocou reverzného inžinierstva, takže sa nemôže stať, že niekto by pri narábaní so softvérom mohol prísť na spôsob uloženia kľúčov, alebo si spraviť ich kópie, zatiaľ čo sú načítané a využívané v pamäti [3].

Ochrana pred slovníkovým útokom

Kľúče, ktoré TPM ochraňuje, môžu vyžadovať autentizáciu, napríklad pomocou PINu (Personal Identification Number). S ochranou pred slovníkovým útokom dokáže modul TPM zabrániť útokom, ktoré sa pokúšajú o stanovenie tohto kódu PIN veľkým počtom odhadov. Po určitom počte pokusov TPM jednoducho vráti chybovú hlášku a žiadne ďalšie zadanie PINu nie je umožnené po určitú dobu. Softvérové riešenia môžu poskytovať podobné funkcie, ale nedokážu zabezpečiť rovnakú úroveň ochrany, najmä ak sa systém reštartuje, alebo sa zmenia systémové hodiny, či súbory na pevnom disku, ktoré počítajú neúspešné pokusy sú pozmenené. Okrem toho, s touto ochranou slovníkových útokov, môžu byť autentizačné hodnoty ako PIN lepšie využiteľné pre užívateľa a to kvôli menšej dĺžke na zapamätanie, pričom stále je zachovaná rovnaká úroveň bezpečnosti, naproti komplexnejším (dlhším) hodnotám, ktoré používajú softvérové riešenia [3].

1.1.2 Hardvérový bezpečnostný modul

Hardware security module (HSM) je fyzické zariadenie, ktoré poskytuje mimoriadnu bezpečnosť citlivých údajov pre počítače, servery, ako aj pre odvetvie IoT (Internet of Things). Tento typ zariadenia sa používa na poskytovanie kryptografických kľúčov pre kritické funkcie, ako je šifrovanie, dešifrovanie a autentizácia pre použitie aplikácií, identít, databáz, atď. Celý životný cyklus kryptografického kľúča, od poskytovania, cez správu a ukladanie, až po likvidáciu, alebo archiváciu, nastáva prostredníctvom HSM. Taktiež digitálne podpisy môžu byť zaznamenané a uchovávané prostredníctvom HSM. Úlohou tohto modulu môže byť aj zabezpečenie a správa verejných kľúčov bez dopadu na rýchlosť aplikácií [4]. Ďalšou možnosťou využitia tohto modulu je napríklad generovanie a ochrana kľúčov koreňových certifikačných autorít. Tiež dokáže digitálne podpísať kód aplikácie, aby softvér zostal bezpečný, nezmenený a autentický. HSM dokáže vytvárať digitálne certifikáty na poverenie a autentizáciu proprietárnych elektronických zariadení pre aplikácie IoT a ostatné zariadenia v sieti [5].

HSM by mal byť chránený pred manipuláciou alebo poškodením tak, že by mal byť umiestnený vo fyzicky zabezpečenej oblasti, priestore. Modul môže byť vložený do iného hardvéru, pripojený k serveru ako súčasť siete, alebo použitý ako samostatný prístroj v offline režime [4].

Výhody oproti softvérovým riešeniam

Softvérové riešenia pre uchovávanie šifrovacích a autentizačných kľúčov, väčšinou ukladajú kľúče do hlavnej pamäte, alebo na miestne úložisko. To znamená, že ho-

cikto, kto má prístup na server, alebo dané zariadenie, má teoreticky aj prístup k daným kľúčom a môže si vytvoriť dodatočné kópie a prostredníctvom nich pristupovať k chráneným údajom a dátam. Taktiež softvérové riešenia umožňujú útočníkom využívať slabé miesta v kóde programov a tým dokážu spustiť útoky aj na diaľku. Tento problém rieši HSM. V porovnaní so softvérovými riešeniami ponúka hardvérový modul silnú bezpečnosť. Modul dokáže rozpoznať, kedy dôjde k akémukoľvek útoku, vrátane tých fyzických, ako je napríklad vŕtanie, pôsobenie vysokej teploty, chemický útok alebo výpadok napájania. V takomto prípade dokáže okamžite automaticky odstrániť všetky kľúče. Modul by mal aj zaznamenávať používanie, zmenu stavu, alebo pokus o zmenu kľúčového materiálu [6].

Certifikácie a normy

Všetky vlastnosti zabezpečenia HSM sú odvodené od toho, akú majú certifikáciu podľa uznávanej normy a certifikačnej schémy, ako sú NIST FIPS 140-2, ISO 15408. Norma NIST FIPS 140-2 udáva [7], že úložisko kryptografických kľúčov môže uchovávať kľúče v rámci kryptografického modulu v overenom texte (plaintext) alebo v šifrovanej forme. Plaintext tajomstvá a privátne kľúče nesmú byť prístupné zvonku kryptografického modulu k neoprávneným operátorom. Tento modul musí pričleniť kryptografický kľúč (tajný, súkromný alebo verejný) uložený v module správnej entite (napríklad osoba, skupina alebo proces), ku ktorej je kľúč určený.

1.2 Softvérové riešenia

Citlivé kryptografické materiály môžeme zabezpečiť jednoduchými rišeniami, ako zamedzenie prístupov pomocou oprávnení súviciacich s daným objektom, či aj tým, že budú uložené na disku, ktorý je celý zašifrovaný, alebo len jeho časť, či prípadne ich môžeme uložiť do súboru, ktorý bude zašifrovaný samostatne. Ide o najdostupnejšie riešenia pre obyčajného používateľa, ktorý chce mať kľúče zabezpečené, no zároveň nechce používať zložité riešenia. Väčšina šifrovacích systémov využíva symetrickú šifru AES (Advanced Encryption Standard) s veľkosťou kľúča 128 alebo 256 bitov v móde CBC (Cipher block chaining), alebo XTS (XEX-based tweaked-codebook), niektoré využívajú taktiež šifru 3DES (Triple Data Encryption Standard).

1.2.1 Využívané kryptografické algoritmy

V tejto časti sme popísali často využívané kryptografické algoritmy na platforme PC ako sú DES či AES. Bližšie sme si uviedli ich štruktúru ako aj módy šifry AES.

3DES

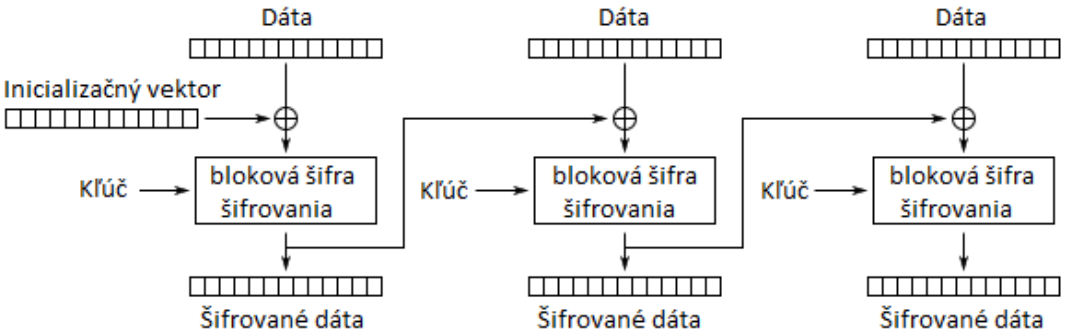
Triple Data Encryption Standard je symetrická bloková šifra, založená na trojnásobnom aplikovaní šifry DES na každý dátový blok. Originálna šifra DES má veľkosť kľúča 56 bitov, ktorá je už v tejto dobe nedostatočná, a preto je táto šifra vystavená útokom hrubou silou. 3DES zvyšuje bezpečnosť tým, že obsahuje kľúčový zväzok s tromi unikátnymi kľúčmi DES. Šifra je konštruovaná ako viacnásobná, často označovaná aj ako kaskáda šifier. Avšak už aj táto šifra je bezpečná len za určitých podmienok, ktoré určil Americký inštitút NIST (National Institute of Standards and Technology) [8]. Jeden kľúčový zväzok nemôže byť použitý na šifrovanie väčšieho množstva dát ako 2^{20} 64 bitových dátových blokov. 3DES je nahradzovaná šifrou AES [9].

AES

Advanced Encryption Standard (AES), tiež známy ako Rijndael, je symetrická bloková šifra, ktorá je štandardom od roku 2001 [10]. Symetrická šifra znamená, že sa používa ten istý kľúč na šifrovanie aj dešifrovanie. Bloková znamená, že spracovávané dáta v prípade AES, sú rozdelené do blokov veľkosti 128 bitov. Počas šifrovania a dešifrovania algoritmus preženie dáta niekoľkými okruhmi s rôznymi základnými operáciami. AES má fixnú dĺžku blokov 128 bitov a veľkosť kľúča 128, 192 a 256 bitov. Varianty AES používajú rozdielne počty kôl na základe dĺžky kľúčov. AES-128 má 10 kôl, AES-192 12 kôl a AES-256 14 kôl. Každé kolo je zložené zo 4 operácií: AddRoundKey (pripočítanie podkľúča), SubBytes (substitúcia bajtov), ShiftRows (posun riadkov), MixColumns (miešanie stĺpcov). Šifra AES môže pracovať v rôznych módoch. Módy ECB (Electronic Code Book), CBC, OFB (Output FeedBack), CTR (Counter Mode) a XTS, poskytujú dôvernosť, ale neposkytujú ochranu pred náhodnými zmenami alebo škodlivými neoprávnenými manipuláciami. Režimy, ktoré kombinujú dôvernosť a integritu údajov sa označujú ako módy overeného šifrovania. Sú to napríklad CCM (Counter Mode Cipher Block Chaining Message Authentication Code Protocol), GCM (Galois/Counter Mode), CWC (Carter–Wegman CTR mode), OCB (Offset Codebook Mode). Budeme sa podrobnejšie zaoberať hlavne módom CBC a XTS [11], [12].

CBC

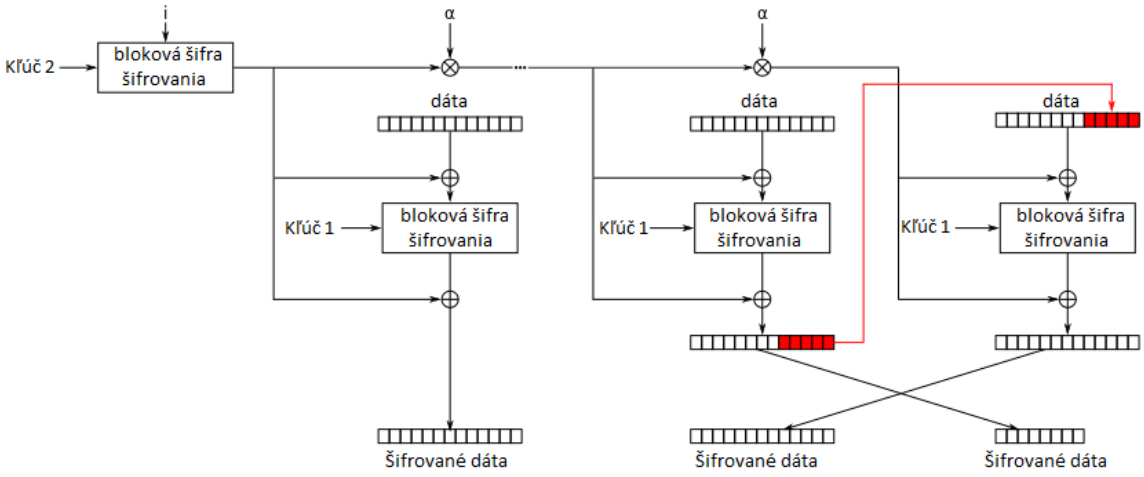
V tomto móde na každý blok dát je aplikovaná operácia XOR (eXclusive OR) s predchádzajúcim blokom šifrovaného textu ešte pred šifrovaním (Obr. 1.1). Tento krok dáva istotu, že rovnaké údaje v rôznych sektoroch nám dajú odlišný výstup po šifrovaní. Na prvý blok v tomto móde je aplikovaná operácia XOR s náhodným inicializačným vektorom [11].



Obr. 1.1: CBC mód šifrovanie.

XTS

Tento mód vykonáva taktiež operáciu XOR medzi blokmi, ale pridáva dodatočný kľúč na zlepšenie permutácie (Obr. 1.2). Tento kľúč môže byť adresa sektoru alebo kombinácia adresy sektoru a jeho indexu. XTS mód je odvodený od XEX (XOR Encrypt XOR) šifry. Tento mód je najvhodnejší režim pre šifrovanie celého disku [12].



Obr. 1.2: XTS mód šifrovanie.

AES-NI

Ide o inštrukcie novej generácie pre šifru AES na vybraných procesoroch Intel. Ponúkajú úplnú podporu hardvéru na použitie šifry AES. Práve vďaka tomu ponúkajú výrazné zvýšenie výkonu v porovnaní s aktuálnymi softvérovými implementáciami.

Okrem zlepšenia výkonu zabezpečujú aj zlepšenie bezpečnosti AES. Eliminujú väčšinu časových útokov a tiež útoky na vyrovnávaciu pamäť, ktoré ohrozujú softvérové implementácie založené na tabuľkách, ktoré sa tu nepoužívajú. Navyše AES-NI zjednodušujú samotnú implementáciu AES znížením veľkosti kódu, čo pomáha znižovať riziko neúmyselného zavedenia bezpečnostných chýb, ako napríklad ťažko detekovateľné bočné kanály [13]. AES-NI vie zvýšiť výkon AES od 3 do 10-krát oproti softvérovým riešeniam [14].

1.2.2 Nástroje a funkcie operačných systémov

V tejto časti popisujeme základné nástroje a funkcie na uchovávanie citlivých informácií (dát) od najjednoduchších, ako je Access Control List (ACL), až po zložité riešenia. Dôkladne rozoberáme ich funkcionalitu a taktiež ohodnotíme ich bezpečnosť tak, že upozorníme na vyskytujúce sa nedostatky či známe zraniteľnosti.

Zoznam pre riadenie prístupu

ACL je zoznam oprávnení súvisiacich s objektom. Popisuje konkrétne prístupové práva k objektu pre isté subjekty. Právom môže byť napríklad právo čítať, zapisovať, alebo spúšťať daný objekt subjektom. V prípade operačného systému subjekty sú používatelia a objekty súbory, programy, atď. Jednotlivý používatelia môžu nastavovať prístupové práva k súborom a tým zamedziť, alebo iba obmedziť práva ostatných užívateľov na prácu s týmito objektmi. ACL sa dá ľahko využiť na vymedzenie prístupu určitým osobám k našim kryptografickým, či iným citlivým údajom. Je to minimum, čo môžeme pre zabezpečenie urobiť. Nevýhodou je, že administrátor bude mať vždy prístup k týmto objektom. Výhodou je napríklad oproti šifrovaniu ľahká implementácia a zároveň nie je potrebné použiť skoro žiaden výpočtový výkon. V operačnom systéme Windows má každý objekt v súborovom systéme NTFS (New Technology File System) svoj zoznam ACL. Ten obsahuje záznamy ACE (Access Control Entry), ktoré špecifikujú práva užívateľov, či skupín na základe jednoznačného bezpečnostného identifikátora SID (Security Identifier) [15].

Systém šifrovania súborov

Medzi základný šifrovací nástroj v OS Windows patrí Encrypting File System (EFS). Pomocou tohto nástroja je možné šifrovať súbory a zložky užívateľa. Využíva symetrickú aj asymetrickú kryptografiu. Užívateľské dáta sa šifrujú symetrickou šifrou AES. V minulosti (do Windows XP service pack 1) sa využívala šifra DESX (DES s prídavným XORom), ktorá je už v dnešnej dobe zastaraná a nie je kryptograficky bezpečná. EFS funguje iba v prípade využitia súborového systému New Technology

File System (NTFS). Zašifrované súbory spoznáme podľa zeleného fontu názvov súborov alebo zložiek. V základnom nastavení je veľkosť kľúčov 256 bitov, a je použitý mód CBC.

Fungovanie EFS sa líši od verzie NTFS, ale aj od rôznych iných faktorov, ako je napríklad doména. Pre vysvetlenie fungovania budeme popisovať počítač s jedným užívateľom mimo domény s verziou NTFS 3 a vyššou. V takomto prípade sú súbory šifrované v troch krokoch. Prvý je, že sa vygeneruje náhodný File Encryption Key (FEK), ktorý sa použije na šifrovanie súboru pomocou AES. Tento kľúč sa potom zašifruje verejným kľúčom s asymetrickým párom kľúčov (RSA) a následne sa uloží do metadát súboru. Asymetrický pár kľúčov je uložený na disku v šifrovanej forme využívajúc známy kľúč a to DPAPI Master Key (Data Protection application programming interface).

DPAPI kľúč je odvodený od kombinácie užívateľovho NTLM (NT LAN Manager) hesla a SID (Security Identifier), čo je jedinečný identifikátor pre užívateľa. NTLM heslom myslíme prihlasovacie heslo, nie jeho hash. Hlavný kľúč EFS sa odvodzuje pomocou funkcie Password Based Key Derivation Function (PBKDF2). Tento hlavný kľúč sa používa iba na uchovanie EFS asymetrického páru kľúčov a taktiež na všetky údaje, ktoré sú chránené prostredníctvom DPAPI [16], [17].

Zraniteľnosti EFS

Dáta šifrované pomocou EFS sa zdajú byť dobre bezpečne chránené. No nie je to celkom tak. Pretože DPAPI kľúč sme odvodili od užívateľského NTLM hesla, prelomením hashu hesla (pomocou rainbow tables) dostávame informáciu, ktorú potrebujeme na odvodenie kľúča DPAPI, čo nám umožní zistiť EFS asymetrický pár kľúčov (RSA). Táto operácia znamená, že máme možnosť ďalej dešifrovať EFS do takej miery, kde získame FEK, ktorým následne môžeme dešifrovať výsledné dáta (súbory, priečinky) [18].

Jednou z možností, ako tomu zabrániť, je export EFS certifikátu a privátneho kľúča do PFX (Personal Information Exchange) súboru [19], ktorý je zabezpečený separátnym heslom, čo znamená ťažšie prelomenie. Tento krok ale znamená, že musíme vymazať PFX súbor z lokálneho úložiska kľúčov, keď ho nepoužívame, a následne ho vždy importovať, pokiaľ budeme chcieť mať prístup k zašifrovaným dátam. Tento spôsob nie je veľmi praktický, hlavne ak máme zašifrovaný celý adresár. Export EFS certifikátu nám ale zabezpečí, že budeme mať uloženú separátnu zálohu tohto certifikátu a pripraví nás tak proti ďalšiemu typu útoku. Útočník sa nemusí dát chcieť len zmocniť, niekedy stačí, aby nejako uškodil užívateľovi. Jednou z vlastností EFS je, že pri zmene užívateľského hesla sa znemožní prístup k zašifrovaným dátam. Ak sa útočník zmocní administrátorského konta, môže nám zmeniť heslo, a tým pádom

nemáme prístup k zašifrovaným dátam. A práve tu nám pomôže naša záloha certifikátu v súbore PFX, odkiaľ si môžeme ľahko kľúč naimportovať a získať späť prístup k dátam.

Ďalšou možnosťou je, že pokiaľ využívame šifrovanie disku, alebo partície pomocou BitLockeru, náš DPAPI Master Key môže byť taktiež zašifrovaný týmto spôsobom, a tým pádom je zabezpečený zariadením, ako je napríklad TPM (Trusted Platform Module). Toto zariadenie môže taktiež vyžadovať dodatočnú autentizáciu, ktorou je napríklad smart karta, ešte predtým, ako budú kľúče odomknuté. Čo znamená ďalšie rapídne zvýšenie bezpečnosti. Tak isto môžeme uložiť DPAPI Master Key, alebo dvojicu asymetrických kľúčov EFS na čipovú kartu, alebo USB disk, čím sa vyhneme ich ukladaniu na disk a vyriešime problém s prelomením NTLM hashov.

Windows stále využíva NTLM hashe, ktoré sú jedny z najrýchlejšie prelomiteľných, pretože nepoužívaju takzvané solenie (pridávanie náhodneho reťazca znakov k heslu pred hashovaním) hesla. Ale aj napriek tomuto faktoru, ak používame bezpečné heslo pozostávajúce z aspoň 12 a viac znakov, ktoré spĺňa náhodnosť, obsahuje čísla a špeciálne znaky, je skoro nemožné, že útočník takýto hash prelomí [18], [20].

Mac OS Keychain

Keychain je systémový nástroj pre macOS a iOS operačné systémy, ktorý dokáže bezpečne držať heslá, privátne kľúče, certifikáty, zabezpečené poznámky v šifrovaných súboroch na zariadení. Pri mobilných zariadeniach je Keychain synchronizovaný s iCloudom. V systéme macOS Sierra sú súbory s kľúčovým materiálom uložené v ∼/Library/Keychains, /Library/Keychains/ a /Network/Library/Keychains/. Tieto súbory sa zobrazujú a upravujú prostredníctvom aplikácie s názvom Keychain Access, ktorá sa nachádza v Utilities v priečinku Applications. Tie isté úkony môžeme vykonávať aj prostredníctvom príkazového riadku [21].

Keychain súbory ukladajú niekoľko dátových polí ako názov, adresa URL (Uniform Resource Locator), poznámky a heslá. Dáta, názvy, URL a iné, sú uložené v bežnom texte, ale heslá a zabezpečné poznámky sú šifrované pomocou Triple DES. Všetky súbory (položky) musia byť indexované a tieto indexy musia byť verejné pre potreby vyhľadávania položky v zoznamoch prístupu [22].

Zabezpečené poznámky v Keychain sa môžu použiť na uloženie rôznych citlivých informácií ako identifikačné čísla bankových účtov, osobné údaje, ale hlavne kryptografické kľúče [23].

Zmena hesla v Keychain

V základnom stave je heslo od kľúčenky rovnaké ako užívateľské heslo, ktoré sa používa na prihlásenie do zariadenia. Ak sa užívateľské heslo zmení, tak sa automaticky

zmení aj heslo Keychain tak, aby sa zhodovali. Avšak, ak heslo zmení správca, na zmenu hesla kľúčenky budeme vyzvaní pri najbližšom prihlásení. Musíme však najskôr zadať staré heslo od Keychain. Ak si toto predchádzajúce heslo nepamätáme, môžeme resetovať kľúčenku, ale tým pádom prídeme o všetky uložené heslá a kľúče. Ak si vytvoríme novú kľúčenku, ktorá nemá rovnaké heslo ako užívateľ, budeme vyzvaní na jeho zadanie vždy, ak bude nejaká aplikácia potrebovať do nej prístup a kľúčenka bude uzamknutá [24].

ACL prístupy v Keychain

Každá položka (napríklad súkromný kľúč) v Keychain má asociovanú inštanciu prístupu, ktorá obsahuje prístupový zoznam ACL. Položky v tomto zozname zase obsahujú pole operácií a súbor aplikácií, ktoré majú prístup a dokážu z Keychain položkou narábať. Keď sa aplikácia pokúsi získať prístup k kľúču na konkrétny účel (napríklad privátny kľúč na podpísanie dokumentu), systém hľadá v zozname ACL túto aplikáciu a či má povolenie na konkrétnu operáciu. Ak túto aplikáciu vôbec nenájde, odmietne prístup, ak ju nájde, ale nemá povolenie na danú operáciu, tak odmietne prístup, ale dá aplikácií vedieť, že má skúsiť niečo iné. Ak sa aplikácia nenachádza medzi dôveryhodnými aplikáciami, systém vyzve užívateľa na potvrdenie. Ten môže aplikácii odmietnuť prístup, alebo ho udeliť, a to jednorázovo, alebo natrvalo. Natrvalo znamená, že aplikácia sa zapíše do zoznamu dôveryhodných aplikácií pre túto položku a v budúcnosti už nebude pýtať prístup od používateľa [25].

Bezpečnostné hrozby v Keychain

Bezpečnostné hrozby vychádzajú hlavne z konania používateľa. Pokiaľ si dobre neoverí, ktorým aplikáciám a na ktoré operácie dáva povolenie na prístup do Keychain, môže sa ľahko stať, že prístup k citlivým údajom (ako napríklad súkromné kľúče) nadobudnú aj aplikácie, ktoré ich nepotrebujú pri svojej činnosti a tým pádom môžu byť zneužité. Taktiež voľba hesla a jeho neopakovanie sa v iných aplikáciách, či zariadeniach, zohráva svoju rolu v bezpečnosti Keychain, ako aj celého operačného systému.

GNOME Keyring

GNOME (GNU Network Object Model Environment) Keyring je kolekcia komponentov (program) v systéme Linux v prostredí GNOME, ktorá ukladá tajomstvá, heslá, kľúče, certifikáty a sprístupňuje ich aplikáciám. Na zariadení beží ako démon v relácii, podobne ako ssh-agent. Program môže spravovať niekoľko kľúčových zväzkov, každý s vlastným master heslom, alebo kľúčový zväzok, ktorý je uchovávaný výhradne počas určitej relácie (potom sa zabudne) a nikdy sa nezapíše na disk [26].

GNOME Keyring tiež zaisťuje, že tajomstvá požívané v pamäti sú uzamknuté vo fyzickej RAM, aby sa zabránilo ich vyňatiu (stránkovaniu) do swap súboru na disku [27].

Všetky utajené údaje sú šifrované pomocou bezpečného AES s veľkosťou kľúča 128 bitov, preto nemôžu byť dešifrované pri absencii hesla. Na hash hesla sa používa algoritmus SHA-256. Ak je Keyring integrovaný s užívateľským loginom, tak bezpečné úložisko tajomstiev sa odomkne vždy, keď sa užívateľ prihlási. Vymazanie celého obsahu Keyringu je možné vymazaním súborov login.keyring a user.keystore z umiestnenia /home/{username}/.local/share/keyrings/ [26].

Ochrana a zraniteľnosti GNOME Keyring

GNOME Keyring nás chráni pred krádežou kľúčov, alebo hesiel z disku pri vypnutom, alebo inaktívnom zariadení, ďalej pred čítaním tajomstiev z pamäte po odhlásení používateľa, alebo pred ich presunom do swap oblasti na disku, a taktiež nás chráni pred čítaním keyring súboru iným používateľom, či pred prístupom k nášmu keyring démonu nežiaducou osobou (používateľom).

Stále ale existuje množstvo zraniteľností, ktoré sa dajú zneužiť. Sú možné útoky, keď nie je keyring uzamknutý, ako napríklad, keď je počítač v stave hibernácie (dlhodobý spánok), alebo keď sa užívateľ neodhlási korektne napríklad pri výpadku prúdu. Ďalej môže byť praktikované hádanie hlavného (master) hesla. Aj keď je toto heslo najskôr zahashované a až následne sa používa na šifrovanie keyringu, je tu stále možnosť, že útočník hrubou silou uhádne heslo, hlavne v prípade použitia slabého hesla. Pokiaľ je hlavné heslo to isté ako prihlasovacie heslo používateľa do účtu, tak je možné použiť nástroj John The Ripper na prelomenie hesla a následne získanie prístupu ku kryptografickým kľúčom [28], [29].

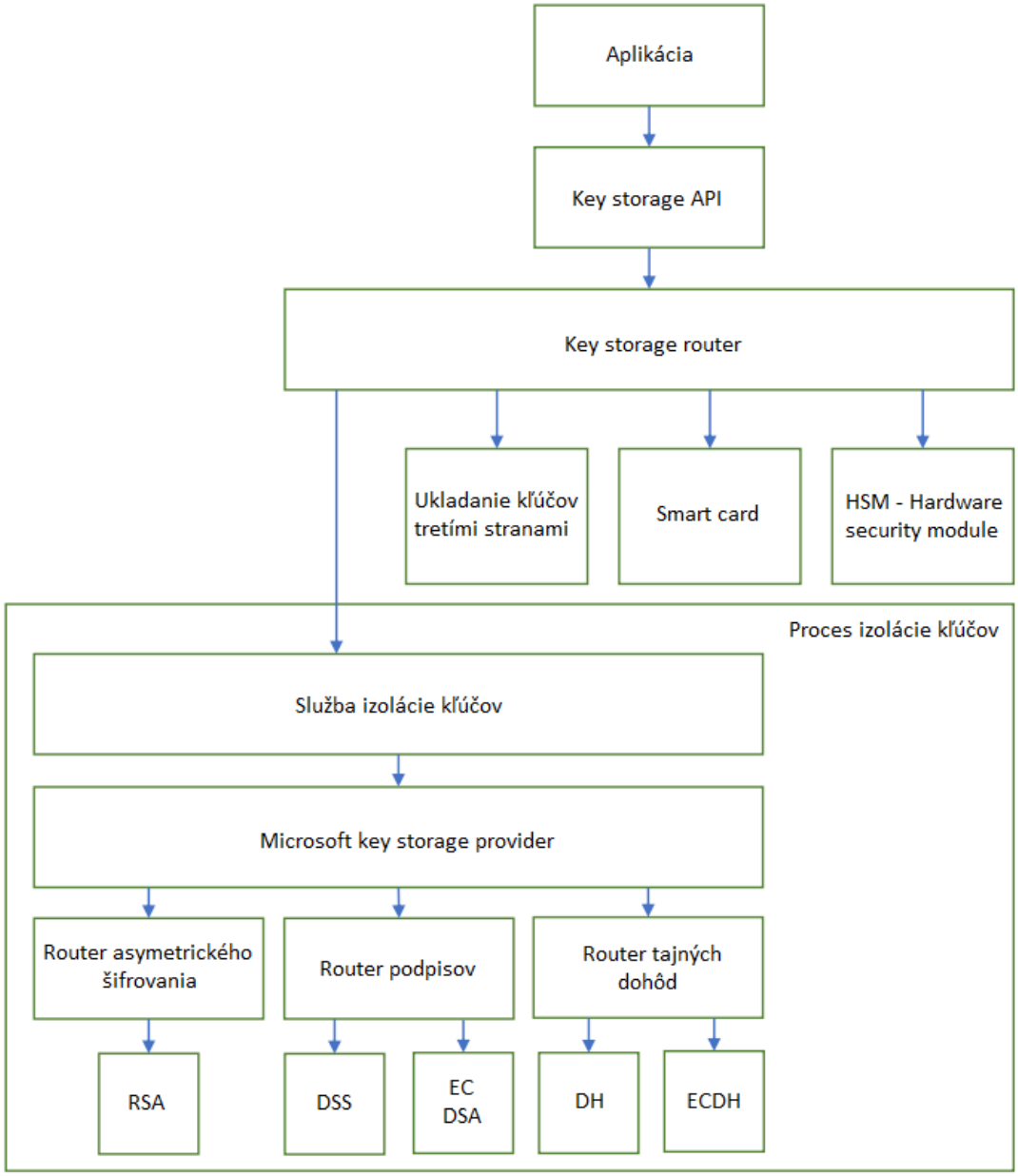
Kryptografické aplikačné rozhranie novej generácie

Cryptography API: Next Generation (CNG) je dlhodobou náhradou za CrytoAPI, čo je aplikačné programovacie rozhranie, ktoré umožňuje vývojárom aplikácií pridávať autentizáciu, kódovanie a šifrovanie do aplikácií bežiacich pod systémom Windows. CNG je navrhnutý tak, aby zabezpečoval čo najširšiu škálu kryptografických bezpečnostných opatrení [30]. Ďalej sme sa venovali hlavne systému ukladania a vyhľadávania kľúčov znázorneného v schéme (Obr. 1.3).

Ukladanie a vyhľadávanie kľúčov

CNG poskytuje model pre ukladanie súkromných kľúčov, ktorý umožňuje prispôsobenie sa súčasným a budúcim požiadavkám na vytváranie aplikácií, ktoré využívajú funkcie kryptografie, ako je šifrovanie verejných, alebo súkromných kľúčov, ako aj

požiadavky na ukladanie kľúčových materiálov. Keď aplikácie chcú pristupovať ku Key Storage Providerom (KSP), tak musia prejsť cez takzvaný Key Storage Router, ktorý utajuje detaily napríklad o izolácii kľúčov, a to od samotnej aplikácie a tiež od samotného poskytovateľa úložiska [31], [32].



Obr. 1.3: Návrh a jednotlivé funkcie architektúry izolácie kľúčov CNG.

Na splnenie požiadaviek Common Criteria musia byť kľúče s dlhou životnosťou izolované tak, že sa do aplikačného procesu nikdy nedostanú. CNG podporuje ukladanie asymetrických súkromných kľúčov pomocou Microsoft softvéru KSP, ktorý je súčasťou operačného systému od verzie Windows Vista. KSP tretích strán nie sú načítané v službe izolácie kľúčov (LSA proces), ale iba výhradne Microsoft KSP.

Proces LSA (Local Security Authority) sa používa ako kľúčový proces izolácie na maximalizáciu výkonu. Celý prístup k súkromným kľúčom prechádza cez Key Storage Router, ktorý poskytuje komplexnú sadu funkcií pre správu a používanie súkromných kľúčov.

CNG ukladá verejnú časť uloženého kľúča oddelene od súkromnej časti. Verejná časť z páru kľúčov je tiež zachovaná v službe izolácie kľúčov a je prístupná pomocou LRPC (Local Remote Procedure Call). Key Storage Router využíva LRPC pri volaní do procesu izolácie kľúčov. Každý prístup k súkromným kľúčom prechádza cez smerovač súkromných kľúčov a kontroluje ho CNG, ktoré taktiež podporuje širokú škálu hardwarových úložných zariadení. V každom prípade je rozhranie so všetkými týmito pamäťovými zariadeniami totožné. Obsahuje taktiež funkcie na vykonávanie rôznych operácií súkromných kľúčov, ako aj funkcie, ktoré sa týkaju ukladania kľúčov a ich správy [32]. CNG poskytuje súbor rozhraní API, ktoré sa používajú na vytváranie, ukladanie a vyhľadávanie kryptografických kľúčov.
CNG podporuje tieto typy kľúčov, ktorých parametre sú popísané v tabuľke 1.1 :

- Diffie-Hellman,
- Digital Signature Algorithm (DSA, FIPS 186-2),
- RSA (PKCS #1),
- Several legacy (CryptoAPI),
- Elliptic Curve Cryptography.

Tab. 1.1: Podporované algoritmy CNG [30].

Algoritmus	**Kľúč/Hash dĺžka v bitoch (bits)**
RSA	512 až 16384
DH	512 až 16384
DSA	512 až 1024
ECDSA	P-256, P-384, P-521 (NIST Curves)
ECDH	P-256, P-384, P-521 (NIST Curves)

Funkcia CryptProtectMemory

Táto funkcia šifruje pamäť na platforme Windows, aby zabránila voľnému prezeraniu citlivých informácií v procese osobám, ktoré im nenáležia. Napríklad, keď používame kryptografické kľúče v procese, tie sa nemusia používať stále a potom sú ponechané v pamäti, alebo presunuté z pamäte do swapového súboru. Tieto kľúče môžu byť tým pádom voľne prezerané. Tiež je tu hrozba, že niekto odpojí zariadenie od elektrickej energie a tým pádom sa nevypne korektne. Kľúče dočasne uložené v swap

súbore môže útočník ľahko extrahovať. Týmto všetkým problémom funkcia CryptProtectMemory zabraňuje.Toto riešenie nám prináša vyššiu bezpečnosť, ale na druhej strane zvyšuje nároky na výpočtový výkon [33].

1.2.3 Zhodnotenie bezpečnosti vybraných nástrojov

Ohodnotili sme bezpečnosť vybraných nástrojov známkami veľmi dobrý, dobrý, uspokojivý alebo neuspokojivý. Treba brať ohľad na to, že všetky nátroje sú založené hlavne na bezpečnostnom texte (hesle), ktoré zadáva používateľ. Mali by sme teda používať silné heslá, ktoré sú jedinečné, to znamená, že nie sú používané viackrát. Hodnotenia a špecifikácie vybraných nástrojov sú zobrazené v tabuľke 1.2 spolu s programovými riešeniami.

Bezpečnosť nástroja EFS

Keďže sa kľúče, ktorými šifrujeme odvádzajú od užívateľského hesla, existuje práve typ útoku (rainbow tables), ktorý porovnáva hashe hesla (NTLM), a tým môže získať prístup k účtu a následne dešifrovať dáta. Tento typ útoku je ale možný iba pri slabých a krátkych heslách. Preto je doporučenie používať 12 a viac znakové heslo, ktoré spĺňa náhodnosť a obsahuje čísla a špeciálne znaky. Pri zmene hesla sa nedostaneme k dátam, budeme musieť importovať starý certifikát s heslom (ktorý je vždy potrebné najskôr zálohovať). Samotné šifrovanie prebieha bezpečným algoritmom AES 256 v móde CBC.

Bezpečnosť hodnotíme ako **dobrú**.

Bezpečnosť Keychain

Zabezpečené poznámky sú šifrované celkom bezpečnou šifrou 3DES. Bezpečnostné hrozby vychádzajú hlavne z konania užívateľa (nastavenie prístupu do Keychain neovereným aplikáciam, voľba hesla, atď.). Aktuálne nie sú známe žiadne bezpečnostné riziká v spojitosti s Keychain na Mac OS.

Bezpečnosť hodnotíme ako **veľmi dobrú**.

1.2.4 Programové riešenia

V tejto časti sme si priblížili najpoužívanejšie programy na šifrovanie súborov či celých diskov. Taktiež sme ohodnotili ich kryptografickú bezpečnosť.

Veracrypt

Je voľne dostupný open source program pre šifrovanie diskov, ktorý podporuje operačné systémy Windows, Mac OS X a Linux. Veracrypt pridáva vylepšenú bezpečnosť algoritmom, ktoré sa používajú na šifrovanie systémov či partícií, čím sa stávajú imúnne voči novo vylepšeným útokom hrubou silou. Veracrypt podporuje šifry ako AES, TwoFish, Serpent a iné, a ich kombinácie na šifrovanie. Kľúče sú veľkosti 256 bitov a bloky dát 128 bitov. Používa sa mód XTS. Dokáže vytvárať takzvané skryté šifrované médium vnorené v inom médiu. Taktiež podporuje celodiskové šifrovanie vrátane systémového disku. Tým je nástroj dostatočne flexibilný na šifrovanie diskov a súborov za behu pre potreby udržania špecifických informácií v bezpečí, alebo na šifrovanie celého systému, ku ktorému budú mať prístup iba oprávnení používatelia [34].

Hlavné funkcie:

- Vytvorenie virtuálneho šifrovaného disku v rámci súboru a jeho pripojenie ako reálny disk.
- Šifrovanie celej partície alebo úložiska ako napríklad USB (Universal Serial Bus) flash disk alebo pevný disk.
- Šifrovanie partície alebo disku, kde je operačný systém Windows nainštalovaný (autentizácia pred spustením systému).
- Šifrovanie je automatické, v reálnom čase a transparentné.
- Paralelizácia a zreťazené spracovanie umožňujú čítanie a zápis údajov skoro tak rýchlo, ako keby nebol disk šifrovaný.
- Šifrovanie môže byť hardvérovo zrýchlené (AES-NI).
- Poskytuje hodnoverné informácie napríklad o šifrovanom disku, aj keď na danom disku existuje skrytá partícia, alebo skrytý operačný systém o ktorom vie len používateľ. Využitie je napríklad v prípade, keď útočník núti obeť zadať heslo [35].

AesCrypt

Tento program je bezplatný open source softvér, ktorý beží na operačných systémoch Windows, Mac OSX a Linux. Sú tu tiež verzie na implementáciu do programovacích jazykov C, C++, C# a Java. Súbory zašifrované jednou platformou sú kompatibilné a môžu byť dešifrované inou platformou. AesCrypt využíva pre jednoduché používanie grafické používateľské rozhranie (GUI) [36].

Pomocou silného 256 bitového šifrovacieho algoritmu môže AES Crypt bezpečne zabezpečiť citlivé súbory. Zašifrované súbory bez znalosti hesla sú prakticky nečitateľné. Tento program je dokonalý nástroj pre každého, kto pracuje s citlivými

informáciami (napríklad šifrovacie a autentizačné kľúče), ktoré si buď chce bezpečne uchovať pred odcudzením, alebo ich nahrať na vzdialený server (cloud) v bezpečnej nečitateľnej podobe. Vďaka tomu, že je softvér open souce, ľudia dokážu kontrolovať, ako je zdrojový kód bezpečný, a ako bezpečne sú vykonávané jednotlivé operácie, čím sa zlepšuje zabezpečenie uchovávaných dát [37].

7-Zip

Je voľne dostupný open source program, ktorý archivuje súbory s vysoko kompresným pomerom. 7-Zip taktiež podporuje šifrovanie s AES algoritmom s 256 bitovým kľúčom. Tento kľúč sa generuje z užívateľsky zadaného hesla algoritmom založeným na SHA-256 (Secure Hash Algorithm) hashi. Pre zvýšenie odolnosti pred útokmi hrubou silou 7-Zip používa veľký počet opakovaní na vytvorenie šifrovacieho kľúča z hesla. Presnejšie 2^{18} (262144), čo spôsobuje výrazne zvýšené nároky na výpočtový výkon. Základom bezpečnosti ale stále zostáva sila zadaného hesla [38].

Z pohľadu užívateľa program nie je náročný na obsluhu, ale musíme upozorniť na absenciu spustenia na iných operačných systémoch. Ďalej je nevýhodou, že sa nedá šifrovanie uskutočniť zvlášť, pretože archivácia potrebuje zbytočný výpočtový výkon navyše. Taktiež veľký počet opakovaní na vytvorenie šifrovacieho kľúča z hesla zvyšuje náročnosť na výkon.

1.2.5 Bezpečnosť programových riešení

Je dôležité pre používateľa poznať, aké zabezpečenie využíva program, ktorý sa chystá použiť na ochranu citlivých dát. Ohodnotili sme bezpečnosť často používaných programov známkami veľmi dobrý, dobrý, uspokojivý alebo neuspokojivý. Hodnotenia a porovania špecifikácií daných riešení sú v tabuľke 1.2 spolu s vybranými nástrojmi.

Bezpečnosť AesCrypt

Šifra AES, ktorú používa tento program je bezpečná, ale vyskytujú sa bezpečnostné zraniteľnosti priamo v programe. Útočník môže umelo zmeniť veľkosť šifrovaného súboru. Šifrované dáta, ako také, ale nie sú v ohrození. Ďalej sa vyskytuje aj chyba, keď v príkazovom riadku nezadáme pri dešifrovaní správny súbor (s príponou .aes). Vtedy môže dôjsť k vymazaniu vstupného súboru, ktorý zadáme. O týchto chybách ale developer vie (aj vďaka open source kódu), a deklaroval, že sa ich bude snažiť opraviť.

Z pohľadu bezpečnosti hodnotíme program ako **dobrý**.

Bezpečnosť 7-Zip

Šifrovací algoritmus AES 256 je považovaný za bezpečný. Program používa veľký počet opakovaní (262144) na vytvorenie šifrovacieho kľúča z hesla pomocou hashovacieho algoritmu SHA-256.

Je však známa jedna bezpečnostná chyba v programe. Keď si zašifrovaný súbor otvoríme po zadaní hesla, program 7-Zip ukladá tento súbor dočasne do priečinku *Temp* a pri zatvorení programu tieto dáta vymazáva. Ak náhodou zatvoríme program skôr ako daný súbor, program tento súbor nevymaže a ostane v priečinku *Temp* až do ďalšieho zatvorenia programu v nešifrovanej forme.

Bezpečnosť hodnotíme ako **uspokojivú**.

Bezpečnosť VeraCrypt

Na šifrovanie diskov je na výber množstvo algoritmov. Pri vytvorení sa disk zaplní náhodnými dátami, čo považujeme sa dobré bezpečnostné vylepšenie. Výhodou je aj možnosť vytvoriť skrytú partíciu v samotnom disku. Ide vlastne o rozdelenie úložného priestoru na 2 časti, ktoré nevidíme na prvý pohľad. Podľa toho, aké heslo zadáme, do tej časti sa dostaneme. V dnešnej dobe neexistujú známe zraniteľnosti v programe, ktoré by dokázali útočníkom prístup k citlivým zašifrovaným dátam.

Bezpečnosť hodnotíme ako **veľmi dobrú**.

Tab. 1.2: Porovnanie jednotlivých programových riešení a vybraných nástrojov.

	AesCrypt	**7-Zip**	**VeraCrypt**	**EFS**	**MacOS Keychain**
Bezpečnostné hodnotenie	dobrý	uspokojivý	veľmi dobrý	dobrý	veľmi dobrý
Šifrovacie algoritmy	AES 256	AES 256	AES 256 TwoFish Serpent atď.	AES 256 mód CBC	3DES
Platformy	Windows MacOS Linux	Windows	Windows MacOS Linux	Windows	MacOS
Dostupnosť softvéru	zadarmo	zadarmo	zadarmo	súčasťou Windows	súčasťou MacOS
Open source	áno	áno	áno	nie	nie

2 Analýza možností uloženia kľúčov na zariadeniach využívaných v internete vecí

IoT je sieť zariadení, ktoré sú medzi sebou prepojené a dokážu komunikovať a vymieňať si dáta. IoT pozostáva z veľkej variácie zariadení, od primitívnych senzorov, cez tablety, televízie, až po počítače či servery, ktoré riadia primitívnejšie zariadenia. Preto je potrebné určiť najvhodnejšiu technológiu pre šifrovanie, autentizáciu, ukladanie kľúčov, atď., na základe výkonu a použitia týchto zariadení. Tieto zariadenia sú obzvlášť náchylné na útoky, pretože sa často zhlukujú do väčších celkov a pracujú s dátami, ktoré sa dajú ľahko zneužiť. Preto netreba brať na ľahkú váhu zabezpečenie ani toho najmenšieho prvku v takejto sieti, pretože práve ten môže byť vstupnou bránou pre útočníkov do celej siete zariadení [39].

2.1 Rozdelenie IoT zariadení

Existuje viacero štandardov, ktoré rozdeľujú IoT zariadenia. Ukážeme si prienik klasifikácií zariadení podľa IETF (Internet Engineering Task Force) a ENSIA (European Union Agency for Network and Information Security) [40]. Tieto klasifikácie sú založené na RFC 7228 (Request For Comments), a rozdeľujú zariadenia do troch až štyroch tried podľa obmedzení nákladov, rozmerov, a ďalších faktorov ako kapacita pamäte RAM a veľkosť úložného priestoru (Flash pamäť) [41], ktoré vidíme nižšie v tabuľke 2.1.

Tab. 2.1: Rozdelenie IoT zariadení.

Klasifikácia		**Kapacita pamäte (RAM)**	**Kapacita úložiska (Flash)**
Zariadenia s obmedzenými zdrojmi	Trieda 0	<<10 KiB	<<100 KiB
	Trieda 1	∼10 KiB	∼100 KiB
	Trieda 2	∼50 KiB	∼250 KiB
Vysoko výkonné zariadenia		>>50 KiB	>>250 KiB

Na základe týchto klasifikácií, môžeme zariadenia roztriediť do štyroch tried [42].

Trieda 0
Charakteristika: Architektúra vytvorená s minimálnymi komunikačnými a konfiguračnými súbormi.
Procesor: žiadny.
Operačný systém: Frimware.

Typ zariadenia: Lacné senzory používajúce predkonfigurovaný súbor.
Podporovaný šifrovací algoritmus: Skoro nemožná podpora šifrovacích algoritmov.

Trieda 1
Charakteristika: Použitie ľahkého protokolu ako napríklad CoAP (Constrained Application Protocol), je potrebné obmedziť spotrebu pamäte a kódu.
Procesor: Atmega 128 (8 bitov) alebo MSP430 (16 bitov).
Operačný systém: TinyOS, Contiki, NanoQPlus, a ďalšie.
Typ zariadenia: Zariadenia založené na 8 až 16 bitovom procesore ako napríklad fitness pomôcky, tlakomery, jednoduché meteostanice, jednoduché smart home zariadenia, atď.
Podporovaný šifrovací algoritmus: AES 128/192/256, RSA 1024/2048, SHA-1 256/384.

Trieda 2
Charakteristika: Podpora existujúcich protokolov bez skoro žiadnych limitov na zdroje.
Procesor: ARM-Cortex A (32 bitov).
Operačný systém: Linux, Windows 10 IoT, atď.
Typ zariadenia: zariadenia založené na 32 bitovom procesore ako napríklad kamery, inteligentné merače, výrobné roboty, atď.
Podporovaný šifrovací algoritmus: Výkonné algoritmy ako AES 128/192/256, RSA 1024/2048, SHA-1 256/384.

Trieda 3
Charakteristika: Rovnaký výkonnostný level ako počítače a servery bez skoro žiadneho obmedzenia zdrojov.
Procesor: Všetky dostupné procesory ktoré sa využívajú v IT zariadeniach.
Operačný systém: Android, iOS, Linux, Windows 10 IoT, atď.
Typ zariadenia: Smartphony, tablet PC, smart TV, smart hub, atď.
Podporovaný šifrovací algoritmus: Výkonné algoritmy ako AES 128/192/256, RSA 1024/2048, SHA-1 256/384.

2.2 IoT zariadenia trieda 0

Tieto zariadenia sú vlastne senzory, ktoré sa väčšinou zhlukujú do takzvaných senzorových sietí, uzlov. V odvetví IoT výrobcovia alebo OEM distribútori implementujú

vo väčšine prípadov do pamäte týchto zariadení rôzne certifikáty, šifrovacie alebo autentizačné kľúče, či takzvaný master key od ktorého sa ostatné kľúče odvodzujú. Tu nastáva problém, ako výrobcovia narábajú s týmito kľúčmi a certifikátmi, a tiež ako ich generujú a uchovávajú, pretože aj tento faktor veľmi ovplyvňuje našu bezpečnosť.

Celý výrobný proces by tak mal byť prísne sledovaný a zdokumentovaný. Riziko prístupu nepovoleným osobám ku kľúčom je veľké. Tým pádom môžu byť naše šifrovacie či autentizačné kľúče, na ktorých stojí bezpečnosť našich senzorov, okopírované a zneužité, čím môžeme spochybniť bezpečnosť celej infraštruktúry. Tento problém sa dá riešiť pomocou implementovania HSM do výrobného procesu, ktorý bezpečne vygeneruje a hlavne bezpečne uchová kritické kľúče. Toto je prvý krok k bezpečnosti uloženia a zabezpečenia integrity kľúčov v odvetví IoT [43].

2.2.1 ZigBee

Dôležitá je aj následná implementácia a celková štruktúra a funkčnosť štandardov a protokolov. Štandard ZigBee, ktorý je jedným z najrozšírenejších kompletným riešením pre IoT, od mesh sietí, až po univerzálny jazyk, umožňuje inteligentným objektom spolupracovať. Mesh sieť je sieť, ktorá pozostáva z uzlov, ktoré sú na seba priamo napojené a efektívne komunikujú.

ZigBee protokol je založený na modeli otvorenej dôvery. To znamená, že všetky vrstvy protokolu si navzájom dôverujú. Preto existuje kryptografická ochrana len medzi zariadeniami. Každá vrstva je zodpovedná za bezpečnosť svojich rámcov. Bezpečnosť sietí ZigBee je založená na ich šifrovacích kľúčoch. Je možné rozlíšiť dva typy bezpečnostných kľúčov.

Sieťový kľúč (network key) slúži na zabezpečenie broadcast komunikácie. Tento kľúč je 128 bitový a je zdieľaný medzi všetkými zariadeniami v sieti. Zariadenie musí získať sieťový kľúč prostredníctvom prenosu, alebo musí byť predinštalovaný vo výrobnom procese. Prepojovací kľúč (link key) sa používa na zabezpečenie komunikácie unicast. Tento kľúč je taktiež 128 bitový, ale je zdieľaný iba medzi dvomi zariadeniami. Tieto kľúče sa získavajú taktiež z prenosu, alebo počas výrobného procesu predinštaláciou, alebo vytvorením kľúča.

Prepojovací kľúč je odvodený od hlavného kľúča (master key). Zariadenie musí získať hlavný kľúč vzdialene prenosom, ale častejšie je vložený priamo pri výrobnom procese. Hlavný kľúč sa používa výhradne na účel vytvorenia zodpovedajúcich prepojovacích kľúčov [44].

Bezpečnosť ZigBee

Bezpečnosť ZigBee je založená na predpoklade, že kľúče sú bezpečne uložené a zariadenia majú prednahraté symetrické kľúče. Kvôli nízko nákladovému charakteru

niektorých typov zariadení, ako sú spínače svetla, snímače teploty, atď., ako aj potrebe nízkej spotreby elektrickej energie, sú kľúče uložené v pamäti v nešifrovanej podobe. Tieto zariadenia taktiež nemajú hardvér zabezpečený proti neoprávnenému zásahu. Takže ak útočník nadobudne možnosť fyzického prístupu k takémuto zariadeniu, má možnosť pristúpiť k tajným kľúčom a iným privilegovaným informáciam.

ZigBee štandard je celkom bezpečný, problém nastáva iba pri lacných a malo výkonných snímačoch. Je otázka, či nízka cena je tou správnou cestou v porovnaní s dostatočnou bezpečnosťou kľúčov. Ak by boli zariadenia odolné voči falšovaniu a neželanému fyzickému prístupu, mohli by detekovať neoprávnenú manipuláciu a automaticky vymazali citlivé informácie vrátane bezpečnostných kľúčov. Malé zvýšenie ceny senzorov by tak zvýšilo rapídne bezpečnosť celého systému [45].

2.2.2 Z-Wave

Z-Wave je bezdrôtový komunikačný protokol primárne určený pre domácu automatizáciu v prostredí IoT. Je to sieť uzlov využívajúca málo energeticky náročné radio vlny na komunikáciu typu spotrebič - spotrebič a skupina spotrebičov s inou skupinou. V podstate je to technológia podobná ako pri už popisovanom protokole ZigBee.

Pri nízko nákladových zariadeniach je pri tejto technológií podobný problém pri uchovávaní kľúčov na týchto zariadeniach ako v prípade ZigBee. V dokumentácii k bezpečnosti Z-Wave, už aj pri jej vylepšenej verzií S2 výrobca udáva, že je tu teoretická možnosť, že skúsená osoba, ak má fyzický prístup k zariadeniu a dokáže rozobrať tento senzor, tak môže vyňať a skopírovať sieťové kľúče tým, že sa útočník pripojí priamo na obvody pamäte senzora [46].

Najnovšia technológia čipov, ako je Silicon Labs Z-Wave 500-series čip [47], pozostáva z toho, že tento čip obsahuje typ pamäte, ktorú je možné čítať iba firmvérom bežiacom na tom danom čipe. Tým sa zabráni neoprávneným prístupom do pamäte. Táto pamäť sa automaticky vymaže, ak je nový firmware nahraný na čip cez externé programovacie rozhranie.

2.3 IoT zariadenia trieda 1

Tieto zariadenia sú založené na 8 alebo 16 bitovom procesore. Používajú sa napríklad v senzoroch alebo smart home zariadeniach na náročnejšie výpočty a operácie v porovnaní s triedou 0. Tieto zariadenia podporujú širokú škálu šifrovacích algoritmov a rôznych protokolov, preto sa predpokladá väčšie množstvo citlivých údajov, ako sú napríklad kryptografické kľúče, ktoré treba uchovať v bezpečí. V poslednom čase sa hlavne pre potreby zabezpečenia odvetvia IoT začali vyrábať špeciálne čipy

alebo pamäte, ktoré buď pomáhajú k bezpečnosti mikrokontrolérom alebo sú priamo implementované do nich.

2.3.1 Ochrana zónami

Použitie polovodičových materiálov v zariadeniach IoT dokáže zvýšiť úroveň hardvérovej bezpečnosti. Rodina čipov MSP430FR využíva výhody a flexibilitu FRAM (Ferroelectric Random Access Memory) pamäte [48]. Tento typ pamäte už sám o sebe zlepšuje bezpečnosť uloženia kryptografických kľúčov v mikrokontroléri. Zápis, alebo čítanie z tejto pamäte potrebuje na svoju prácu veľmi malé napätie a to 1,5 V v porovnaní s Flash pamäťou, ktorá potrebuje 10-14 V. Tento fakt sa odráža do nízkej spotreby energie, a tým sťažuje útočníkom, ktorý fyzicky alebo elektronicky detekujú, alebo monitorujú údaje v zariadení, zachytiť moment, kedy sa údaje zapisujú alebo čítajú z tejto pamäte. Táto pamäť je teda odolná voči DPA (differential power analysis), a taktiež vďaka svojej feroelektrickej architektúre aj voči pôsobeniu silného magnetického poľa. Tým pádom sú naše kľúče dobre chránené aj pred násilným fyzickým vymazaním pomocou magnetu [49].

Pomocou tohto typu mikrokontroléru môžu vývojári chrániť kritické údaje, ako sú kryptografické kľúče tým, že ich umiestnia do takzvaných zón IPE (IP Encapsulation). Keď údaje uložíme do nejakej zóny, môže k nim pristupovať iba kód v rámci rovnakej zóny. Pomocou tohto systému zón môžu vývojári umiestniť kryptografické kľúče spolu s ich algoritmami v rovnakej zóne IPE a zabrániť tak akémukoľvek externému prístupu pomocou JTAG debuggrov, bootloaderov, DMA (Direct Memory Access) [50], alebo aj priamemu čítaniu registrov.

Pre systémy IoT, kde sa vyžaduje používanie mikrokontrolérov bez integrovanej funkcie správy kľúčov, čip ATECC508A CryptoAuthentication IC ponúka efektívne riešenie [51]. Za hlavný mikrokontrolér tento čip zabezpečuje mechaniku výmeny kľúčov a hlavne bezpečné uloženie kľúčov. Tento čip dokáže tieto funkcie vykonávať po pripojení k akémukoľvek mikrokontroléru pomocou dvojvodičového hardvérového rozhrania.

Tento typ zariadení je vhodný na dynamické vytváranie, mazanie a uchovávanie tajných kľúčov. Ale nie všetky IoT aplikácie fungujú na tomto type bezpečnostnej politiky. Niektoré závisia od trvalého uloženia kryptografického kľúča, alebo iného tajomstva implementovaného priamo vo výrobe, alebo vývojárom pred nasadením zariadenia [52].

2.3.2 Permanentné uloženie

Pre aplikácie, ktoré vyžadujú trvalé ukladanie kľúčov, môžu dizajnéri využívať mikrokontroléry s integrovanou bezpečnou jednorázovo programovateľnou pamäťou OTP

(one-time programmable) [50]. Táto pamäť môže byť naprogramovaná buď v továrni, alebo tesne pred použitím. Po prvom zapísaní sa však dáta nedajú vymazať ani softvérovým príkazom, alebo hardvérovým resetom, či vymazať preprogramovaním zariadenia. Táto pamäť sa využíva tým spôsobom, že kľúč v nej uložený sa používa na kryptografické úkony priamo, alebo sa od neho odvodzujú ostatné kľúče. Mikrokontrolér nemá žiadne inštrukcie na čítanie údajov z tejto pamäte do akéhokoľvek úžívateľsky prístupného pamäťového priestoru v každom operačnom móde. Jediný prístup má priamo kryptografický modul, ktorý vykonáva šifrovacie, či iné funkcie.

2.3.3 Fyzicky neklonovateľné funkcie

Technológia Physically Unclonable Functions (PUF) reprezentuje zmenu formy ochrany kľúčov. Namiesto ukladania kľúčov (kde sú náchylné na fyzický útok), sú kľúče odvodené od jedinečných fyzikálnych vlastností, napríklad v pamäti SRAM (Static Random Access Memory) na čipe a existujú iba keď je čip napájaný. To znamená, že radšej, ako bezpečne ukladať súkromný kľúč, môžeme rovnaký kľúč vygenerovať znova a znova (po celú životnosť zariadenia) na požiadanie. Použitím PUF založeného na SRAM sa zaručuje, že kľúče sú jedinečné, pretože využívajú rozdielnosti tranzistorov obsiahnutých v pamäti SRAM. Technológia PUF v spojení s TEE (kapitola 2.4.4) predstavuje atraktívne riešenie pre potreby nízkonákladovej a ľahko integrovanej bezpečnej kľúčovej ochrany. Zároveň PUF v spojení s PKI predstavuje komplexné riešenie totožnosti [53].

2.4 IoT zariadenia trieda 2

Zariadenia IoT triedy 2 sú založené na 32 bitových procesoroch architektúry ARM, ktoré dokážu splniť požiadavky aj pre zložitejšie operačné systémy, výkonné algoritmy a podporujú skoro všetky dostupné komunikačné protokoly. Využitie týchto zariadení je napríklad v priemyselných robotoch, inteligentných meračoch, či kvalitných IP kamerách.

2.4.1 Zabezpečenie kľúčov na Raspberry Pi

Raspberry Pi je obľúbená počítačová platforma používaná v komerčných a priemyselných aplikáciách, a v stále rastúcom odvetví IoT. Hlavne v IoT sú tieto zariadenia často umiestnené vonku, alebo na mieste, kde nie je vylúčená fyzická manipulácia. A práve Raspberry Pi zariadenia sú vystavené veľkému riziku zneužitia údajov, pretože celý systém a aj citlivé kryptografické kľúče sú uložené na SD karte [54], ktorá môže byť jednoducho vytiahnutá, a dáta z nej skopírované.

Práve tento problém riešia takzvané bezpečnostné moduly, ktoré sa pripoja jednoducho k arduinu, či už cez rozhranie na doske, alebo prostredníctvom USB. Tieto moduly zvyčajne zaobstarávajú širokú škálu bezpečnostných funkcií, ale hlavne dokážu bezpečne vygenerovať, manažovať a uložiť kryptografické kľúče. Tento modul je odolný proti manipulácii, takže keď zdetekuje nežiadanú manipuláciu, všetky citlivé informácie automaticky vymaže [55].

2.4.2 Zabezpečenie na platforme Android

Android je open-source softvérový balík založený na Linuxe, ktorý je vytvorený pre širokú škálu zariadení. Keďže je základom platformy jadro Linuxu, tento fakt umožňuje využiť kľúčové funkcie zabezpečenia a umožňuje výrobcom zariadení vyvinúť ovládače hardvéru pre dobre známe jadro [56].

Android keystore system

Systém android umožňuje ukladať šifrovacie kľúče do kontajnera, čím sa sťažuje ich extrakcia zo zariadenia. Akonáhle sú kľúče v úložisku kľúčov, môžu byť použité na kryptografické operácie, pričom kľúčový materiál zostáva neexportovateľný. Okrem toho ponúka možnosť obmedziť, kedy a ako sa môžu používať kľúče. Tento systém na ukladanie kľúčov využíva rozhranie KeyChain, ako aj funkciu Android keystore provider, ktorá je zavedená od verzie Android 4.3 [57]. KeyChain je trieda, ktorá poskytuje prístup k súkromným kľúčom a ich príslušným certifikátom v úložisku poverení [58].

Bezpečnostné funkcie

Systém Android Keystore chráni kľúčový materiál pred neoprávneným použitím tak, že v prvom rade zmenšuje rozsah neoprávneného používania kľúčových materiálov mimo zariadenia so systémom Android tým, že zabraňuje extrakcii týchto materiálov z aplikačných procesov a zo zariadenia Android ako celku a následne ešte tým, že aplikácie určujú autorizované používanie ich kľúčov, a dokážu presadiť tieto obmedzenia mimo procesov aplikácií [57].

Prevencia extrakcie

Kľúčový materiál nikdy nevstúpi do aplikačného procesu. Keď aplikácia vykonáva kryptografické operácie pomocou kľúča z Android Keystore, v pozadí bežný text, šifrovaný text a správy, ktoré sa majú podpísať alebo overiť, sa napoja do systémového procesu, ktorý vykonáva kryptografické operácie [57]. Kľúčový materiál môže

byť viazaný na zabezpečený hardvér (Trusted Execution Environment, Secure Element) zariadenia Android. Tým pádom takto uložený kľúč nie je nikdy vystavený mimo tohto hardvéru. Ak je systém Android napadnutý, alebo útočník môže čítať vnútorné úložisko zariadenia, je možné používať hociktorú aplikáciu využívajúcu Android Keystore, ale nedokáže tieto dáta extraktovať zo zariadenia.

Android Hardware security module

Zariadenia so systémom od verzie Android 9 [57], môžu obsahovať modul StrongBox Keymaster. Ten je implementáciou modulu Keymaster HAL, ktorý sa nachádza v HSM. Tento modul obsahuje vlastný procesor, bezpečné úložisko, generátor pravých náhodných čísel, a mechanizmy detekujúce neoprávnenú manipuláciu s aplikáciami. Pri kontrole kľúčov uložených v StrongBox Keymaster, systém potvrdzuje integritu kľúča prostredníctvom TEE. StrongBox Keymaster podporuje algoritmy ako RSA (2048), AES (128, 256), ECDSA (P-256), HMAC-SHA256 (8-64), Triple DES (168) [59]. Keď importujeme kľúče pomocou triedy KeyStore je na nás, či vyberieme ako úložisko StrongBox Keymaster, alebo nie.

Oprávnené použitie kľúčov

Android Keystore umožňuje aplikáciám špecifikovať autorizované použitie ich kľúčov pri generovaní alebo importovaní týchto kľúčov. Po vygenerovaní alebo importovaní kľúča sa jeho oprávnenia nemôžu meniť. Autentizácie sú potom uplatňované pomocou Android Keystore pri každom použití kľúča. Táto funkcia sa využíva pri kryptografii (autentizovanie kľúčového algoritmu, šifrovanie, dešifrovanie, overovanie, atď.), časovom intervale platnosti (platnosť použitia kľúča), overení používateľa (kľúč môže byť použitý iba vtedy, pokiaľ bol užívateľ nedávno overený) [57], [59].

2.4.3 Uloženie kľúčov v Secure Enclave

Secure Enclave je hardvérový správca kľúčov na platforme od spoločnosti Apple (iOS, MacOS). Uloženie privátneho kľúča do kľúčenky (keychain) je skvelý spôsob ako ho zabezpečiť. Kľúčové dáta sú zašifrované na disku a prístupné iba autentizovaným aplikáciám. Ak však chceme použiť kľúč, musíme skopírovať stručnú textovú verziu do systémovej pamäte. Aj keď tento problém predstavuje malé riziko útoku, stále je tu možnosť, že dôjde ku kompromitácii aplikácie, a tým pádom aj k ohrozeniu bezpečnosti kľúča. Ako dodatočnú ochrannú vrstvu môžeme použiť Secure Enclave, kde uložíme kľúč. Secure Enclave je hardvérový správca kľúčov, ktorý je izolovaný od hlavného procesora, aby poskytoval dodatočnú úroveň zabezpečenia. Keď uložíme súkromný kľúč do Secure Enclave, nikdy s ním nebudeme môcť manipulovať, čo dosť

sťažuje ohrozenie kľúča. Namiesto toho môžeme vydať pokyn Secure Enclave, aby vytvoril kľúč, bezpečne ho uložil a vykonával s ním potrebné operácie. Zobrazuje sa iba výstup z týchto operácií, ako sú šifrované dáta, alebo výsledok overenia kryptografického podpisu. Secure Enclave zdieľa pamäť RAM s hlavným procesorom pre aplikácie, ale táto časť pamäte (známa ako TZ0) je zašifrovaná. Samotný modul obsahuje procesorové jadro, na ktorom beží vlastný mikro kernel s kapacitou pamäte 4 MB tiež nazývaným ako Secure Enclave Processor (SEP) [60].

Technológia je podobná ako pri ARM TrustZone/SecurCore, ale obsahuje proprietárny kód pre ovládanie procesoru SEP a tiež zodpovedá za generovanie kľúča UID na A9 alebo novších čipoch, ktorý chráni dáta používateľov v nečinnosti [61]. Obrovské výhody Secure Enclave nám ale prinášajú aj isté obmedzenia. Secure Enclave je hardvérová funkcia dostupná iba na zariadeniach iOS s procesorom Apple A7, alebo novším, a na MacBook Pro s Touch Bar a Touch ID. Ďalej Secure Enclave ukladá súkromné kľúče s 256 bitovou dĺžkou typu eliptických kriviek. Tieto kľúče možno použiť iba na vytváranie a overovanie kryptografických podpisov, alebo na výmenu kľúčov Diffie-Hellman. Nedajú sa importovať už existujúce kľúče. Kľúče sa musia priamo vytvoriť v Secure Enclave. Neexistuje mechanizmus na prenos kľúčových údajov do, alebo mimo tohto modulu, ktorý je základom bezpečnosti [62].

2.4.4 Bezpečné operačné prostredie

Trusted Execution Environment (TEE) je bezpečná oblasť vo vnútri hlavného procesora. Beží paralelne s operačným systémom v izolovanom prostredí. Zaručuje, že kód a uložené údaje v TEE sú chránené s ohľadom na dôvernosť a integritu. Takýto zdvojený systém je bezpečnejší než klasický REE (Rich Execution Environment), pretože využíva hardvér aj softvér na zabezpečenie citlivých dát a kódu [63]. Dôveryhodné aplikácie spustené v systéme TEE majú prístup k plnému výkonu hlavného procesora a pamäte zariadenia, zatiaľ čo izolácia hardvéru chráni tieto súčasti pred užívateľsky nainštalovanými aplikáciami, ktoré sú spustené v hlavnom operačnom systéme. Softvér a kryptografické izolácie v rámci TEE chránia rôzne oddelené dôveryhodné aplikácie od seba navzájom. Technológia TEE umožňuje škálovateľnosť v IoT vložením hardvérovej ochrany do srdca zariadení [64].

2.4.5 ARM TrustZone

Príkladom využitia TEE je ARM TrustZone. Táto architektúra nám umožňuje na jednom procesore spustiť dve separované prostredia TEE a REE. Firmvér v TEE prostredí je vo všeobecnosti kompaktný, prísne kontrolovaný výrobcom a je určený na poskytovanie bezpečnostných služieb aplikáciám v REE. Firmvér v REE pozostáva z jadra operačného systému ovládaného výrobcom a aplikačnej vrstvy, ktorú

si užívateľ obvykle upravuje. Toto riešenie je kompatibilné s viacerými variantami systému Linux. Týmto spôsobom sa architektúra ARM stáva akousi rozdvojenou osobnosťou. Keď je zabezpečený režim (TEE) aktívny, softvér spustený na procesore má iný pohľad na celý systém, ako softvér bežiaci v nezabezpečenom režime (REE). Týmto spôsobom môžu byť funkcie systému, najmä bezpečnostné funkcie, kryptografické poverenia a kľúče skryté pred REE [65].

Kryptografické kľúče uložené v TrustZone sú celkom dobre zabezpečené. Bezpečnostné vlastnosti spomaľujú pokusy o reverzné inžinierstvo, alebo ukradnutie citlivých informácií. Avšak každý systém má trhliny. Rizikom ale nie je koncepcia systému ako takého. Väčšinou stoja za bezpečnostnými trhlinami jednotlivé implementácie operačných systémov (alebo aplikácií), ktoré bežia v TEE [63]. Existujú prípady, keď sa kvôli vývojovej chybe bolo možné dostať do časti pamäte v jadre TEE, a tým pristupovať k celému systému. Niektoré implementácie zas nepodporujú metódu Address space layout randomization (ASLR), ktorá umiestňuje dáta v operačnej pamäti od náhodne zvolenej adresy, a tým znemožňuje niektoré typy útokov a exploitov [65].

Ďalšou hrozbou je práve v prostredí IoT fyzický prístup k zariadeniu. Riziko spojené s fyzickým útokom je, že je možné extrahovať informácie z jedného zariadenia, ako sú kľúče alebo zdrojový kód, a vykonať rozsiahly softvérový útok. Fyzické útoky môžeme rozdeliť do dvoch hlavných kategórií, neinvazívne a invazívne [66]. Neinvazívne útoky môžu mať veľa foriem. Najčastejší je ale útok postrannými kanálmi, ktorého cieľom je odhaliť tajné informácie (napríklad kryptografické kľúče). Tento útok zahŕňa sledovanie správania sa čipu, ako napríklad časovanie operácií, spotreba energie, alebo elektromagnetické vyžarovanie. Neinvazívne útoky zvyčajne nie sú drahé na realizáciu, a preto dokážu byť celkom jednoducho realizovateľné. Invazívne útoky sa začínajú odstránením obalu čipu. Po otvorení čipu je možné vykonať perturbačné útoky, ako aj hĺbkovú kontrolu, či útok úpravami (leptaním, vŕtaním, laserovým rezaním, atď.). V minulosti predstavovali invazívne útoky značné investície, pretože trvalo celé dni, či týždne, kým vysokokvalifikovaní pracovníci laboratória dospeli k nejakému výsledku. V súčasnej dobe je možnosť si špeciálne zariadenia ľahko prenajať (tak isto aj vedomosti od špecialistov), a tým pádom sa aj tento útok stáva dostupnejším.

Problém s fyzickým prístupom rieši nový čip ARM Cortex-M35P [67], ktorý je vyvinutý hlavne pre potreby IoT a chráni naše dáta hlavne pred neinvazívnymi útokmi. Čip obsahuje popri fyzickej ochrane aj rozšírené bezpečnostné opatrenia. Funkcii TrustZone napomáha aj takzvaný CryptoCell [68], čo je vstavaná bezpečnostná platforma, ktorá vykonáva širokú škálu kryptografických operácií. CryptoCell je zameraný na optimalizáciu výkonu v oblasti bezpečnosti, a tým výrazne znižuje spotrebu energie celého čipu. Táto platforma umožňuje vyňať väčšinu kryptogra-

fických úloh z CPU, ktoré sa vykonávajú efektivne na optimalizovanej hardvérovej logike. Obsahuje taktiež hardvérové úložisko kľúčov, takzvané sloty, čo sú vlastne vyhradené hardvérové registre. Modul ponúka službe TEE nastavenie hodnôt do slotu a REE môže špecifikovať slot (podľa indexu) pri volaní AES služby. Toto je jediný možný prístup k týmto slotom. REE teda nemôže čítať kľúče a TEE ich nemôže použiť na šifrovanie. Modul definuje 4 hardvérové sloty na kľúče pre použitie služby AES. Do slotov môžu byť uložené kľúče veľkostí 128, 192, 256 bitov samostatne, alebo ako dvojité kľúče 2×128 alebo 2×256 bitov. Kľúče sú v slotoch uložené vo forme obyčajného textu. Ostatné užívateľské kľúče sú uložené v module tiež vo forme obyčajného textu [69].

2.4.6 Ľahké blokové šifry

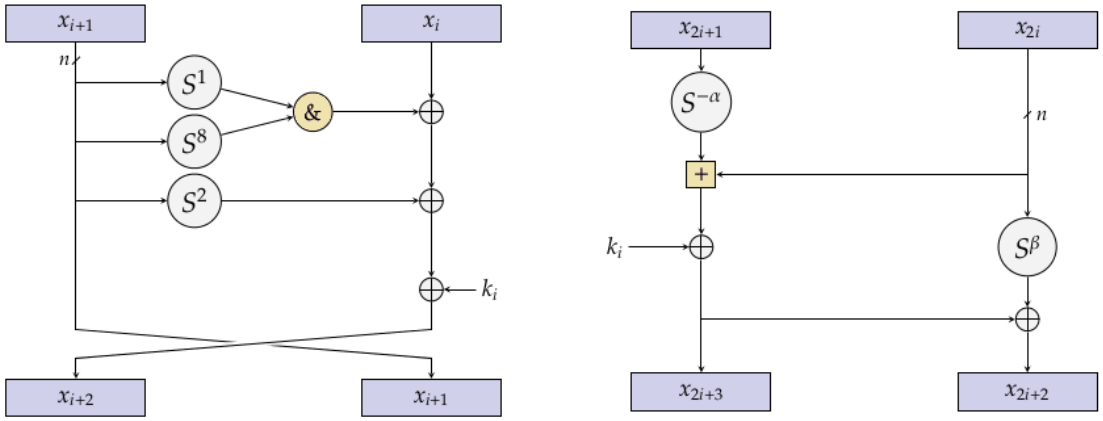
Tieto šifry sú navrhnuté tak, aby boli schopné efektívne pracovať na celej škále zariadení, od nízko-výpočtových až po výkonné stanice. Preto sú vhodné do prostredia IoT. Sú výsledkom kompromisu medzi náročnosťou na výpočtový výkon a bezpečnosťou. Existuje veľmi veľa rôznych ľahkých blokových šifier a ich variant. Niektoré sa špecializujú na softvérové, iné na harvérové implementácie. Vďaka nim dokážeme napríklad aj na malých nevýkonných zariadeniach, ako sú napríklad 8 či 16 bitové mikroprocesory, ako aj 32 bitové procesory architektúry ARM bezpečne a efektívne šifrovať dáta. Vybrané z nich si popíšeme bližšie.

Simon a Speck

Šifry boli navrhnuté Americkou národnou bezpečnostnou agentúrou (NSA). Obe sú riešené Feistelovými sieťami s dvomi vetvami, ale každá má odlišný dizajn Feistelovej funkcie (Obr. 2.1) [70].

Simon je považovaný za hardvérovo orientovanú šifru. Štruktúra predstavuje klasickú Feistelovu sieť, kde Feistelova funkcia pozostáva z aplikovaní základných operácií na vetve, ako sú AND, XOR a slovnej rotácie.

Speck je považovaná za softvérovo orientovanú šifru. Jej konštrukcia je typická pre šifry typu ARX. Operácie pozostávajú zo sčítania, XOR a slovnej rotácií. Dokáže pracovať s veľkosťami blokov 16, 24, 32 48, 64 bitov a kľúčov od 64 až po 256 bitov. Každá verzia má iný počet vykonaných kôl, napríklad s blokom o veľkosti 128 bitov (2x64) a s kľúčom 128 bitov (2x64) šifra vykoná 32 kôl.



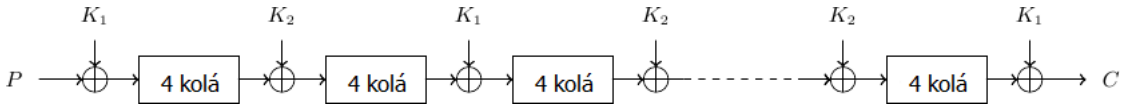
Obr. 2.1: Jedno kolo šifry Simon (v ľavo) a Speck (v pravo) [70].

LEA

Je 128 bitová bloková šifra operujúca na 4 vetvách po 32 bitoch. Je navrhnutá pre vysokorýchlostné softvérové šifrovanie na 32 bitových platformách, pričom sa dá tiež elektívne implementovať aj priamo na hardvér. Využíva operácie 32 bitového modulárneho sčítania, XOR, a slovnej rotácii. To znamená, že jej konštrukcia je typu ARX. Veľkosť kľúčov môže byť 128, 192 a 256 bitov a počet kôl 24, 28 a 32 [71].

LED

Založená na základoch šifry AES. Je to 64 bitová bloková šifra s možnosťou 64 a 128 bitového kľúča. Menší z kľúčov je používaný operáciou XOR napriamo s vnútorným stavom. Väčší je rozdelený na dva 64 bitové podkľúče, ktoré sú použité striedavo (Obr. 2.2). Šifrovanie pozostáva z krokov, kde po každom z nich sa vykoná operácia XOR spoločne s kľúčom. Samostatný krok obsahuje 4 kolá. Každé z nich zas 4 funkcie (AddConstants, SubCells, ShiftRows, MixColumnsSerial). Tieto funkcie sa aplikujú na vnútorný stav, ktorý si môžeme predstaviť ako maticu s veľkosťou 4x4. Šifra s 64 bitovým kľúčom vykonáva 32 kôl, a s 128 bitovým kľúčom 48 kôl [72].



Obr. 2.2: Šifra Led, pri veľkosti kľúča 128 bitov (dva kľúče po 64).

3 Porovnanie výkonu blokových šifier

Vybrané blokové šifry sme podrobili meraniam, pričom sme sa zamerali buď na rýchlosť samotnej šifry, alebo na potrebné systémové požiadavky na výpočtový výkon. Namerané hodnoty sme porovnávali s dostupnými výsledkami meraní (z rôznych literatúr).

3.1 Porovnanie blokových šifier vhodných pre implementáciu na PC

V tejto časti porovnávame najpoužívanejšie šifry z pohľadu výpočtového výkonu na platforme PC. Väčšinu porovnaní sme získali z rôznych dokumentácií, článkov či odborných prác. Všetky tieto výsledky sme sa snažili overiť na našich zariadeniach (PC) pomocou nástroja OpenSSL [73], ktorý dokáže zmerať rýchlosť šifier na danom zariadení. Náš test prebiehal na dvoch zariadeniach s odlišným procesorom (Intel) a operačným systémom (Ubuntu, Windows). Výsledky merania sú priemerné hodnoty, ktoré sme namerali pri veľkosti dát 16, 64, 256, 1024 a 8192 bajtov.

3.1.1 3DES a AES

Pri malých súboroch v radoch desiatkach kilobajtov je 3DES rýchlejší, alebo rovnako rýchly ako AES, ale ako sa počet dát zväčšuje, AES je výrazne rýchlejší. V prípade pamäťovej náročnosti je na tom lepšie AES [74]. V teste na našich zariadeniach (PC) sme potvrdili výsledok, že AES je rýchlejší ako 3DES. Merania sme uskutočňovali pri móde cbc v oboch prípadoch.

Použité príkazy v OpenSSL:
 # openssl speed -evp aes-128-cbc -rýchlosť šifrovania AES 128, mód cbc,
 # openssl speed -evp des-ede3-cbc -rýchlosť šifrovania 3DES, mód cbc.

3.1.2 AES (128, 256 CBC a XTS)

Podľa dostupných testov šifra AES v móde CBC je rýchlejšia pri použití 128 bitového kľúča v porovnaní s 256 bitovým kľúčom [75], [76], [77] pri šifrovaní aj dešifrovaní. V našich meraniach sme potvrdili tento fakt s menšími odchýlkami, ktoré sme uviedli do príslušnej tabuľky 3.1. Tá zobrazuje percentuálne navýšenie rýchlosti pri použití 128 bitového kľúča voči 256 bitovému.

Tab. 3.1: Porovnanie rýchlostí módu CBC.

CBC	**Dostupné testy**	**Naše meranie**
Veľkosť kľúča	128 bitov	128 bitov
Šifrovanie	+ 40 %	+ 40 %
Dešifrovanie	+ 20 %	+ 30 %

Použité príkazy v OpenSSL:
 # openssl speed -evp aes-128-cbc - rýchlosť šifrovania AES 128, mód cbc,
 # openssl speed -evp aes-256-cbc - rýchlosť šifrovania AES 256, mód cbc,
 # openssl speed -decrypt -evp aes-128-cbc - rýchlosť dešifrovania AES 128, mód cbc,
 # openssl speed -decrypt -evp aes-256-cbc - rýchlosť dešifrovania AES 256, mód cbc.

V móde XTS je šifra AES podľa dostupných testov rýchlejšia pri použití 128 bitového kľúča v porovnaní s 256 bitovým [75] pri šifrovaní aj dešifrovaní. Pri našich meraniach sme namerali podobné hodnoty a potvrdili tak túto teóriu. V príslušnej tabuľke 3.2 zobrazujeme percentuálne navýšenie rýchlosti pri použití 128 bitového kľúča voči 256 bitovému.

Tab. 3.2: Porovnanie rýchlostí módu XTS.

XTS	**Dostupné testy**	**Naše meranie**
Veľkosť kľúča	128 bitov	128 bitov
Šifrovanie	+ 30 %	+ 30 %
Dešifrovanie	+ 30 %	+ 35 %

Použité príkazy v OpenSSL:
 # openssl speed -evp aes-128-xts - rýchlosť šifrovania AES 128, mód xts,
 # openssl speed -evp aes-256-xts - rýchlosť šifrovania AES 256, mód xts,
 # openssl speed -decrypt -evp aes-128-xts - rýchlosť dešifrovania AES 128, mód xts,
 # openssl speed -decrypt -evp aes-256-xts - rýchlosť dešifrovania AES 256, mód xts.

Keď sme v našom teste porovnali módy CBC a XTS pri veľkosti kľúča 256 bitov, potvrdili sme teóriu, ktorá hovorí, že mód XTS je výhodné použiť pre šifrovanie celých diskov, alebo jeho častí. Čo sa týka rýchlosti, pri menších súboroch (16 bajtov a menej) dosahuje lepšie výsledky mód CBC, ale čím sú súbory väčšie, tak XTS dokáže byť mnohonásobne rýchlejší (pri veľkosti súboru 8192 bajtov bol rozdiel už päť-násobný).

Použité príkazy v OpenSSL:
 # openssl speed -evp aes-256-cbc
 # openssl speed -evp aes-256-xts
 # openssl speed -decrypt -evp aes-256-cbc
 # openssl speed -decrypt -evp aes-256-xts

3.1.3 Dodatočné merania s novými inštrukciami AES

Ďalšie testy, ktoré sme vykonali, majú dokázať, aké veľké je zvýšenie výkonu šifry AES vďaka využitiu inštrukcií AES-NI. V našom prípade boli namerané hodnoty v móde CBC pri 256 bitovom kľúči pri šifrovaní dvojnásobné a dešifrovaní 10-násobné v porovnaní s čisto softvérovým AES. Ďalej sme testovali aj mód XTS s veľkosťou kľúča 256 bitov. Pri šifrovaní nám vyšlo 6-násobné a pri dešifrovaní 7-násobné navýšenie výkonu. Mód XTS sme zmerali aj v programe VeraCrypt, ktorý dokáže šifrovať a aj merať AES XTS. Výsledkami sa preukázalo, že s použitím AES-NI bolo navýšenie výkonu 4-násobné. Výsledky sú zobrazené v nasledujúcej tabuľke 3.3, ktorá zobrazuje rýchlostné navýšenie v násobkoch pri použití AES-NI voči klasickej implementácii pri použití jednotlivých módov CBC a XTS.

Tab. 3.3: Porovnanie rýchlostí AES a AES-NI pri módoch CBC a XTS.

		Dostupné testy	**Naše meranie**
		AES-NI	AES-NI
CBC 256	**Šifrovanie**	+ 2 až 5-násobok	+ 2-násobok
	Dešifrovanie	+ 10-násobok	+ 10-násobok
XTS 256	**Šifrovanie**		+ 6-násobok
	Dešifrovanie		+ 7-násobok

Použité príkazy v OpenSSL:
 # openssl speed -evp aes-256-cbc
 # openssl speed -evp aes-256-xts
 # openssl speed -decrypt -evp aes-256-cbc
 # openssl speed -decrypt -evp aes-256-xts
 # OPENSSL_ia32cap="~0x200000200000000" openssl speed -evp aes-256-cbc
 # OPENSSL_ia32cap="~0x200000200000000" openssl speed -evp aes-256-xts
 # OPENSSL_ia32cap="~0x200000200000000" openssl speed -decrypt -evp aes-256-cbc
 # OPENSSL_ia32cap="~0x200000200000000" openssl speed -decrypt -evp aes-256-xts

Pridaním OPENSSL_ia32cap=”∼0x200000200000000” pred klasický príkaz sme nastavili nástroj OpenSSL tak, aby nevyužíval AES-NI.

3.1.4 Zhrnutie meraní

Tieto merania nemali ukázať reálne rýchlosti, pretože tie sa líšia v závislosti na výkone hardvéru či softvérovej časti, ktorá nám umožnila testy vykonať. V tejto časti sme overili teóriu, ktorá šifra je rýchlejšia a za akých podmienok. Zistili sme, že rozdiel v rýchlostiach šifry AES pri 128 a 256 bitovom kľúči, nie je dvojnásobný, ako by sa niekomu mohlo zdať, ale je to podstatne menej. Tým pádom môžeme citlivé údaje zašifrovať väčším kľúčom, kde sa nám rapídne zvýši bezpečnosť, pri nie až tak veľkom zvýšení potreby výpočtového výkonu. Tak isto sme potvrdili, že mód XTS je vhodný na šifrovanie diskov, pretože pri väčších súboroch vykazoval niekoľkonásobne väčšiu rýchlosť oproti módu CBC. Všetky testy boli uskutočnené na procesoroch podporujúcich takzvanú hardvérovú akceleráciu šifry AES. Pri Intel procesoroch ide o AES-NI inštrukcie. Po preštudovaní výsledkov porovnaní AES a AES-NI sme zistili, že pri móde CBC a veľkosti kľúča 256 bitov sa zvýšil výkon 5-krát pri šifrovaní a 10-krát pri dešifrovaní. Úspora na spotrebe elektrickej energie sa priblížila k 40 % [78]. V ďalšej dokumentácií sme zistili, že pri šifrovaní v móde CBC pri 256 bitovom kľúči bolo namerané dvojnásbné zvýšenie výkonu [79], [73].

3.2 Porovnanie blokových šifier vhodných pre implementáciu na zariadeniach využívaných v IoT

Porovnanie bolo vykonané nástrojom FELICS [80]. Výsledky boli namerané a následne zosumarizované do tabuliek z 3 odlišných zariadení: 8 bitový Atmel AVR ATmega128 (Tab. 3.5), 16 bitový Texas Instruments MSP430F1611 (Tab. 3.6), 32 bitové Arduino Due ARM Cortex-M3 (Tab. 3.7), ktorých parametre sú ďalej popísané v tabuľke 3.4 [81].

Tab. 3.4: Charakteristika použitých mikrokontrolérov.

	AVR	**MSP**	**ARM**
CPU	8-bit	16-bit	32-bit
Frekvencia (MHz)	16	8	84
Architektúra	Harvard	Von Neumann	Harvard
Flash pamäť (KB)	128	48	512

Meranie bolo vykonané ako šifrovanie 128 bitov dát s využitím módu CTR [81]. Pri každej architekture bola použitá najoptimálnejšia implementácia danej testovanej

verzie šifry. Veľkosti blokov a kľúčov sú udávané v bitoch. Veľkosť kódu a využitie pamäte RAM je vyjadrená v bajtoch. Čas potrebný na vykonanie šifrovania (kódu) je uvádzaný v cykloch (procesora). Do porovnania sme pridali šifru AES, aby sa ukázalo, ako obstojí v porovnaní so šiframi stavanými na tieto zariadenia. Výsledky sú podrobne zapísané v tabuľkách a najlepšie z nich aj zvýraznené.

Tab. 3.5: Porovnanie ľahkých blokových šifier na architektúre AVR.

			AVR		
Šifry	Blok [b]	Kľúč [b]	Kód [B]	RAM [B]	Čas [cykly]
Speck	64	128	**504**	**58**	2995
Speck	64	96	558	**58**	**2725**
Simon	64	96	652	62	4347
Simon	64	128	660	62	4523
LEA	128	128	894	78	4029
AES	128	128	1234	79	3414
LED	64	80	2584	256	125987

Z porovnania sa nám potvrdilo (Tab. 3.5), že šifra Speck je optimalizovaná pre softvérové riešenia a taktiež je najviac vhodná na použitie na zariadeniach s architektúrou AVR z vybraných šifier. Úspora kódu oproti AES je viac ako polovičná, a v potrebe pamäti RAM štvrtinová. Aj cez túto skutočnosť šifra v meraní obstála lepšie, ako špecializovaná LED, ktorá potrebovala na operáciu šifrovania štvornásobný čas oproti sponínanému AES (Tab. 3.5). Z toho vyplýva, že nie vždy špecializovaná šifra pre IoT má lepšie parametre v porovnaní s optimalizovanou klasickou šifrou.

Tab. 3.6: Porovnanie ľahkých blokových šifier na architektúre MSP.

			MSP		
Šifry	Blok [b]	Kľúč [b]	Kód [B]	RAM [B]	Čas [cykly]
Speck	64	128	**332**	**48**	2256
Speck	64	96	372	**48**	**2068**
Simon	64	96	460	56	3124
Simon	64	128	468	56	3238
LEA	128	128	722	78	3344
AES	128	128	1170	80	4617
LED	64	80	4420	104	148334

Aj v tomto porovnaní (Tab. 3.6) nám vyšla šifra Speck ako najlepšia z porovnávaných. Šifra LED sa nám zas umiestnila na poslednom mieste, kde identicky túto

architektúru znovu nezvládla a čas potrebný na šifrovanie je opäť enormný oproti ostatným oponentom. Tak isto veľkosť kódu je štvornásobná oproti AES.

Tab. 3.7: Porovnanie ľahkých blokových šifier na architektúre ARM.

			ARM		
Šifry	Blok [b]	Kľúč [b]	Kód [B]	RAM [B]	Čas [cykly]
Speck	64	128	**278**	**60**	965
Speck	64	96	284	72	**964**
Simon	64	96	414	68	1372
Simon	64	128	420	80	1422
LEA	128	128	536	120	1146
AES	128	128	1352	124	3969
LED	64	80	2220	336	36128

Pri ARM architektúre najlepšou šifrou z porovnávaných je Speck, kde ale vidíme (Tab. 3.7), že pri použití menšieho kľúča (96 bitov) je čas potrebný na šifrovanie takmer identický s 128 bitovým kľúčom. Tak isto, keď ide o pamäť RAM, je táto šifra s väčším kľúčom úspornejšia. V tomto prípade je aj šifra LEA rýchlejšia ako Simon.

V nasedujúcom meraní sme pomocou knižnice **Crypto++ Library** [82] verzie 8.2 zmerali výkon vybraných šifier na našom zariadení PC. Táto knižnica obsahuje framework, ktorý umožňuje meranie algoritmov v danej knižnici. Po stiahnutí knižnice v priečinku existuje predpripravený program na meranie výkonu šifier (cryptest.exe). Cez konzolu CMD sme daný program spustili s atribútmi **b 4.5** (b znamená meranie výkonu všetkých algoritmov v knižnici, 4.5 je takt jadra procesora).

Meranie sa uskutočnilo na procesore Intel Core i5-4670K (takt jadier 4,5 GHz) s Intel inštrukčnou sadou SSSE3 (Supplemental Streaming SIMD Extensions 3). Šifry boli merané pri móde CTR. Výsledky merania sú zapísané v tabuľke 3.8, kde sú jednotlivé merané šifry zoradené podľa veľkosti kľúča od 128 až po 256 bitov. Kolónka MB/s označuje priepustnosť (rýchlosť) šifier a Cykly na bajt zas náročnosť na výpočtový výkon. Zvýraznené sú vždy najlepšie hodnoty z dého typu šifry.

Tab. 3.8: Porovnanie ľahkých blokových šifier na PC.

Šifry	**Blok [b]**	**Kľúč [b]**	**MB/s**	**Cykly na bajt**
Speck	128	128	**42**	3,813
Speck	128	192	40	**3,407**
Speck	128	256	39	3,488
Simon	128	128	**10**	**5,261**
Simon	128	192	9	5,442
Simon	128	256	9	7,781
LEA	128	128	**20**	**4,110**
LEA	128	192	17	6,255
LEA	128	256	15	6,975

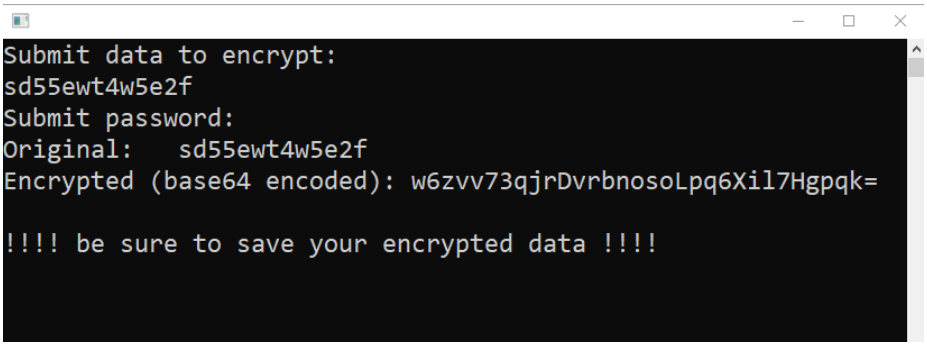
Najrýchlejšou šifrou bola v našom meraní šifra Speck (Tab. 3.8). Jej varianty s väčším kľúčom dosahovali síce nižšie rýchlosti, ale boli úspornejšie pre výpočtový výkon. Z toho vyplýva, že navýšením úrovne bezpečnosti pri tejto šifre na platforme PC neuškodíme razantne jej výkonu. Pri šifre Simon platí táto teória iba po kľúč 192. Väčší 256 bitový už výrazne navyšuje prostriedky na výpočtový výkon. LEA je rýchlejšia ako Simon pri menšom 128 a potom pri väčšom 256 bitovom kľúči.

4 Návrh a implementácia bezpečného uloženia kľúčov na PC

Naše riešenie sme navrhli ako aplikáciu, ktorá je schopná šifrovať reťazec znakov, ktorý môže predstavovať šifrovací, či autentizačný kľúč, alebo iné citlivé údaje. Hlavnou vlastnosťou je bezpečnosť a jednoduchosť pre používateľa. Ako základ sme zvolili symetrickú šifru AES s veľkosťou kľúča 128 bitov v móde CBC. Vybrali sme ju preto, lebo je pokladaná za bezpečnú šifru a hlavne rýchlu, vďaka podpore inštrukcií AES-NI v skoro každom dostupnom procesore v počítačoch. Pri šifrovaní a dešifrovaní pomocou AES je potrebný kľúč a inicializačný vektor. Tieto údaje je potrebné niekde uchovať pre následnú dešifráciu dát. Keďže sme nechceli, aby ich program ukladal niekde externe (napríklad do textového súboru), uchovávame ich priamo v binárnom kóde programu. Pri šifrovaní je potrebné zadať užívateľské heslo, ktoré sa následne vyžaduje pri dešifrovaní. Toto heslo je špecifické tým, že program neberie v úvahu pri overovaní len znaky, ktoré sa zadávajú, ale aj časové medzery medzi nimi. Celá aplikácia je určená na jedno použitie, to znamená, že keď zašifrujeme jeden reťazec znakov, následne danú aplikáciu môžeme použiť už iba na dešifrovanie. Pri zabudnutí hesla alebo strate šifrovaného textu je možnosť vymazania hesla aj údajov z binárneho kódu.

4.1 Popis implementácie aplikácie

Náš návrh sme implementovali do konzolovej aplikácie (Obr. 4.1) použitím jazyka C# (.NET Framework 4.7.2), ktorá je spustiteľná na operačnom systéme Windows. Pre AES sme použili menný priestor System.Security.Cryptography a pre ukladanie do binárneho kódu knižnicu Vestris.ResourceLib.



Obr. 4.1: Konzolová aplikácia po vykonaní operácie šifrovanie.

System.Security.Cryptography

Poskytuje kryptografické služby vrátane bezpečného kódovania a dekódovania údajov, ako je napríklad hashovanie, generovanie náhodných čísel a overenie pravosti správ [83]. V našej aplikácii sme použili triedu RijndaelManaged, ktorá zabezpečuje šifrovanie aj dešifrovanie, či generovanie kľúčov. Oproti triede AesManaged máme voľnosť výberu veľkosti blokov 128, 160, 192, 224, 256 bitov. My sme vybrali 128 bitov, čiže ide o klasické AES.

Vestris.ResourceLib

Táto knižnica zabezpečuje zápis a čítanie reťazcu znakov do a z kompilovaného binárneho kódu. Dokáže prepísať údaje vo vlastnostiach programu, ako sú verzia súboru, informácie o spoločnosti, autorské práva a mnoho ďalších. Vždy záleží od typu súboru. V našom prípade pri súbore s príponou .exe ide o Popis súboru, Produktové meno, Ochrannú známku [84].

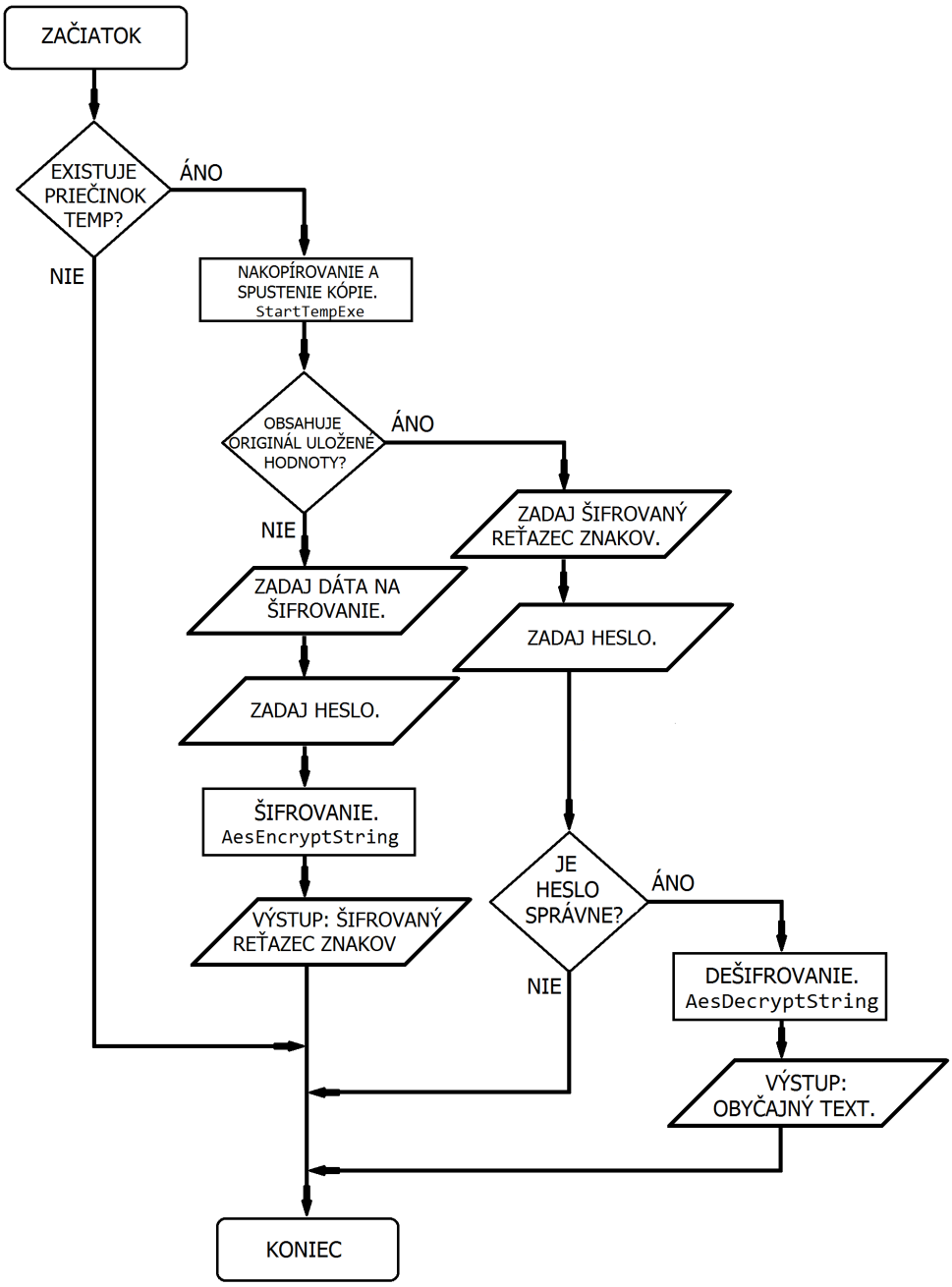
4.2 Samotná funkcionalita programu

Náš program sa po zapnutí automaticky sám nakopíruje do priečinku Temp, odkiaľ sa táto kópia aj spustí (vývojový diagram Obr. 4.2). Originál sa ukončí. Funkcionalitu zabezpečuje funkcia StartTempExe. Tento krok je potrebný pre prípravu na uloženie hesla, kľúča a vektora potrebného pri šifrovaní do binárneho kódu, pretože spustený program (používaný ako aktívny v procesoch) nedokáže zapisovať sám do seba.

Nasleduje kontrola zapísaných dát v binárnom kóde. Ak sa už predtým uskutočnilo šifrovanie, budú tam uložené potrebné kľúče. Ak nie, premenné ostanú prázdne. Podľa toho sa program rozhodne, či ponúkne možnosť šifrovať, alebo dešifrovať dát. Čítanie a zápis do a z binárneho kódu sa uskutočňuje funkciami ResourcesRead respektíve ResourcesWrite (Výpis 4.1).

Všetko prebieha za pomoci knižnice Vestris.ResourceLib. Šifrovací kľúč sa ukladá ako informácia o súbore pod premennú FileDescription, inicializačný vektor pod ProductName a užívateľské heslo pod LegalTrademarks. Všetky hodnoty sú enkódované do formátu Base64 a zapisujú sa alebo čítajú z originálneho neskopírovaného súboru.

Pri šifrovaní zadávame náš reťazec znakov a ďalej užívateľské heslo. Keď toto heslo píšeme do konzoly, funkcia SubmitPassword ukladá znaky, ktoré zadávame a zároveň aj čas v milisekundách medzi jednotlivými stlačeniami kláves. Na meranie času používame funkciu stopWatch z menného priestoru System.Diagnostics, čo sú klasické stopky.



Obr. 4.2: Vývojový diagram konzolovej aplikácie.

Nasleduje samotné šifrovanie pomocou AES. To prebieha vo funkcií AesEncryptString (Obr. 4.2). Tu sa najskôr vygeneruje kľúč a inicializačný vektor. Ako výplň (padding) používame ISO10126, aby sme šifrovanú správu mali v dĺžke potrebného 128 bitového bloku. Mód šifry je CBC. Pomocou kľúča a vektora šifrujeme zadaný reťazec znakov, výstupom je pole naplnené 8-bitovými celými číslami (pole byte). Tie sa pri vypisovaní do konzoly prekonvertujú na reťazec znakov a enkódujú do tvaru Base64. Taktiež sa nám výsledný reťazec automaticky vloží do clipboardu pre

Výpis 4.1: Čítanie a zápis do a z binárneho kódu.

```
static void ResourcesRead(ref string key, ref string vector,
   ref string pw){

            VersionResource versionResource = new
               VersionResource();
            versionResource.LoadFrom(originalExePath);

            StringFileInfo stringFileInfo = (StringFileInfo)
               versionResource["StringFileInfo"];

            key = stringFileInfo["FileDescription"];
            vector = stringFileInfo["ProductName"];
            pw = stringFileInfo["LegalTrademarks"];

            key = Base64.Decode(key);
            vector = Base64.Decode(vector);
            pw = Base64.Decode(pw);

        }
        static void ResourcesWrite(string key, string vector,
            ref string pw){
            VersionResource versionResource = new
               VersionResource();
            versionResource.LoadFrom(originalExePath);

            StringFileInfo stringFileInfo = (StringFileInfo)
               versionResource["StringFileInfo"];

            key = Base64.Encode(key);
            vector = Base64.Encode(vector);
            pw = Base64.Encode(pw);

            stringFileInfo["FileDescription"] = key;
            stringFileInfo["ProductName"] = vector;
            stringFileInfo["LegalTrademarks"] = pw;
            versionResource.SaveTo(originalExePath);
        }
```

ľahšie uloženie mimo konzolu. Po šifrovaní sa nám heslo, kľúč a vektor uložia do binárneho kódu originálu. Tým pádom je program pripravený na dešifrovanie.

Po druhom spustení nám program automaticky poskytne možnosť dešifrovania dát, pretože našiel vopred vygenerované heslo, kľúč a vektor pri šifrovaní v binárnom kóde. Ďalej musíme zadať heslo, aby sa vykonalo dešifrovanie. Vo funkcii PasswordVerification sa porovnáva uložená fráza z aktuálne zadanou, a keď sa tie zhodujú, potom sa porovnávajú jednotlivé časy medzi stlačenými klávesami. Funkcia LoadTimeToArray najskôr načíta časové záznamy z obidvoch hesiel do polí a následne sa porovnajú prihliadajúc na možnú odchýlku 250 milisekúnd (premenná timeCheckingRange) medzi dvoma stlačenými klávesami. Tento proces zabezpečuje funkcia CheckTimeInRange (Výpis 4.2).

Výpis 4.2: Overenie veľkosti odchýlky času.

```
static bool CheckTimeInRange(string[] pw_time_array, string[]
    pw_time_comparison_array){
            bool passIsGood = false;

            for (int i = 1; i < pw_time_array.Length - 1; i
               ++)
            {
                if (Math.Abs(Int32.Parse(pw_time_array[i]) -
                   Int32.Parse(pw_time_comparison_array[i]))
                   <= timeCheckingRange)
                {
                    passIsGood = true;
                }
                else
                {
                    passIsGood = false;
                    break;
                }
            }
            return passIsGood;
        }
```

Dešifrovanie prebieha vo funkcií AesDecryptString (Obr. 4.2), kde sa nám kľúč a vektor prekonvertujú z reťazca znakov do potrebného formátu pre pole byte. Nastavenie šifry je rovnaké, ako pri šifrovaní. Výstupom je originálny reťazec znakov.

Pre prípad ďalšieho použitia programu, alebo zabudnutia hesla je tu možnosť vymazať nami pridané dáta z binárneho kódu. Vždy na konci akcie (šifrovanie, dešifrovanie) nám program ponúkne vymazať heslo, vektor, kľúč, keď stlačíme dvakrát tú istú klávesu. Vymazanie prebieha prepísaním funkciou ResourcesWrite premenné prázdnymi reťazcami znakov.

Po vykonaní všetkých úkonov, čo sme potrebovali sa program môže ukončiť. Teda jeho kópia v priečinku Temp, ktorá je nám už zbytočná. Preto sa tesne pred ukončením programu vykoná funkcia DeleteThisExe (Výpis 4.3), ktorá má za úlohu spustiť nový proces s novou konzolou na pozadí, takže je pre používateľa neviditeľná, počkať tri sekundy a vymazať daný súbor. Čakaním docielime to, že sa zatiaľ hlavný program ukončí a tým pádom sa môže s ním ľubovoľne narábať, teda aj vymazať.

Výpis 4.3: Samovymazanie kópie programu.

```
static void DeleteThisExe()
        {
            Process.Start(new ProcessStartInfo()
            {
                Arguments = "/C choice /C Y /N /D Y /T 3 &
                   Del \"" + Application.ExecutablePath +
                   "\"",
                WindowStyle = ProcessWindowStyle.Hidden,
                CreateNoWindow = true,
                FileName = "cmd.exe"
            });
        }
```

4.3 Náročnosť na výpočtový výkon

To, koľko program potrebuje výpočtového výkonu, sme namerali prostredníctvom Performance profileru vo Visual studiu 2019. Meranie prebehlo na počítači s procesorom Intel Core i5-4670K (takt jadier 4,5 GHz). Vyťaženie procesora sa meralo rýchlosťou 1000 vzorkov za sekundu. Meranie počas šifrovania prebiehalo pri práci s reťazcami znakov do veľkosti jedného bloku (128 bitov). Výsledky sme zapísali do tabuľky 4.1.

Spustenie programu zaberie 126 milisekúnd času procesora a jeho klonu dodatočných 126. Beh programu pri šifrovaní 80 ms, z toho načítanie kľúčov z binárky 1 ms,

vygenerovanie AES kľúča taktiež 1 ms, zapísanie kľúčov do binárky 5 ms. Najväčšie zaťaženie procesora v jednom okamihu bolo pri šifrovaní a to 4 %. V pamäti zaberá miesto o veľkosti 8,5 MB, z čoho hosťujúca konzola 6,5 MB a samotný program 2 MB.

Tab. 4.1: Porovnanie s programom AESCrypt.

	Náš program	**AESCrypt**
Pamäťová náročnosť	8,5 MB	2 MB
Časové vyťaženie procesora (beh programu pri šifrovaní)	80 ms	60 ms
Maximálne vyťaženie v jedenom momente	4 %	6 %

4.4 Bezpečnosť našej implementácie na PC

Program je zabezpečený heslom, kde sa pri overovaní neberú do úvahy len znaky čo užívateľ zadáva, ale aj časové medzery medzi jednotlivými stlačeniami kláves. Týmto krokom dokážeme eliminovať prípadné ukradnutie zadávaného hesla škodlivým softvérom, takzvaným keyloggerom hlavne preto, že vývojári týchto softvérov nerátajú s takouto možnosťou overenia, pretože je zriedkavá, respektíve nemáme informáciu o podobnej implementácií. Taktiež dokážeme zamedziť formy sociálneho inžinierstva, kedy sa útočník môže pozerať na klávesnicu užívateľa a nevediac o tejto funkcii si zapamätá len znaky. Ďalším zabezpečením je, že uchovávanie hesla, kľúča, ako aj inicializačného vektora potrebného k dešifrovaniu, sme vyriešili uložením do vlastného binárneho kódu. To znamená, že si ich program berie zo sebou hocikde ho uložíme. Tým pádom je možné po šifrovaní uložiť daný program na externé médium, iné zariadenie, či cloud. Túto možnosť ocenia hlavne tí užívatelia, ktorí nevedia zaručiť, aby s programom na danom zariadení nemanipulovala iná osoba. Pomocou programu ConfuserEx [85] sme dosiahli, že naša aplikácia obsahuje v tele knižnicu (Vestris.ResourceLib), ktorá bola dynamicky linkovaná (samostatný súbor dll). Pri tejto akcii sa vykonalo aj základné zneprehľadnenie kódu (obfuskácia). V prípade potreby, by sa mohla implementovať možnosť šifrovania uložených dát v binárnom kóde užívateľským heslom pre zvýšenie bezpečnosti.

Bezpečnosť samotného šifrovania

Pre samotné šifrovanie sme použili symetrickú blokovú šifru AES. Tá je považovaná za bezpečnú. Sú známe útoky, ako napríklad rozlišovací útok známeho kľúča (Known-key distinguishing attack), ktorý ale popisuje scenár, keď útočník pozná

šifrovací kľúč a taktiež bol vyskúšaný iba na verzií AES pri 7 kolách [86], kdežto naša implementácia ich robí 10.

Útoky typu hrubou silou sú pri použití 128 bitového kľúča veľmi náročné na výpočtový výkon a tým pádom skoro nemožné. Jedinou možnou zraniteľnosťou je útok bočným kanálom (side-channel attack), keď systém uvoľňuje informácie. V našom prípade ide o proces ukladania kľúča, hesla a vektoru z kópie programu do originálu. Pri tejto akcii by mohol útočník teoreticky odchytiť tieto údaje (napr. pri sledovaní pamäte) a použiť ich k dešifrovaniu dát. Kľúč a inicializačný vektor sa pri každom šifrovaní náhodne generujú. K tejto operácii sa využíva funkcia menného pristoru System.Security.Cryptography, ktorá generuje bezpečné náhodné čísla. Tieto čísla sú nazývané bezpečnými, pretože klasický pseudo náhodný generátor je založený na systémových hodinách počítača a tým pádom vytvorenie náhodných čísel v rovnakom čase môže viesť k rovnakému výstupu, čiže nie je bezpečný na vytvorenie šifrovacích kľúčov.

5 Návrh a implementácia bezpečného uloženia kľúčov na IoT zariadení

Naše riešenie sme navrhli ako aplikáciu v prostredí IoT, ktorá je schopná šifrovať reťazec znakov, ktorý môže predstavovať šifrovací, či autentizačný kľúč, alebo iné citlivé údaje. Zamerali sme sa na zariadenia Triedy 2 a 3, kde hosťujúci operačný systém predstavuje Windows 10 IoT Core. Ako základ sme zvolili ľahkú symetrickú blokovú šifru Speck, špecializovanú pre zariadenia IoT. Prístup do samotnej aplikácie je realizovaný prostredníctvom prihlásenia sa do účtu Microsoft.

5.1 Popis implementácie aplikácie

Náš návrh sme implementovali ako UWP (Universal Windows Platform) aplikáciu v jazyku C#. Tým pádom ju môžeme spustiť na klasickom operačnom systéme Windows 10 (verzia 1809, build 17763) a zároveň aj na Windows 10 IoT Core. Pre šifru Speck sme použili knižnicu SimonSpeckNet.Speck spoločne s menným priestorom System.Security.Cryptography, a na prístup do aplikácie menný priestor Microsoft.Identity.Client a službu Microsoft Graph.

SimonSpeckNet.Speck

Táto knižnica je C# nadstavbou C++ implementácie ľahkej blokovej šifry Simon a Speck [87]. Podporuje veľkosť blokov 128 bitov a veľkosť kľúčov 128, 192 a 256 bitov, pri módoch ECB a CTR.

Universal Windows Platform

Táto platforma umožňuje vytvárať aplikácie, ktoré sú schopné bežať na všetkých zariadeniach s operačným systémom Windows 10 a jeho rôznymi verziami (PC, mobily, IoT zariadenia, Xbox, a iné). Podporuje viaceré programovacie jazyky C#, C++, Visual Basic a ďalšie. Pre používateľské prostredie je možné použiť XAML, HTML či DirectX. Prvky používateľského rozhrania dokážu reagovať na zmenu rozlíšenia, či veľkosti obrazovky, úpravou rozloženia a mierky zobrazujúcich častí, komponentov. Celá aplikácia dobre spolupracuje s viacerými typmi vstupov, ako sú klávesnica, myš, dotyk, pero a iné. Čo sa týka bezpečnosti, je možné presne definovať aké rozhrania a dáta môže naša aplikácia využívať (napríklad prístup k mikrofónu, webkamere, informácie o polohe a iné) [88].

Microsoft.Identity.Client

Obsahuje viaceré knižnice pre .NET na Microsoft autentizáciu (hlavne MSAL.NET), ktorá dokáže jednoducho obstarávať tokeny pre platformu identity Azure AD v2.0 od prihlásených užívateľov rôznych typov Microsoft účtov. Tieto tokeny sprístupňujú Microsoft Cloud API a všetky ďalšie API zabezpečené platformou identity spoločnosti Microsoft [89].

Microsoft Azure portál

Vždy je potrebné svoju aplikáciu najskôr zaregistrovať na portáli Microsoft Azure v sekcii App registrations, kde po registrácii obdržíme Klient ID (Application (client) ID), ktoré je potrebné zadať do kódu pre správny chod MSAL. Toto ID je unikátne a identifikuje našu aplikáciu. Na tomto portáli už v našej registrovanej aplikácii v sekcií API permissions môžeme pridať aké API bude povolené a čo bude môcť realizovať (resourceAccess v manifeste). V sekcii manifest zase vidíme celkové nastavenie našej aplikácie, ktoré môžeme ľubovoľne meniť podľa potreby (JSON reprezentácia, Obr. 5.1). Za zmienku stojí možnosť singInAudience, kde po nastavení hodnoty AzureADMyOrg dostanú prístup do aplikácie iba používatelia zo zoznamu vlastnej organizácie, ktorý sa nachádza v Azure Active Directory v podsekcii All users.

Obr. 5.1: Ukážka časti manifestu (JSON reprezentácia).

Microsoft Graph

Toto API vyžaduje token, ktorý umožňuje prístup k špecifickým zdrojom. V našom prípade je to možnosť prihlásenia a čítania profilu užívateľa (User.read). Token sa vyžiada pomocou nástroja MSAL a pridá sa do hlavičky HTTP autentizácie pri každom volaní chráneného špecifického zdroja (Obr. 5.2) [90].

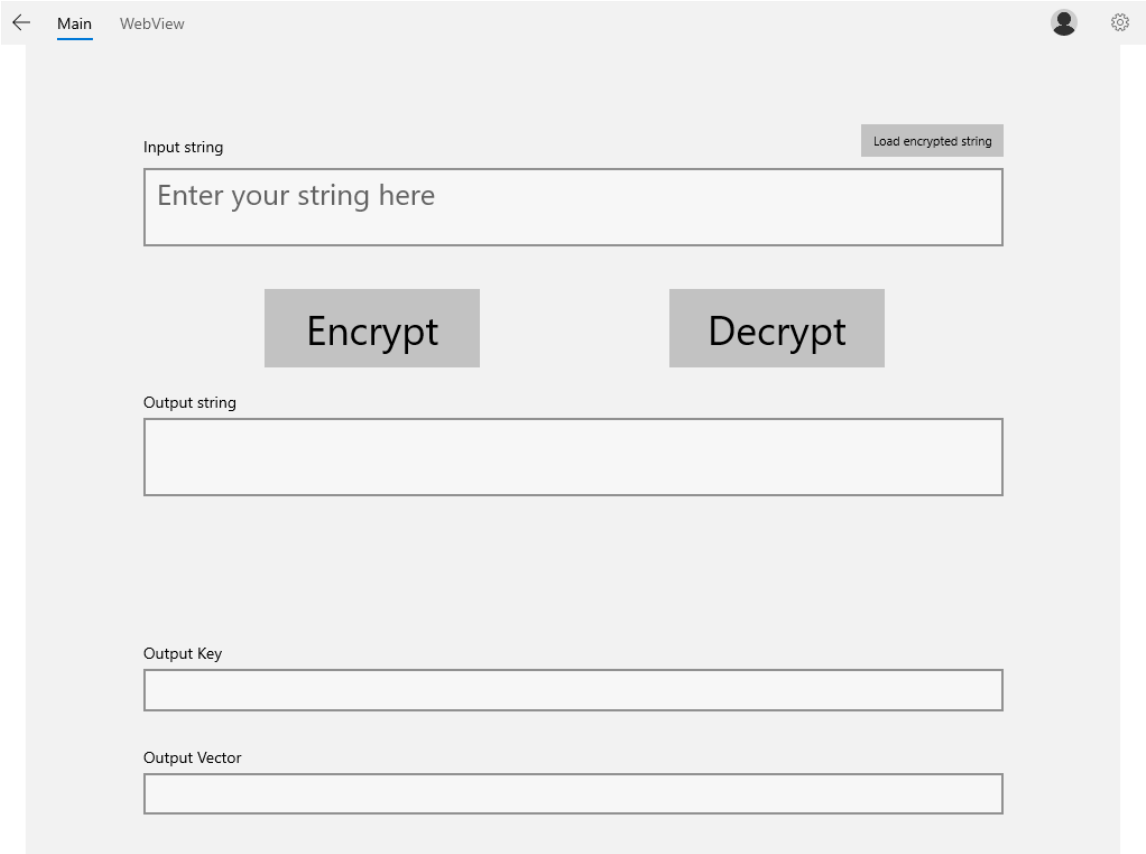
Obr. 5.2: Žiadosť o token a prístup ku GRAPH API [90].

5.2 Samotná funkčnosť aplikácie

Po spustení našej aplikácie sa nám najskôr ponúkne možnosť prihlásenia (LogInPage.xaml.cs). Po stlačení tlačidla (Click to log in) sa nám zobrazí klasický prihlasovací formulár, ktorý poznáme z každej Microsoft webovej stránky, či aplikácie. Po prihlásení pomocou platformy identity Azure AD v2.0 obdrží Microsoft Graph požadovaný token, cez ktorý vyžiada údaje o užívateľovi a uloží ich do aplikácie, aby bolo viditeľné, kto je prihlásený (IdentityService.cs, MicrosoftGraphService.cs). Token sa ďalej uchová, a pokiaľ sa neodhlásime a vypneme aplikáciu, tak pri ďalšom jej spustení budeme automaticky prihlásení.

Po prihlásení sa nám zobrazí samotná možnosť šifrovania (MainPage.xaml.cs Obr. 5.3). Používateľské rozhranie je veľmi jednoduché a prehľadné. Do prvej kolónky na text (Input string) vložíme náš reťazec znakov, ktorý chceme bezpečne uchovať. Po stlačení tlačidla Encrypt nám náš program zobrazí v kolónke Output string zašifrovaný reťazec znakov a zároveň ho aj uloží do súboru EncryptedString.txt na zariadení. Taktiež nám zobrazí vygenerovaný kľúč (Output Key) a vektor (Output Vector), ktoré si treba pre možnosť dešifrovania uchovať (odporúča sa externé úložisko). Tlačidlo Load encrypted string načíta posledný uložený šifrovaný reťazec znakov zo súboru.

Nastavenie šifry je nasledovné. Veľkosť bloku 128 bitov, veľkosť kľúča 128 bitov a ako výplň (padding) používame nuly (Zeros). Mód šifry je CTR. Samotné šifrovanie prebieha vo funkcii EncryptStringAsync, kde sa najskôr nami zadaný reťazec znakov prekonvertuje na 8 bitový tvar do poľa byte. Potom sa vygeneruje kľúč



Obr. 5.3: Užívateľské rozhranie v aplikácii pre IoT pri šifrovaní.

a vektor s ktorých sa vytvorí encryptor používaný ďalej pri šifrovaní. Výsledný šifrovaný reťazec znakov je enkódovaný do formátu Base64. Ten sa následne zobrazí na obrazovku a uloží do súbora. Kľúč aj vektor sa taktiež zobrazia na obrazovke vo formáte Base64. Dešifrovanie prebieha vo funkcií DecryptString (Výpis 5.1). Najskôr sa dekóduje šifrovaný vstup z tvaru Base64 a prekonvertuje na 8 bitový tvar do poľa byte. To isté sa vykoná s kľúčom a vektorom. Následne sa vytvorí decryptor na dešifrovanie. Výstupom je reťazec znakov zadaný nami na šifrovanie.

5.3 Príprava virtuálneho stroja

Náš virtuálny stroj sme pripravili nasledovne. Najskôr sme si stiahli Windows 10 IoT core predpripravený pre MinnowBoard Turbot/MAX. Stiahol sa nám ISO súbor, ktorý obsahuje inštalátor. Ten keď spustíme, tak sa nám nainštaluje špeciálny FFU obraz do priečinka:

C:\Program Files (x86)\Microsoft IoT\FFU\MinnowBoardMax_x64 .

Výpis 5.1: Dešifrovanie šifrou Speck.

```
void DecryptString(object sender, RoutedEventArgs e)
   {
    String EncryptedText = InputText.Text;
    byte[] plainEnc = Base64.DecodeToBytes(EncryptedText);

    using (SymmetricAlgorithm algo = new SpeckCTR())
      {
        algo.BlockSize = 128;
        algo.KeySize = 128;
        algo.Padding = PaddingMode.Zeros;

        algo.Key = Base64.StringToByte(Base64.Decode(key));
        algo.IV = Base64.StringToByte(Base64.Decode(vector));

        using (ICryptoTransform decryptor = algo.
           CreateDecryptor())
           {
             byte[] plainDec = decryptor.TransformFinalBlock(
                plainEnc, 0, plainEnc.Length);
             OutputText.Text = Base64.ByteToString(plainDec);
           } } }
```

Netreba zabúdať, že je to 64 bitový obraz, preto musí byť aj náš virtuálny stroj založený ako 64 bitový. Ďalej stiahneme nástroj ImgMount Tool (verzia 1.0.15) a uložíme ho na plochu. Spustíme konzolu cmd ako administrátor a napíšeme do nej nasledujúci príkaz:

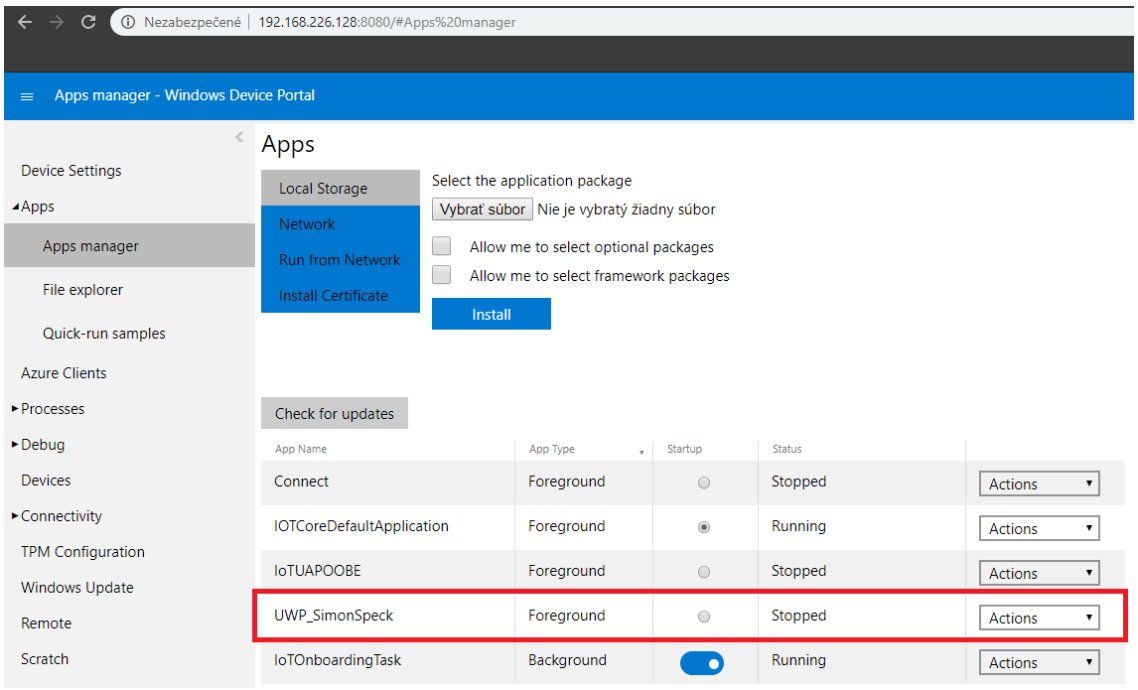
C:\Users\YourUser\Desktop\ImgMount.exe 'C:\Program Files (x86)\Microsoft IoT\FFU\MinnowBoardMax\flash.ffu' ,

ten spustí náš nástroj, ktorý prekonvertuje FFU do formátu VHD a následne ho integruje do našeho súborového systému. V správe diskov (Disk Management) sa nám objaví náš nový disk so 4 partíciami. Odpojíme ho a zobrazíme uloženie VHD obrazu. Z tohto priečinka si obraz skopírujme a použijeme na vytvorenie nového virtuálneho stroja. Virtuálny stroj zakladáme s parametrami, ako sú: Operačný systém Windows 10, typ firmvéru UEFI, typ virtuálneho disku SATA. Pri možnosti vytvorenia disku zadáme existujúci disk a pridáme náš obraz VHD. Je dobré vyhradiť virtuálnemu stroju aspoň 2 GB pamäte. Čo sa týka sieťových nastavení, odporúčame vytvoriť dve sieťové karty, jedna pre lokálne prepojenie pre potreby vzdialenej

správy a druhá pre zdieľanie internetového pripojenia počítača. Základné prihlasovacie údaje do operačného systému sú Meno: Administrator Heslo: p@ssw0rd .

5.4 Testovanie aplikácie

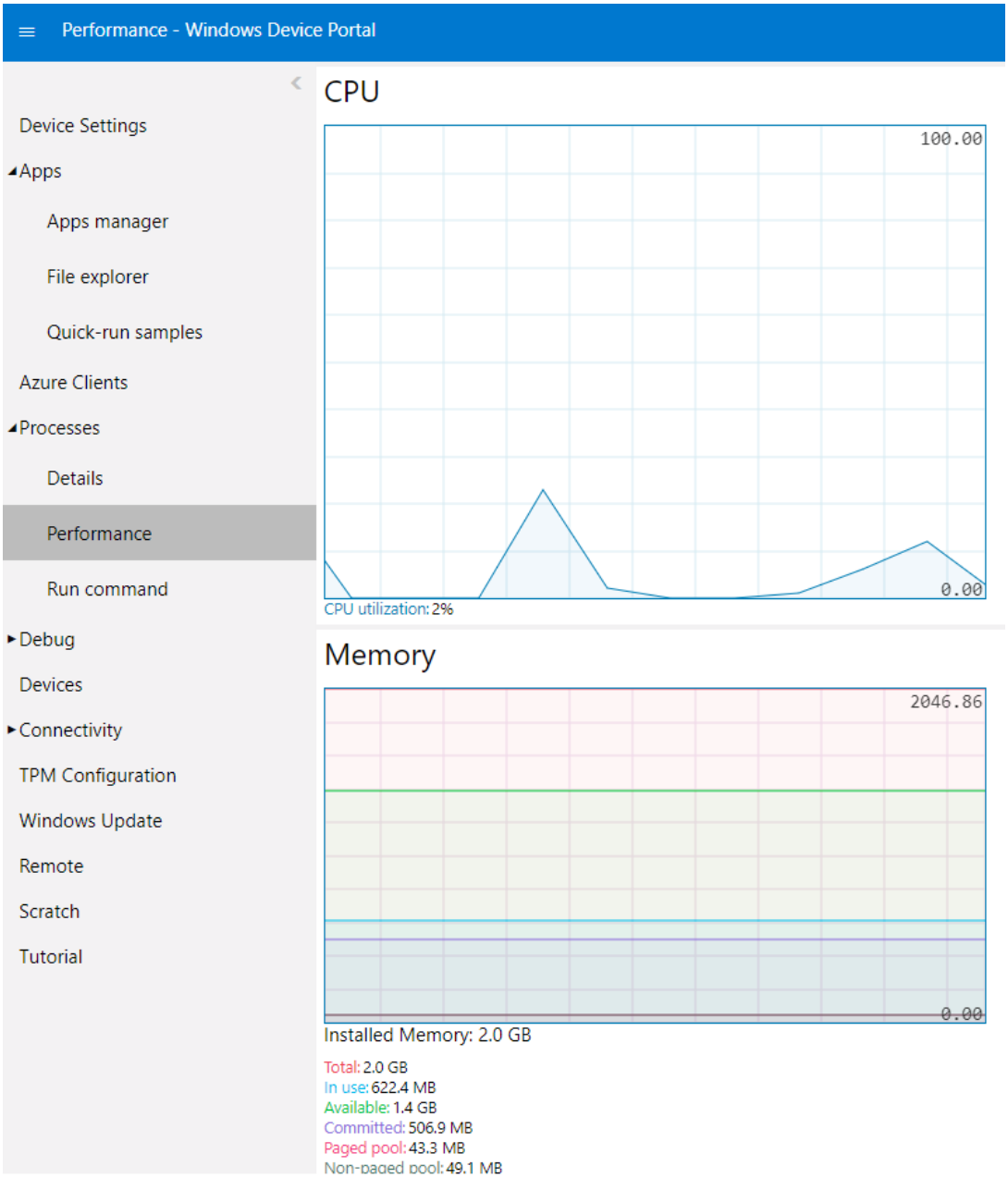
Aplikáciu sme testovali na virtuálnom zariadení, ktoré predstavovalo vývojovú dosku MinnowBoard Max s operačným systémom Windows 10 IoT Core. Výhodou je, že sa tento virtuálny stroj dá sieťovo prepojiť a simuluje sa tak reálna prevádzka zariadenia. Tým pádom je možnosť pri vývoji aplikáciu rovno spúšťať z Visual Studia na cieľové vzdialené zariadenie (remote device). Bezproblémovo funguje aj vzdialená správa tohto zariadenia z webového rozhrania, ktoré predstavuje IP virtuálneho zariadenia a port 8080, kde sa po prihlásení za administrátora dostaneme k celému radu možností na správu. Pre nás dôležitou možnosťou je Aplikačný manažér (Apps manager Obr. 5.4) v sekcií Apps, kde máme prehľad nahratých aplikácií na zariadení, a či sú spustené. Taktiež ich môžeme sami spustiť a zastaviť či rovno nainštalovať buď z lokálneho, alebo sieťového úložiska.



Obr. 5.4: Aplikačný manažér vzdialenej správy, zvýraznenie našej aplikácie.

Vo File explorer (sekcia Apps) sa dajú prehliadať súbory, čo znamená, že máme prístup k zašifrovanému reťazcu znakov, ktorý sa nám tam pri šifrovaní uložil. V prípade potreby dokážeme nahrať do tohto súboru vlastný šifrovaný reťazec, ktorý chceme zas na zariadení dešifrovať. Využitie systémových zdrojov, ako sú procesor, pamäť,

rozhrania a sieť môžeme kontrolovať v Performance (sekcia Apps). Tu sme namerali, že naša aplikácia zaberá v pamäti približne 55 MB. Pri šifrovaní predstavuje maximálne zaťaženie procesora v jeden moment približne 22 % (Obr. 5.5).



Obr. 5.5: Aplikačný manažér vzdialenej správy, výkonnostný monitoring.

Týmto krokom sme dokázali, že nie je potrebné pre vývoj takéhoto typu aplikácií reálny hardvér. Vývojárovi stačí virtuálny stroj, ktorý je plne funkčný a splní všetky potrebné požiadavky v porovnaní s reálnym zariadením. Pre špeciálne aplikácie, kde sa využívajú rôzne snímače pripojené priamo do vývojovej dosky je už ale hardvérové riešenie potrebné.

5.5 Bezpečnosť implementácie pre IoT zariadenie

Zabezpečenie vstupu do aplikácie je riešené prihlásením sa cez externú autoritu (v tomto prípade Microsoft). Po úspešnom prihlásení sa nám vráti token, v ktorom je presne definované, čo môže prihlásený užívateľ s dátami a zdrojmi aplikácie vykonávať a ku ktorým má vôbec prístup. Tieto hodnoty môže nastaviť vývojár, či iná poverená osoba v portáli Microsoft Azure. Taktiež v tomto portáli možno definovať presnú skupinu užívateľov, ktorí budú mať prístup do aplikácie. Je výhodou, že samotné prihlasovanie, overovanie, či nastavovanie oprávnení sa robí externe a nie sme závislí len čisto od zariadenia, pretože to môže byť vystavené útočníkom po softvérovej, ako aj fyzickej stránke. Problémom je práve spomínaná hardvérová časť zariadenia, ktorá môže byť vystavená širokej verejnosti v podobe platobných terminálov, reklamných pútačov, pokladní, automatov, atď. Preto by sa mali citlivé údaje ukladať mimo týchto zariadení a v prípade potreby si ich načítať a uchovať len pre nevyhnutný čas.

Bezpečnosť samotnej ľahkej blokovej šifry Speck

Do dnešnej doby nie je známy úspešný útok na túto šifru. Jedine varianty, ktoré majú redukovaný počet kôl boli prelomené. Pri našej použitej šifre variantu Speck 128/128, bolo celkových 32 kôl redukovaných na 23, čo predstavuje 72 percent. Typ útoku bola diferenciálna kryptoanalýza, ktorá sa osvedčila ako najlepší a najúčinnejší spôsob prelomenia [91]. Keďže táto šifra je typu ARX, nepoužíva S-boxy, alebo iné vyhľadávacie tabuľky, a preto je oproti šifre AES imúnna voči útokom bočným kanálom, ako je útok na časovanie vyrovnávacej pamäte (cache-timing attacks). Ako skoro každá bloková šifra je aj Speck zraniteľná voči analýze napájania (power analysis).

6 Záver

Jedným z cieľov tejto práce bola analýza možností bezpečného uloženia šifrovacích a autentizačných kľúčov na PC a na zariadení využívanom v internete vecí. Pri tejto analýze v odvetví PC sme si rozdelili rišenia na hardvérové a softvérové. Hardvérové (HSM, TPM) majú výhodu, že ponúkajú silnú bezpečnosť kľúčov, hlavný kľúč nikdy neopustí toto zariadenie, iba sa z neho odvodzujú ostatné kľúče. Sú odolné voči väčšine známych softvérových útokov a certifikované zariadenia aj proti hardvérovým. Nevýhodou je ich cena. Ďalej sme zanalyzovali softvérové riešenia (programy, funkcie), kde sme zamerali hlavne na bezpečnostné nedostatky, ktoré často spočívajú v zlej implementácii šifier a systému manipulácie s údajmi ako takými programom, ako aj slabé heslá zadané používateľmi. Práve použitie silných hesiel eliminuje podstatnú časť bezpečnostných dier. Také heslo by malo mať dĺžku 12 a viac znakov s použitím čísel a špeciálnych atribútov zároveň, a taktiež spĺňať dokonalú náhodnosť. To znamená, že heslo by nemalo obsahovať známe slová, a zároveň celé heslá by sa nemali používať viackrát.

V prostredí internetu vecí sme si zariadenia rozdelili do 4 tried podľa výkonu, použitia, architektúry, atď., pretože existuje veľký počet typov zariadení, architektúr, riešení. Celkovo musíme hodnotiť, že zariadenia IoT majú problém s fyzickým prístupom k dátam, pretože sú často vystavené verejným priestranstvám, a tým aj potencionálnym útočníkom. Tento problém dokážu riešiť špeciálne moduly, čipy, či integrované obvody, ktoré vykonávajú kryptografické operácie a dokážu detekovať nežiadúcu fyzickú manipuláciu so zariadením. Trendom sú aj vstavané bezpečnostné riešenia ako TrustZone a CryptoCell na architektúre ARM. Špeciálne čipy, či celé moduly, tým, že vykonávajú operácie za hlavné CPU oveľa efektívnejšie, znižujú radikálne potrebu výpočtového výkonu a elektrickej energie, čo je veľmi žiadané v prostredí IoT.

Priblížili sme si aj šifrovací algoritmus AES v móde XTS a CBC, ktorý sme otestovali nástrojom OpenSSL, kde sme potvrdili teóriu, že mód XTS je rýchlejší pri šifrovaní väčších súborov, a že hardvérová akcelerácia šifry AES pomocou nových inštrukcií rapídne (3 až 10-krát) zvyšuje rýchlosť šifrovania a dešifrovania. Taktiež sme vykonali merania vybraných ľahkých blokových šifier (Simon, Speck, LEA), kde sme ukázali rozdiel v rýchlostiach medzi jednotlivými architektúrami a platformami. Zhodnotili sme tiež, že hardvérovo orientované šifry sú na nízko výkonných zariadeniach rýchle a efektívne.

Obidva naše návrhy implementácie bezpečného uloženia kľúčov boli softvérové riešenia, kde sme pri platforme PC vytvorili konzolovú aplikáciu (v jazyku C#), ktorá je schopná šifrovať reťazec znakov. Použili sme šifru AES-128 v móde CBC. Jedná sa o bezpečnú a rýchlu šifru. Bezpečne vygenerovaný kľúč a vektor používaný

pri operáciách šifry sme nechceli ukladať externe (napr. do textového súboru), tak ich zapisujeme pre zvýšenie bezpečnosti priamo do binárneho kódu programu, ktorý sme z časti zneprehliadnili. Pri šifrovaní/dešifrovaní je nutné zadať heslo, ktoré je špecifické tým, že program pri overovaní neberie do úvahy len znaky, ktoré zadávame ale aj časové medzery medzi nimi. Tým dokážeme eliminovať viaceré škodlivé softvéry na zachytávanie vstupu klávesnice, ako aj rôzne formy sociálneho inžinierstva. Pri testovaní náročnosti na výpočtový výkon, v porovnaní s podobným programom (AESCrypt), náš návrh nevykazoval nadmerné nároky na výpočtový výkon. Do prostredia IoT sme navrhli a vytvorili aplikáciu, ktorá je schopná šifrovat reťazec znakov. Využíva k tomu ľahkú blokovú šifru Speck 128/128 v móde CTR. Jedná sa o bezpečnú a hlavne rýchlu šifru, pretože je stavaná pre zariadenia IoT. Na zabezpečenie prístupu do aplikácie pred nežiadúcimi osobami sme využili prihlásenie sa cez externú autoritu (v našom prípade Microsoft). Práve toto riešenie nám umožňuje ľahkú a bezpečnú správu prístupu a poverení pre jednotlivých užívateľov v portáli Microsoft Azure, ktorý je prepojený s našou aplikáciou prostredníctvom jedinečného identifikátora. Výhodou tejto aplikácie je, že ju dokážeme spustiť na operačnom systéme Windows 10 na normálnom PC, ako aj na špeciálnych vývojových doskách s upraveným operačným systémom Windows pre odvetvie IoT.

Nakoniec musíme podotknúť, že bezpečnosť uloženia šifrovacích a autentizačných kľúčov závisí od kombinácie softvérového a hardvérového riešenia, a taktiež od dodržiavania určitých bezpečnostných zásad v podaní užívateľov. Najväčšou bezpečnostnou hrozbou ostáva používateľ, ktorý často nedodržava bezpečnostné pokyny, alebo sa nechá oklamať a sám zverejní, alebo zníži bezpečnostnú ochranu citlivých údajov. Práve v dnešnej dobe, pri čoraz viac vyspelých bezpečnostných opatrenich sa dostáva do popredia sociálne inžinierstvo, pretože je to pre útočníkov oveľa jednoduchší spôsob ako sa dostať k informáciám, ktoré potrebujú.

Literatúra

[1] *TPM Main Specification* [online], 01. 03. 2011 [cit. 2018-12-06]. Dostupné z: <https://trustedcomputinggroup.org/resource/tpm-main-specification/>

[2] *Trusted Platform Module (TPM) Summary* [online], 04. 01. 2008 [cit. 2018-12-06]. <Dostupnéz:https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/>

[3] *How Windows 10 uses the Trusted Platform Module* [online], 27. 10. 2017 [cit. 2018-12-06]. Dostupné z: <https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/how-windows-uses-the-tpm>

[4] ROUSE, Margaret, Hardware security module (HSM). *TechTarget* [online]. September 2015 [cit. 2018-12-06]. Dostupné z: <https://searchaws.techtarget.com/definition/hardware-security-module-HSM>

[5] *NShield General Purpose HSMs* [online], [cit. 2018-12-06]. Dostupné z: <https://www.thalesesecurity.com/products/general-purpose-hsms>

[6] *Key Management with Utimaco CryptoServer* [online], [cit. 2018-12-06]. Dostupné z: <https://hsm.utimaco.com/wp-content/uploads/2017/09/Internal-vs.-external-Key-Storage-_White-Paper_Final.pdf>

[7] *SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES: FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION,* 2001. In: . Gaithersburg, MD: U.S. Government Printing Office Washington, FIPS PUB 140-2. Dostupné také z: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf>

[8] BARKER, Elaine a Nicky MOUHA, 2017. *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher* [online]. 2. National Institute of Standards and Technology [cit. 2018-12-10]. NIST SP 800-67. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf>

[9] KATZ, Jonathan a Yehuda LINDELL, 2014. *INTRODUCTION TO MODERN CRYPTOGRAPHY: CRYPTOGRAPHY AND NETWORK SECURITY*. 2. CRC Press. ISBN 978-1-4665-7027-6.

[10] *Announcing the ADVANCED ENCRYPTION STANDARD (AES)* [online], 2001. National Institute of Standards and Technology [cit. 2018-12-10]. FIPS

PUB 197. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>

[11] DAEMEN, Joan a Vincent RIJMEN, 2002. *The Design of Rijndael: AES - The Advanced Encryption Standard.* 1. Springer. ISBN 3540425802.

[12] LUTHER, Martin, XTS: A Mode of AES for Encrypting Hard Disks. *Computing Now* [online]. 24 May 2010 [cit. 2018-12-10]. DOI: 10.1109/MSP.2010.111. ISSN 1558-4046. Dostupné z: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5470958>

[13] HOFEMEIER, Gael a Robert CHESEBROUGH, 2012. *INTRODUCTION TO INTEL AES-NI AND INTEL SECURE KEY INSTRUCTIONS* [online]. Intel [cit. 2018-12-10]. Dostupné z: <https://software.intel.com/sites/default/files/m/d/4/1/d/8/Introduction_to_Intel_Secure_Key_Instructions.pdf>

[14] ROTT, Jeffrey, *Intel Advanced Encryption Standard Instructions (AES-NI)* [online]. 2. February 2012 [cit. 2018-12-10]. Dostupné z: <https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni>

[15] VAN TILBORG, Henk C. A. a Sushil JAJODIA, ed., 2011. *ENCYCLOPEDIA OF CRYPTOGRAPHY AND SECURITY.* 2. Springer Science & Business Media. ISBN 978-0387-23473-1.

[16] BRAGG, Roberta, *The Encrypting File System* [online]. Microsoft, 29. 06. 2009 [cit. 2018-12-11]. Dostupné z: <https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc700811(v%3dtechnet.10)>

[17] *Encrypting File System (EFS) on Windows 10 explained* [online], TheWindowsClub, 22. August 2018 [cit. 2018-12-11]. Dostupné z: <https://www.thewindowsclub.com/encrypting-file-system-efs-windows-10>

[18] BHAMBURKAR, Ajinkya, Cracking Windows 10 Account Password: Is it Still Possible? How to Prevent it?. *Guiding Tech* [online]. 31. December 2016 [cit. 2018-12-11]. Dostupné z: <https://www.guidingtech.com/61991/cracking-windows-10-password-prevent/>

[19] HUDEK, Ted, *Personal Information Exchange (.pfx) Files* [online]. Microsoft, 20. 04. 2017 [cit. 2018-12-11]. Dostupné z: <https://docs.microsoft.com/en-us/windows-hardware/drivers/install/personal-information-exchange---pfx--files>

[20] *Encrypting File System* [online], Microsoft, 02. 07. 2012 [cit. 2018-12-11]. Dostupné z: <https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc749610(v=ws.10)>

[21] *MacOS Security: Overview for IT* [online], Apple, March 2018 [cit. 2018-12-06]. Dostupné z: <https://www.apple.com/business/resources/docs/macOS_Security_Overview.pdf>

[22] *Keychain Items* [online], Apple [cit. 2018-12-06]. Dostupné z: <https://developer.apple.com/documentation/security/keychain_services/keychain_items>

[23] *Príručka užívateľa pre Kľúčenku: Zabezpečené poznámky* [online], Apple [cit. 2018-12-06]. Dostupné z: <https://support.apple.com/sk-sk/guide/keychain-access/aside/kyca9f3803ce/10.5/mac/10.14>

[24] *Pomocník pre Kľúčenku* [online], Apple [cit. 2018-12-06]. Dostupné z: <https://support.apple.com/sk-sk/guide/keychain-access/kyca2429/10.0/mac/10.13>

[25] *Access Control Lists* [online], Apple [cit. 2018-12-06]. Dostupné z: <https://developer.apple.com/documentation/security/keychain_services/access_control_lists>

[26] *GNOME/Keyring* [online], [cit. 2018-12-06]. Dostupné z: <https://wiki.archlinux.org/index.php/GNOME/Keyring>

[27] *Gnome Keyring's 'Secure' Memory* [online], 26. 11. 2013 [cit. 2018-12-06]. Dostupné z: <https://wiki.gnome.org/Projects/GnomeKeyring/Memory>

[28] *Gnome-keyring Security Philosophy* [online], [cit. 2018-12-06]. Dostupné z: <https://wiki.gnome.org/Projects/GnomeKeyring/SecurityPhilosophy>

[29] *Security FAQ* [online], [cit. 2018-12-06]. Dostupné z: <https://wiki.gnome.org/Projects/GnomeKeyring/SecurityFAQ>

[30] *CNG Cryptographic Algorithm Providers* [online], Microsoft, 31. 05. 018 [cit. 2018-12-11]. Dostupné z: <https://docs.microsoft.com/en-us/windows/desktop/seccertenroll/cng-cryptographic-algorithm-providers>

[31] BICHSEL, Andrea, et al, *How Windows 10 uses the Trusted Platform Module* [online]. Microsoft, 27. 10. 2017 [cit. 2018-12-11]. Dostupné z: <https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/how-windows-uses-the-tpm>

[32] *Key Storage and Retrieval* [online], Microsoft, 31. 05. 2018 [cit. 2018-12-11]. Dostupné z: <https://docs.microsoft.com/en-us/windows/desktop/seccng/key-storage-and-retrieval>

[33] *CryptProtectMemory function* [online], Windows, 11. 09. 2018 [cit. 2018-12-06]. Dostupné z: <https://docs.microsoft.com/en-us/windows/desktop/api/dpapi/nf-dpapi-cryptprotectmemory>

[34] *Encryption Scheme* [online], VeraCrypt [cit. 2018-12-11]. Dostupné z: <https://www.veracrypt.fr/en/Encryption%20Scheme.html>

[35] *VeraCrypt main features* [online], VeraCrypt [cit. 2018-12-11]. Dostupné z: <https://www.veracrypt.fr/en/Home.html>

[36] KESSLER, Gary C., 2013. *AES Crypt User Guide* [online]. Packetizer [cit. 2018-12-11]. Dostupné z: <https://www.aescrypt.com/documentation/AES%20Crypt%20User%20Guide.pdf>

[37] *AES Crypt* [online], Packetizer [cit. 2018-12-11]. Dostupné z: <https://www.aescrypt.com/>

[38] PAVLOV, Igor, *7z Format* [online]. [cit. 2018-12-11]. Dostupné z: <https://www.7-zip.org/7z.html>

[39] ROSE, Karen, Scott ELDRIDGE a Lyman CHAPIN, MARSAN, Carolyn, ed., 2015. *The Internet of Things: An Overview: Understanding the Issues and Challenges of a More Connected World.* Geneva, Switzerland: The Internet Society.

[40] LÉVY-BENCHETON, Cédric, Eleni DARRA, Guillaume TÉTU, Guillaume DUFAY a Mouhannad ALATTAR, 2015. *Security and Resilience of Smart Home Environments: Good practices and recommendations.* Heraklion, Greece: European Union Agency for Network and Information Security. ISBN 978-92-9204-141-0. DOI:10.2824/360120.

[41] BORMANN, C., M. ERSUE a A. KERANEN, 2014. RFC 7228: Terminology for Constrained-Node Networks. *Internet Engineering Task Force (IETF)* [online]. [cit. 2018-12-05]. DOI: 10.17487/RFC7228. ISSN 2070-1721. Dostupné z: <https://www.rfc-editor.org/rfc/rfc7228.txt>

[42] CHOI, EUNYOUNG a HAERYONG PARK, 2017. A study on encryption key management technology in a light IoT device environment. *International Journal of Innovative Research in Technology & Science* [online]. (5), 3-8 [cit. 2018-12-05]. ISSN 2321-1156. Dostupné z: <http://ijirts.org/volume5issue5/IJIRTSV5I5010.pdf>

[43] *Key injection* [online], [cit. 2018-12-06]. Dostupné z: <https://hsm.utimaco.com/solutions/applications/key-injection/>

[44] ZIGBEE STANDARDS ORGANIZATION, *ZIGBEE SPECIFICATION* [online], 2012. Document 053474r20. San Ramon, CA: ZigBee Alliance [cit. 2018-12-06]. Dostupné z: <http://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf>

[45] ZILLNER, Tobias, ZIGBEE EXPLOITED: The good, the bad and the ugly. *Blackhat* [online]. 06. 09. 2015 [cit. 2018-12-06]. Dostupné z: <https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly-wp.pdf>

[46] ABR a BBR, 2018. *Z-Wave Security Whitepaper: Introduction to the Z-Wave Security ecosystem* [online]. 3. Silicon Labs [cit. 2018-12-06]. Dostupné z: <https://www.silabs.com/documents/login/white-papers/INS13474-Z-Wave-Security-Whitepaper.pdf>

[47] MHANSEN, MVO, ANI a JFR, 2018. *500 Series Z-Wave Chip Programming Mode* [online]. 13. Silicon Labs [cit. 2018-12-06]. Dostupné z: <https://www.silabs.com/documents/login/user-guides/INS11681-Instruction-500-Series-Z-Wave-Chip-Programming-Mode.pdf>

[48] CHU, Fan, James NICHOLSON a Yaner WANG, 2014. *F-RAM, nvSRAM, and MRAM Magnetic Field Immunity* [online]. San Jose, CA: Cypress Semiconductor [cit. 2018-12-07]. Dostupné z: <http://www.cypress.com/file/46731/download>

[49] *FRAM FAQs* [online], 2014. Dallas, Texas: Texas Instruments [cit. 2018-12-07]. Dostupné z: <http://www.ti.com/lit/ml/slat151/slat151.pdf>

[50] Key Security is Foundational to IoT Application Security, *Digi-Key Electronics* [online]. 03. 05. 2016 [cit. 2018-12-07]. Dostupné z: <https://www.digikey.com/en/articles/techzone/2016/may/key-security-is-foundational-to-iot-application-security>

[51] *CryptoAuthentication Family of Crypto Elements with Hardware-Based Key Storage: ATSHA204A, ATAES132A and ATECC508A* [online], 2015. San Jose, CA: Atmel Corporation [cit. 2018-12-07]. Dostupné z: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8756-ATSHA204A-ATAES132A-ATECC108A-ATECC508A-Flyer-E.pdf>

[52] *CryptoAuthentication ATECC508A: Crypto Element with ECDH and ECDSA* [online], 2015. San Jose, CA: Atmel Corporation [cit. 2018-12-07]. Dostupné z: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8938-Crypto-ATECC508A-Flyer.pdf>

[53] YADAV, Govid, Cryptographic Key Storage Options & Best Practices. *GlobalSign* [online]. 11. 07. 2017 [cit. 2018-12-07]. Dostupné z: <https://www.globalsign.com/en/blog/cryptographic-key-management-and-storage-best-practice/>

[54] *Securing Your Raspberry Pi: Best practises, and the role of Hardware Security Modules* [online], [cit. 2018-12-07]. Dostupné z: <https://www.zymbit.com/securing-raspberry-pi/>

[55] *CryptoAuthentication Device Summary Datasheet: ATECC608A* [online], 2018. U.S.A.: Microchip Technology [cit. 2018-12-07]. ISBN 978-1-5224-3616-4. Dostupné z: <http://ww1.microchip.com/downloads/en/DeviceDoc/ATECC608A-CryptoAuthentication-Device-Summary-Data-Sheet-DS40001977B.pdf>

[56] *Platform Architecture* [online], 2018 [cit. 2018-12-07]. Dostupné z: <https://developer.android.com/guide/platform/>

[57] *Android keystore system* [online], 2018 [cit. 2018-12-07]. Dostupné z: <https://developer.android.com/training/articles/keystore>

[58] *KeyChain* [online], 2018 [cit. 2018-12-07]. Dostupné z: <https://developer.android.com/reference/android/security/KeyChain>

[59] COOIJMANS, Tim, 2014. *Secure Key Storage and Secure Computation in Android.* Nijmegen, Holandsko. Master Thesis. Radboud University Nijmegen.

[60] WESTERMAN, Wayne, Byron HAN a Craig MARCINIAK, 2013. Efficient Texture Comparison. U.S.A. 20130308838. Zapsáno March 12, 2013.

[61] *Storing Keys in the Secure Enclave* [online], [cit. 2018-12-07]. Dostupné z: <https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave>

[62] POT, Justin, What Is Apple's "Secure Enclave", And How Does It Protect My iPhone or Mac?. *How-To Geek* [online]. 22. 01. 2018 [cit. 2018-12-07]. Dostupné z: <https://www.howtogeek.com/339705/what-is-apples-secure-enclave-and-how-does-it-protect-my-iphone-or-mac/>

[63] *GUILBON, Joffrey, Introduction to Trusted Execution Environment: ARM's TrustZone* [online]. 19. 06. 2018 [cit. 2018-12-07]. Dostupné z: <https://blog.quarkslab.com/introduction-to-trusted-execution-environment-arms-trustzone.html>

[64] *HAYTON, Richard, Trusted execution environments: What, how and why?. TechTarget* [online]. 18. 04. 2018 [cit. 2018-12-07]. Dostupné z: <https://internetofthingsagenda.techtarget.com/blog/IoT-Agenda/Trusted-execution-environments-What-how-and-why>

[65] *An Exploration of ARM TrustZone Technology* [online], Genode Labs [cit. 2018-12-07]. Dostupné z: <https://genode.org/documentation/articles/trustzone>

[66] *MITIGATING PHYSICAL ATTACKS* [online], [cit. 2018-12-07]. Dostupné z: <https://www.arm.com/products/silicon-ip-security/physical-security-solutions?_ga=2.73369299.1524321125.1543699569-2080943332.1543113511>

[67] WALRATH, Josh, ARM Introduces Physical Security to Cortex-M Line. *PC perspective* [online]. 02. 05. 2018 [cit. 2018-12-07]. Dostupné z: <https://www.pcper.com/category/tags/cortex-m35p>

[68] *CRYPTOCELL-300 FAMILY* [online], [cit. 2018-12-07]. Dostupné z: <https://www.arm.com/products/silicon-ip-security/crypto-cell-300>

[69] *Arm TrustZone CryptoCell-712: FIPS 140-2 Non-Proprietary Security Policy*, 2018. Revision 1.19. Cambridge, England: Arm Limited.

[70] BEAULIEU, Ray, D. SHORS, J. SMITH, S. TREATMAN-CLARK, B. WEEKS a L. WINGERS, 2013. *The Simon and Speck Families od Lightweight Block Ciphers* [online]. 9800 Savage Road, Fort Meade, MD 20755, USA: National Security Agency [cit. 2019-05-23]. Dostupné z: <https://eprint.iacr.org/2015/585.pdf>

[71] HONG, D., J. LEE, D. KIM, D. KWON, K. RYU a D. LEE, 2014. *LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors.* Springer, Cham. ISBN 978-3-319-05148-2.

[72] GUO, J., T. PEYRIN, A. POSCHMANN a M. ROBSHAW, 2011. *The LED block cipher. Cryptographic Hardware and Embedded Systems–CHES 2011* [online]. Berlin Heidelberg: Springer, s. 326–341 [cit. 2019-05-23]. Dostupné z: <https://eprint.iacr.org/2012/600.pdf>

[73] *OpenSSL: Cryptography and SSL / TLS Toolkit* [online], [cit. 2018-12-11]. Dostupné z: <https://www.openssl.org/>

[74] WAHID, Nazeh, Ali A a MARWAN, 2018. *A Comparison of Cryptographic Algorithms: Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention* [online]. J Comp Sci Appl Inform Technol [cit. 2018-12-10]. Dostupné z: <https://symbiosisonlinepublishing.com/computer-science-technology/computerscience-information-technology32.pdf>

[75] PATE, Steve a Kelvin PRYSE, *Intel AES-NI Performance Enhancements: HyTrust DataControl Case Study* [online]. 4. September 2014 [cit. 2018-12-10]. Dostupné z: <https://software.intel.com/en-us/articles/intel-aes-ni-performance-enhancements-hytrust-datacontrol-case-study>

[76] *AES-NI SSL Performance: a study of AES-NI acceleration using LibreSSL, OpenSSL* [online], 8. November 2018 [cit. 2018-12-10]. Dostupné z: <https://calomel.org/aesni_ssl_performance.html>

[77] MDAXINI, *OpenSSL Cipher Speed* [online]. 30. October 2014 [cit. 2018-12-10]. Dostupné z: <https://github.com/mdaxini/howto-openssl/wiki/OpenSSL-Cipher-Speed>

[78] Intel Hardware-based Security Technologies Bring Differentiation to Biometrics Recognition Applications Part 2, *Code Project* [online]. 1. February 2016 [cit. 2018-12-10]. Dostupné z: <https://www.codeproject.com/Articles/1075851/Intel-Hardware-based-Security-Technologies-Bring#_Toc435085955>

[79] MEDVED, Josip, *Speeding-Up GELI on NAS4Free* [online]. 31. 10. 2017 [cit. 2018-12-10]. Dostupné z: <https://www.medo64.com/2017/10/speeding-up-geli-on-nas4free/>

[80] FELICS: Fair Evaluation of Lightweight Cryptographic Systems, *Cryptolux* [online]. 2016 [cit. 2019-05-23]. Dostupné z: <https://www.cryptolux.org/index.php/FELICS>

[81] FELICS Block Ciphers Brief Results, *Cryptolux* [online]. [cit. 2019-05-23]. Dostupné z: <https://www.cryptolux.org/index.php/FELICS_Block_Ciphers_Brief_Results>

[82] Crypto++ Library 8.2, *Cryptopp* [online]. [cit. 2019-05-23]. Dostupné z: <https://www.cryptopp.com/>

[83] System.Security.Cryptography Namespace, *Microsoft* [online]. [cit. 2019-05-23]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography?view=netframework-4.8>

[84] ResourceLib C File Resource Management Library, *Github* [online]. [cit. 2019-05-23]. Dostupné z: <https://github.com/resourcelib/resourcelib>

[85] *ConfuserEx* [online], [cit. 2019-05-23]. Dostupné z: <https://yck1509.github.io/ConfuserEx/>

[86] LAKE, JOSH, *What is AES encryption and how does it work?* [online]. October 5, 2018 [cit. 2019-05-23]. Dostupné z: <https://www.comparitech.com/blog/information-security/what-is-aes-encryption/>

[87] Simon-speck-c, *Github* [online]. [cit. 2019-05-23]. Dostupné z: <https://github.com/Naruto/simon-speck-c>

[88] What's a Universal Windows Platform (UWP) app?, *Microsoft* [online]. 07. 05. 2018 [cit. 2019-05-23]. Dostupné z: <https://docs.microsoft.com/sk-sk/windows/uwp/get-started/universal-application-platform-guide>

[89] Call the Microsoft Graph API from a Windows Desktop app, *Microsoft* [online]. 04/10/2019 [cit. 2019-05-23]. Dostupné z: <https://docs.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-windows-desktop>

[90] Downloads: Windows 10 IoT Core Dashboard, *Microsoft* [online]. 05/15/2019 [cit. 2019-05-23]. Dostupné z: <https://docs.microsoft.com/en-us/windows/iot-core/downloads>

[91] *Notes on the design and analysis of Simon and Speck* [online], 19 January 2018 [cit. 2019-05-23]. Dostupné z: <https://eprint.iacr.org/2017/560.pdf>

Zoznam symbolov, veličín a skratiek

TPM	Trusted Platform Module – Modul dôveryhodnej platformy
PIN	Personal Identification Number – Osobné identifikačné číslo
HSM	Hardware security module – Hardvérový bezpečnostný modul
IoT	Internet of Things – Internet vecí
AES	Advanced Encryption Standard – Pokročilý šifrovací štandard
CBC	Cipher block chaining – Reťazové šifrovanie blokov
XTS	XEX-based tweaked-codebook – Utrhnutý kódový zoznam založený na XEX
3DES	Triple Data Encryption Standard – Trojnásobný štandard šifrovania údajov
NIST	National Institute of Standards and Technology – Národný inštitút pre štandardy a technológie
ECB	Electronic Code Book – Elektronická kódova kniha
OFB	Output FeedBack – Výstupná spätná väzba
CTR	Counter Mode – Režim počítadla
CCM	Counter Mode Cipher Block Chaining MAC – Režim počítadla šifrovaných reťazených blokov MAC
GCM	Galois/Counter Mode – Galois/Režim počítadla
CWC	Carter–Wegman CTR mode
OCB	Offset Codebook Mode – Režim kompenzácie kódov
XOR	eXclusive OR
XEX	XOR Encrypt XOR – XOR šifrovanie XOR
ACL	Access Control List – Zoznam pre riadenie prístupu
ACE	Access Control Entry – Vstup riadenia prístupu
SID	Security Identifier – Jednoznačný bezpečnostný identifikátor
EFS	Encrypting File System – Systém šifrovania súborov
DESX	XORed DES – DES s prídavným XORom
NTFS	New Technology File System – Nová technológia súborového systému
FEK	File Encryption Key – Kľúč na šifrovanie súborov
RSA	Rivest–Shamir–Adleman
DPAPI	Data Protection application programming interface – Rozhranie programovania aplikácií na ochranu údajov
NTLM	NT LAN Manager
PBKDF2	Password Based Key Derivation Function – Funkcia odvodzovania kľúčov založená na heslách
PFX	Personal Information Exchange – Výmena osobných informácií
URL	Uniform Resource Locator – Jednotný lokátor zdrojov

GNOME	GNU Network Object Model Environment – GNU sieťový objekt modelového prostredia
CNG	Cryptography API: Next Generation – Kryptografické aplikačné rozhranie novej generácie
KSP	Key Storage Provider – Poskytovateľ kľúčových úložísk
LSA	Local Security Authority – Lokálna bezpečnostná autorita
LRPC	Local Remote Procedure Call – Lokálny vzdialený procedurálny hovor
DSA	Digital Signature Algorithm – Digitálny podpisový algoritmus
SHA	Secure Hash Algorithm – Bezpečný hash algoritmus
IETF	Internet Engineering Task Force – Komisia pre technickú stránku internetu
ENSIA	European Union Agency for Network and Information Security – Agentúra Európskej únie pre sieťovú a informáčnú bezpečnosť
RFC	Request For Comments – Žiadosť pre komentáre
RAM	Random Access Memory – Pamäť s priamym prístupom
CoAP	Constrained Application Protocol – Obmedzený protokol aplikácie
FRAM	Ferroelectric Random Access Memory – Feroelektrická pamäť s náhodným prístupom
DPA	Differential power analysis – Analýza diferenčného výkonu
IPE	IP Encapsulation – Zapuzdrenie IP
OTP	One-time programmable – Jednorázovo programovateľné
PUF	Physically Unclonable Functions – Fyzicky neklonovateľné funkcie
TEE	Trusted Execution Environment – Bezpečné operačné prostredie
SEP	Secure Enclave Processor – Procesor bezpečnostnej enklávy
REE	Rich Execution Environment – Bohaté prostredie vykonávania
SRAM	Static Random Access Memory – Statická pamäť s náhodným prístupom
ASLR	Address space layout randomization – Náhodnosť usporiadania adresového miesta
ARX	add–rotate–xor
FELICS	Fair Evaluation of Lightweight Cryptographic Systems – Spravodlivé hodnotenie ľahkých kryptografických systémov
XAML	Extensible Application Markup Language – Rozšíriteľný aplikačný značkovací jazyk
JSON	JavaScript Object Notation – JavaScript objektová notácia
API	Application programming interface – Rozhranie pre programovanie aplikácií

Zoznam príloh

A Obsah priloženého CD

Na priloženom CD sa nachádza práca vo formáte PDF. Ďalej sú obsahom dve zložky so zdrojovými kódmi programov.

V prvej zložke **UWP_SimonSpeck** sa nachádza celý projekt (Visual studio 2019) aplikície pre zariadenia IoT. Aplikácia bola testovaná na virtuálnom zariadení s operačným systémom Windows 10 IoT Core (verzia pre vývojovú dosku MinnowBoard MAX 10.0.17763.107), a tiež na Windows 10 (verzia 1807). Pre načítanie projektu treba mať nainštalovaný najnovší Software development kit, a Visual studio s podporou tvorby UWP aplikácie.

V druhej zložke **program_CMD_Sulic** sa nachádza celý projekt (Visual studio 2019) konzolovej aplikácie pre PC, ako aj spustiteľný súbor (samostatná aplikácia) **program_CMD.exe**, ktorá má v sebe integrovanú knižnicu Vestris.ResourceLib a z časti zneprehľadnený kód. Aplikácia bola testovaná na PC s operačným systémom Windows 10 (verzia 1807) s nainštalovaným .NET Framework 4.7.2 .