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

KRYPTOGRAFICKÉ ALGORITMY PRO NÍZKOENERGETICKÁ ZAŘÍZENÍ INTERNETU VĚCÍ

CRYPTOGRAPHIC ALGORITHMS FOR LOW-POWER IOT DEVICES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Matěj Oszelda

AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Dzurenda, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Matěj Oszelda ***ID:*** 241081

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Kryptografické algoritmy pro nízkoenergetická zařízení internetu věcí

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku vývoje aplikací pro nízkoenergetická zařízení internetu věcí včetně možností využití operačních systémů IoT, jako je RIOT. Analyzujte možnosti současných kryptografických knihoven jazyku C/C++ pro jejich implementaci na výkonově, výpočetně a paměťově omezených zařízeních, jako jsou mikrokontrolery. Proveďte výkonové a paměťové srovnání kryptografických algoritmů různých knihoven se zaměřením na algoritmy vhodné pro autentizaci, ustavení klíčů, autentizované šifrování a výpočet autentizačních kódů. Po domluvě s vedoucím vyberte vhodnou platformu mikrokontroleru, navrhněte a implementujte praktickou aplikaci podporující bezpečnou komunikaci mezi zařízeními IoT.

DOPORUČENÁ LITERATURA:

[1] MENEZES, Alfred, Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] ŠŤOVÍČEK, Petr. Kryptografie na výkonově omezených zařízeních. Brno, 2020, 72 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 28.5.2024

Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce analyzuje možnosti aplikace operačních systémů, kryptografických knihoven a jejich primitiv v paměťově, výkonově a výpočetně omezeném prostředí internetu věcí. Následně implementuje vybrané nástroje pro zabezpečení komunikace v systému s omezenými zařízeními IoT. Teoretická část rozebírá problematiku IoT, jednotlivé operační systémy a kryptografické knihovny v tomto prostředí. Poté prezentuje měření a porovnání kryptografických knihoven a jejich primitiv. Na základě měření vybere nástroje pro implementaci v zabezpečení komunikace IoT zařízení. Následně vytvoří návrh a systém realizuje.

KLÍČOVÁ SLOVA

Kryptografie, internet věcí, operační systém, kryptografická knihovna, algoritmus, MbedTLS, TinyCrypt, MicroECC

ABSTRACT

The bachelor thesis analyses the possibilities of application of operating systems, cryptographic libraries and their primitives in memory, performance and computationally constrained IoT environment. It then implements selected tools for securing communication in a system with constrained IoT devices. The theoretical part discusses the IoT, the different operating systems and cryptographic libraries in this environment. It then presents the measurement and comparison of cryptographic libraries and their primitives. Based on the measurements, it selects tools for implementation in securing the communication of IoT devices. It then creates a design and implements the system.

KEYWORDS

Cryptography, IoT, operating system, cryptographic library, algorithm, MbedTLS, TinyCrypt, MicroECC

OSZELDA, Matěj. *Kryptografické algoritmy pro nízkoenergetická zařízení internetu věcí*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Matěj Oszelda
VUT ID autora:	241081
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Kryptografické algoritmy pro nízkoenergetická zařízení internetu věcí

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petrovi Dzurendovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V této práci je řešena problematika týkající se kryptografie a bezpečnosti na zařízeních v internetu věcí. Řeší se zde tedy malé zařízení, jenž jsou výkonnostně, výpočetně a energeticky omezená. Většinou mají takovéto nástroje autonomní napájení, jenž musí vydržet poměrně dlouho dobu. Takovéto nástroje se stále více rozšiřují a používají se například v zdravotnictví, chytrých městech, domácnostech, zemědělství, připojení aut a podobně. Dá se s určitostí říci, že v budoucnosti budou prvky internetu věcí všude kolem nás.

Často se jedná a nějaký 'senzor', který provádí nějaká měření, nebo veličinu monitoruje a svá zjištění odesílá k dalšímu využití. Zde přichází ono téma kryptografie a zabezpečování přenášených dat po bezdrátové síti. Komunikace přes otevřený prostor se dá totiž jednoduše odposlouchávat, rušit či dokonce upravovat. Proto je třeba komunikaci zabezpečit kryptografickými prvky. To může být ovšem problematické díky menšímu výkonu a paměti daných nástrojů a důsledkem tohoto problému zde nelze používat standardní kryptografické knihovny. Proto musí být kryptografické prvky modifikované pro integraci do prostředí v internetu věcí. V souvislosti s útoky je také problémové, že většina zařízení v internetu věcí má velikou hardwarovou zranitelnost oproti jiným běžně používaným zařízením.

Proto je v důležité orientovat se v problematice IoT a znát možnost využít kryptografické knihovny, jenž byly vyvinuty pro toto prostředí. Samozřejmě kryptografických knihoven existuje mnoho i v tomto prostředí a každá je stavěna pro trochu jiný systém a využití. Toto je důvod, proč je dobré umět analyzovat používaný systém a vybrat pro něj tu správnou knihovnu, nebo popřípadě operační systém. Následně je nutné být schopen kryptografické knihovny a jejich primitiva využit, testovat a porovnávat je mezi sebou. Přesně o tom bude jednat tato práce v následujících kapitolách.

1 Internet věcí

Prvně, než zde bude řešeno aplikování a výběr nástrojů pro zabezpečení určitých zařízení, se je nutné seznámit s základy problematiky týkající se internetu věcí a bezpečnosti zařízení, jenž se zde využívají.

V internetu věcí se používají sítě WSN (Wireless sensor network). WSN je síť uzlů propojených bezdrátovými médii, které spolupracují na snímání a kontrole okolního prostředí.Typicky má tento typ architektury tři komponenty: senzorové uzly, bránu a pozorovatele (uživatele). Obecně je třeba u WSN zohlednit některé aspekty: spolehlivost (schopnost senzoru udržet funkčnost sítě bez přerušení), hustotu uzlů (oblast pokrytí, spolehlivost a přesnost), latentní charakteristiky sítě (zpoždění, kapacita a robustnost) a složitost směrování dat, která závisí na topologii sítě. Dále se zde musí hledět na spotřebu energie při poslání každého uzlu, jelikož je většina malých zařízení v internetu věcí napájena skrze baterii, jenž musí vydržet dlouhou dobu.

1.1 Kryptografie

Kryptografie je zhruba rozdělena na asymetrickou a symetrickou kryptografii. Symetrická kryptografie používá pro šifrováni i dešifrování stejný klíč. Asymetrická používá dva klíče, veřejný při dešifrování a tajný pro šifrování. Většinou je poměrně obtížné, až pomalu z výpočetního hlediska s dnešními výpočetními možnostmi nemožné, zjistit z veřejného klíče tajný klíč. Symetrická kryptografie se často používá v přenosu tajného klíče pro asymetrickou kryptografii při digitálním podpisu. V zařízeních týkajících se věcí jako je řízením automobilu lze dopředu na vložit tajné klíče a následovně komunikovat symetricky. Toto ovšem nelze provést, když není známa druhá komunikující strana (komunikace mezi vozidly), zde již musíme použít veřejný klíč. Symetrická kryptografie je výhodnější, pokud máme zařízení s omezeným počtem zdrojů. Symetrická kryptografie se skládá ze základních funkcí, jako jsou blokové nebo proudové šifry a metod pro aplikaci základních funkcí na paket, jenž se nazývají režim činnosti blokové šifry. Aby jsme kryptografii odlehčili, je třeba zlepšit účinnost proudových a blokových šifer.

1.2 Kvantová kryptografie

V rámci této kapitoly bylo citováno z: [3, 21, 22] V této práci se nebude využívat prvky kvantové kryptografie, ovšem je dobré být seznámen s touto problematikou, jelikož jak už bylo dříve zmíněno, v budoucnosti se budou zařízení internetu věcí pohybovat všude kolem nás a kvantové počítače se postupně blíží, což znamená

prolomení mnoha kryptografických algoritmů, jenž jsou používány dnes. Prozatím není známé více robustní řešení bezpečnosti komunikace ve světě s kvantovými počítači než kvantová kryptografie.

V takovém stavu, v jakém je kvantová kryptografie dnes je velice nevýhodná a složitá pro aplikaci v internetu věcí, to je v důsledku malé výpočetní síly pro provádění složitých kvantových operací, jako je například generování a manipulace s kvantovými stavy a energetickou náročnost, jenž kvantová kryptografie obsahuje. Také jsou systémy v internetu věcí poměrně nízko nákladové a kvantová kryptografie je pravým opakem a je velmi nákladová. Ale i přesto existuje výzkum v oblasti kvantových technologií zaměřený na vytvoření efektivnějších a energeticky úsporných kvantových zařízení a protokolů, které by mohly být vhodné i pro toto prostředí.

V nynější době je velmi populární v kryptografii pro zařízení v internetu věcí používat eliptické křivky, ovšem s příchodem kvantových počítačů budou všechny tyto systémy lehce prolomeny, což znamená že v dlouhodobém hledisku nejsou tyto sítě dobře zabezpečeny. Tyto systému ponesou choulostivá data, jenž bude třeba zabezpečit. V budoucnosti tedy většinu kritické infrastruktury převezme internet věcí, pro který je ideální to, že kvantovou kryptografii lze použít jak v optických, tak i bezdrátových komunikacích.

Kvantová kryptografie je založená na principech kvantové mechaniky a na Heisenbergově principu neurčitosti, ten říká, že když nedojde k narušení systému, je těžké změřit kvantový stav jakéhokoliv systému. Polarizace světla nebo fotonové částice může být měřena jen v konkrétním čase. Princip polarizace fotonů popisuje způsob, jak světlo polarizuje fotony, nebo jak mohou být orientovány do určitých směrů. Fotonový filtr se správnou polarizací je navíc schopen detekovat pouze polarizovaný foton, jinak by byl foton zničen. Tento princip hraje kritickou roli v kvantové kryptografii, přesněji v kvantové distribuci klíčů.

Kvantová kryptografie nepřenáší žádnou zprávu, jen vytváří a přenáší klíče. Klič je vytvořen podle počtu fotonů, které se dostanou k příjemci a na způsobu přijímání daných fotonů. Díky tomu, že polarizace fotonů může být provedena v různých orientacích, mohou být použity k reprezentaci bitů obsahující jedničky a nuly. Bit s kvantovou informací se nazývá qubit. Foton zde může nabýt polarizace v rozsahu 0° až 180°. Uživatel může navrhnout klíč odesláním fotonů s náhodnou polarizací. Klíč může být přijat bez jakýchkoliv následků, jelikož je to jen sada náhodných bitů, které mohou být vyhozeny. Po přijetí klíče může být tento klíč použit k zašifrování zprávy, kterou lze následně poslat dále. Klíč se ke koncovým uživatelům přenáší pomocí kvantového kanálu a koncový uživatel použije veřejný kanál k ověření klíče.

Dosud nejpopulárnější protokol pro kvantové generování klíče je BB84 protocol, jenž využívá polarizačního stavu jednoho fotonu. Samozřejmě podobných protokolů

existuje více, některé využívají polarizace zapletených fotonů pro navázání bezpečné komunikace jako např. protokol E91. Pro zabezpečení komunikace v IoT je ovšem nejlepší dříve zmíněný BB84 protocol.

2 Operační systémy v internetu věcí

V této kapitole bylo citováno z následujících zdrojů: [2, 10, 1] Operační systém je důležitý pro jednoduší práci na mnoha zařízeních, ovšem na software pro zařízení v internetu věcí jsou pokládány mnohé požadavky. Operační systém musí zabírat málo flash a RAM paměti, ale musí stále být plně funkční a mít funkce pro správné fungování a chování systému. Většina takovýchto zařízení je napájena z baterie a je nutné, aby vydržela fungovat dlouho bez výměny baterií. Některé systémy v tomto okolí potřebují specifické funkce v důsledku velkého množství typů zařízení v internetu věcí, přesněji v různých monitorovacích stanicích, chytrých městech, alarmech a podobných systémech. Využití v internetu věcí je obrovské, proto je nutné pro tyto systémy najít ten správný operační systém, jenž zabírá málo místa a je funkční na výkonnostně omezených zařízeních. Instalace komplexnějších operačních systémů není ovšem často nutné a lze se bez nich obejít, i přesto bude tato kapitola věnována některým z nich z důvodu možnosti nalezení perfektního operačního systému
pro implementaci v praktické části této práce.

2.1 RIOT

RIOT [4] je open-source malý operační systém pro paměťově omezené systémy s zaměřením na málo výkonné bezdrátové zařízení v Internetu věcí. Je vydán pod licencí LGPL-v2.1 (GNU Lesser General Public License).Je založen na mikrokernelové architektuře. Má plné multithreading a real-time schopnosti. Obsahuje jak zásobníky pro bezdrátovou, tak i drátovou komunikaci po síti. RIOT obsahuje kryptografické knihovny, datové struktury jako hashovací tabulky a prioritní fronty. Také podporuje velké množství mikrokontrolerových architektur, senzorů, rádiových ovladačů
a konfiguraci pro celé platformy jako jsou: Atmel SAM R21 Xplained Pro, Zolertia Z1, STM32 Discovery Boards. RIOT podporuje programovací jazyky C, C++ a Rust.

RIOT běží na 8 bitových procesorech jako AVR Atmega, 16 bitových (TI MSP430) a 32 b (ARM Cortex). Nativní port dovoluje RIOTu provádět jak Linuxové, tak i macOS procesy, což dovoluje systému přístup k developing a debugging nástrojům (GNU Compiler Collection, GNU Debugger, Valgrid, Wireshark) RIOT je částečně kompatibilní s rozhraním POSIX (Portable Operating System Interface).

RIOT poskytuje soubor blokových šifer a kryptografických hashovacích algoritmů. Přesněji poskytuje blokové šifry AES-128 až AES-256. Dále RIOT obsahuje: ECB, CBC, CCM, CTR a OCB módy pro AES, SHA-1, SHA-2, SHA-224, SHA-256,

ChaCha a Poly1305. Kryptografická primitiva, jenž RIOT obsahuje se dají jednoduše použít po přidání na USERMODULE-List. Tyto šifry se dají implementovat přímo, ale doporučuje se je používat přes rozhraní API. Pro dekryptaci se v závislosti na zvolených šifrách používá dostatečně velká vyrovnávací paměť cipher_context_t. Některé aspekty implementace AES lze přizpůsobit pomocí pseudomodulů: crypto_aes_precalculated: Použít před počítané T-tabulky. Tím se zvýšila rychlost na úkor zvětšení velikosti programu. Ve výchozím nastavení se většina tabulek počítá za běhu. crypto_aes_unroll: Povolí ručně odrolované smyčky. Výchozí nastavení je nemít je odrolované.

RIOT obsahuje podporu pro kryptografické knihovny RelicToolkit a HACL*, které lze jednoduše použít po přidání do listu obsažených knihoven a více informací o těchto knihovnách se nachází v kapitole níže o kryptografických knihovnách. Dále obsahuje malou knihovnu Micro-ECC, jenž implementuje ECDH a ECDSA pro 32-bitové mikrokontrolery a má licenci BSD 2-clause. Operační systém také podporuje knihovnu TinyCrypt.

2.2 TinyOS

TinyOS [9] je open source operační systém pro bezdrátová zařízení s nízkou spotřebou energie. Architektura tohoto operačního systému je založena na komponentách. Používá se hlavně v bezdrátových senzorech, ale také v osobních sítích, inteligentních měřičích a podobných zařízeních. Operační systém je napsán v programovacím jazyku nesC, což je variantou jazyku C. TinyOS běží na 8 bitových procesorech
a má omezenou paměť na 512 bytů a také omezenou velikost paketů na 30 bajtů.

NesC implementuje blokové šifry AES. V rozhraní TinyOS jsou implementovány dvě rodiny univerzálních hashovacích funkcí: MMH a Poly32. Systém obsahuje dva kryptografické systémy pro kryptografii s veřejným klíčm. Prvním je kryptografická knihovna TinyECC (Elliptic Curve Cryptography for Sensor Networks) a druhou je TinyPBC (Pairing-based Cryptography in Sensor Networks).

V TinyOS se zprávy posílají prostřednictvím komponenty AMStandard. AMSStandart a GenericComn komponenty jsou vždy použity při posílání a odesílání zpráv v jakékoliv aplikaci v TinyOS. Šifrování může být ustanoveno jako modul v samostatném souboru. Poté, co je balíček zpráv připraven v AMStandard komponentě, je přenesen bezdrátově k přijímači prostřednictvím rádia. Uzel, který přijímá zprávu může stále vidět obsah balíčku zpráv v komponentě AMStandard. To znamená, že pokud chceme používat různé šifrovací algoritmy, musíme je přidat do tohoto oddílu, takto přidaný kód je jak modulární, tak i snadno použitelný.

Aplikace TOSSIM je navržena pro simulaci síťových aplikací pro senzory. TinyViz je grafický vizualizační nástroj používán v kombinaci s aplikací TOSSIM. Algoritmy jako AES, Skipjack nebo XXTEA jsou napsány v jazyce C. Zde přichází program TOSSIM, kde po překontrolování algoritmů je kód přeložen do jazyka NesC. Program také provádí výpočet MAC adresy a simuluje výsledky algoritmů. Simulované výsledky jsou obdrženy v uzlech. Při komunikaci se nejdříve vypočítá MAC adresa a zašifruje se zpráva, pak se zpráva pošle / obdrží, následně se odšifruje heslo. Nakonec se opět vypočítá MAC adresa a porovná se s původní adresou, pokud jsou totožné, ukáže se přijatá zpráva.

2.3 Contiki

Jako předešlé operační systémy je Contiki systém pro paměťově a energeticky omezená zařízení, jenž je zaměřen na bezdrátová zařízeni v internetu věcí. Contiki je open-source software vydán pod licencí BSD-3-Clause. Systém je v praxi používán pro systémy pouličního osvětlení, alarmy, monitorování zvuku pro chytrá města nebo například monitorování radiace. Podobně jako předešlé operační systémy
má Contiki paměť v řádu kilobajtů, spotřebu v řádu miliwattů a běží převážně na 8 bitových systémech. Mnoho systémů Contiki je silně omezeno v napájení. Bezdrátové snímače napájené bateriemi mohou potřebovat zajistit roky bezobslužného provozu a s malými prostředky pro dobíjení nebo výměnu baterií. Výchozí mechanismus
pro dosažení nízkoenergetického provozu rádia se nazývá ContikiMAC. Použitím ContikiMAC mohou uzly pracovat v režimu nízké spotřeby a přitom být schopny přijímat a předávat rádiové zprávy.

Pro efektivní běh na systémech s malou pamětí je programovací model Contiki založen na proto-vláknech. Proto-vlákna fungují jako odlehčená vlákna
bez zásobníků, jenž využívá minimální paměť na individuální proto-vlákno v řádu pár bajtů. Proto-vlákno obsahuje vlastnosti vícevláknového i událostmi řízeného programování a díky tomu je dosaženo nízké paměťové režie. Kernel proto-vlákno procesu vyvolává jako odpověď jak na vnitřní, tak i vnější události. Jakožto vnitřní událost si lze představit časovače, nebo zprávy odeslané jinými procesy. Vnější procesy jsou pak senzory nebo příchozí pakety jiného zařízení.

Co se týče komunikace v sítích má Contiki implementovány tři základní mechanismy. První z nich je uIP TCP/IP zásobník s IPv4 a uIPv6 zásobníkem. Software uIP je open-source implementací TCP/IP protokolu pro 8 a 16 bitové
mikrokontrolery. UIP je pro tyto systémy užitečné po stránce velmi malého množství kódu a využití RAM. Zásobník IPv6 obsahuje také směrovací protokol Routing

Protocol for Low power and Lossy Networks (RPL) pro ztrátové sítě IPv6 s nízkou spotřebou energie a vrstvu pro kompresi a adaptaci záhlaví 6LoWPAN pro spoje IEEE 802.15.4. Dále poskytuje Rime zásobník, což je sada odlehčených síťových protokolů pro bezdrátové sítě s nízkou spotřebou energie. Rime je použit v případech, kdy je režie zásobníků IPv4 nebo IPv6 příliš vysoká. Výchozími primitivy, jenž Rime obsahuje jsou single-hop unicast, single-hop broadcast, multi-hop unicast, network flooding a sběr dat bez adres. Primitiva lze používat samostatně nebo je kombinovat do složitějších protokolů a mechanismů.

2.4 FreeRTOS

FreeRTOS [8] je open source operační systém používaný ve vestavěných systémech. Je napsán převážně v programovacím jazyku C a je vydán pod licencí MIT.
Je sestaven pro mnoho architektur procesorů a je to systém pracující v reálném čase. Poskytuje funkce jako semafory, fronty a časovače. Má škálovatelnou velikost kernelu s použitelnou pamětí programu 9 KB. Některé architektury obsahují režim úspory bez tikání. FreeRTOS umí multithreading tak, že v pravidelných intervalech volá metodu thread tick, ta přepíná úlohy v závislosti na prioritě a schématu plánování round-robin. Interval bývá mezi 1 až 10 milisekundami prostřednictvím přerušení z hardwarového časovače. FreeRTOS je aktuální a taktéž stále vychází nové verze.

FreeRTOS obsahuje důležité knihovny pro připojení jako FreeRTOS-Plus-TCP a coreMQTT.Tyto knihovny zabezpečují připojení pro zařízení v internetu věcí k cloudu. FreeRTOS-Plus-TCP je škálovatelný, open source a bezpečný TCP/IP zásobník. Obsahuje jednoduché rozhraní Berkley sockets, podporu pro IPv4 / IPv6 a protokoly ARP, DHCP, DNS, LLMNR, NBNS, RA, ND, ICMP a ICMPv6. Systém má dva certifikáty pro důkaz zabezpečení. První z nich je SESIP certifikát druhé úrovně, což znamená, že systém obsahuje střední úroveň zabezpečení. Druhý z nich je PSA certifikát první úrovně.

Dále FreeRTOS obsahuje API pro kryptografické prvky a kryptografickou knihovnu MbedTLS, ale kryptografickou knihovnu lze lehce nahradit v případě potřeby. Knihovna MbedTLS a její obsah je více rozebrán v kapitole o kryptografických knihovnách.

2.5 Zephyr

Zephyr [6] je malý operační systém fungující v reálném čase pro zdrojově omezená a vestavěná zařízení jako mikrokontrolery. Software je open source a je vydán pod licencí Apache Licence 2.0. Je napsán v jazyce C. Zephyr používá konfigurační

systémy Kconfig a devicetree, zděděné z linuxového jádra, ale implementované v programovacím jazyce Python pro přenositelnost na jiné než unixové operační systémy. Systém umožňuje sestavování aplikací pro Zephyr v Linuxu, MacOS a Microsoft Windows díky sestavovacímu systému, jenž je založen na CMake. Tento OS má od novějších verzí monolitické jádro, což znamená, že celý operační systém pracuje v prostoru jádra. Monolitický model se liší od jiných architektur operačních systémů, jako
je architektura mikrojádra tím, že jako jediný definuje virtuální rozhraní vysoké úrovně nad hardwarem počítače. Ovladače zařízení lze do jádra přidat jako moduly. Zephyr zabírá 8 KB flash paměti a 5 KB paměti RAM. Je kompatibilní s mnoha zařízeními. Obsahuje protokoly IPv4 a IPv6. Zephyr je aktuální a stále vychází aktualizace.

Zephyr podporuje kryptografickou knihovnu TinyCrypt, jenž je blíže vysvětlena v kapitole o kryptografických knihovnách. Také obsahuje generické kryptografické API a generátory náhodných čísel. Operační systém obsahuje API pro generování náhodných čísel pro kryptografické i ne-kryptografické účely. Ne-kryptografické generátory vrátí náhodné číslo mnohem rychleji, než ty kryptografické. Kryptograficky bezpečné náhodné funkce musí být v souladu s doporučenými algoritmy FIPS 140-2. Na platformách s příslušnou hardwarovou podporou lze použít hardwarové generátory náhodných čísel. Na platformách bez hardwarové podpory pro hardwarové generátory se použije algoritmus CTR-DRBG. Algoritmus může být poskytnut pomocí TinyCrypt nebo mbedTLS v závislosti na požadavcích aplikace na výkon a zdroje.

2.6 Nano-RK

Nano-RK [11] je bezdrátový operační systém pro senzory pracující v reálném čase sestavený pro fungování na mikrokontrolerech v senzorových sítích. Podporuje běh sad úloh v reálném čase díky preemptivnímu časovači s pevnou prioritou. Jako ostatní operační systémy je Nano-RK malý operační systém, přesněji zabírá 2 KB RAM a 18 KB paměti flash. Je to open source software, jenž je napsán v jazyku C a systém běží na senzorech FireFly založených na platformě Atmel a MSP430 procesorech. OS obsahuje jádro prostředků (resource kernel), jenž zajišťuje rezervace, jak často lze systémové prostředky používat. To znamená, že úloha může probíhat jen určitý zlomek času nějakého časového intervalu, nebo uzel může přenášet jen určitý počet paketů za minutu. Tyto rezervace tvoří virtuální energetický rozpočet, který zajišťuje, aby uzel dodržel navrženou životnost baterie a zabraňuje tomu, aby selhávající uzel generoval nadměrný síťový provoz. Operační systém není ovšem moc aktuální a poslední aktualizaci dostal v roce 2008.

2.7 Porovnání operačních systémů

Tab. 2.1: porovnání operačních systému

OS	licence	~ min RAM	~ min ROM
RIOT	LGPL-v2.1	1,8 kB	5 kB
TinyOS	BSD 2-Clause	1 kB	4 kB
Contiki	BSD-3-Clause	$<$ 2 kB	$<$ 30 kB
FreeRTOS	MIT	4-9 kB	5-10 kB
Zephyr	Apache Licence 2.0	5 kB	8 kB
Nano-RK	GPL	2 kB	18 kB

Dále lze operační systémy porovnávat v ohledu na to, která zařízení podporují. RIOT je vytvořen pro 8, 16 a 32 bitové procesory. TinyOS a Contiki běží převážně na 8 bitových procesorech. FreeRTOS a Zephyr podporuje mnoho zařízení a všechny podporované si lze lehce dohledat. Nano-RK běží na senzorech FireFly založených na platformě Atmel a MSP430 procesorech. Tento operační systém také oproti ostatním vyjmenovaným již dlouho neobdržel žádné aktualizace a to přesněji 15 let. Každý operační systém podporuje určité platformy a vždy si je lze dohledat na stránkách daného operačního systému.

RIOT má plnou podporu pro jazyk C a C++. Je to modulární operační systém pracující plně v reálném čase. TinyOS nemá podporu pro C a C++, podporuje jen částečně Multi-Threading, není modulární a nepracuje v reálném čase. Contiki má částečnou podporu jazyka C, podporuje Multi- Threading, je částečně modulární a do jisté míry pracuje v reálném čase. FreeRTOS pracuje v reálném čase, obsahuje Milti-Threading, podporuje jazyk C++ a zvládne i jazyk C. Nano-RK také pracuje v reálném čase.

Každý operační systém má svoje silné i slabé stránky a každý z nich je založen pro jiné účely. Záleží na zdrojích, jenž zařízení má a pro která zařízení je operační systém sestaven. Důležité jsou i možnosti připojení zařízení a jak dokáže komunikovat s okolím. Operační systém jako je RIOT má výhodu že je vysoce modulární // a mnoho prvků si lze lehce přidat, jako jsou například některé kryptografické knihovny. Do ostatních operačních systému si tedy většinou musíme třeba již dříve zmíněné kryptografické knihovny přidat sami.

3 Kryptografické knihovny v internetu věcí

V následující kapitole bylo citováno z: [7] Kryptografické knihovny jsou nějakým souborem kryptografických nástrojů, přes které je umožněno shromažďovat dané funkce do specifických kolekcí na základě jejich výkonu, kapacity a funkcí.
Jak již bylo dříve uvedeno, v internetu věcí je potřeba specifických knihoven v důsledku jak malé paměti, tak malé výkonosti. Pro zabezpečení komunikace jsou algoritmy těchto knihoven nutné a proto budou podsekce níže věnovány určitým kryptografickým knihovnám a algoritmům, jenž obsahují. V praktické části práce budou pak některé z těchto knihoven testovány, porovnány a následně bude vybrána nejlepší knihovna pro implementaci algoritmů pro zabezpečení komunikace s určitým omezeným zařízením.

3.1 OpenSSL

Kryptografická knihovna OpenSSL [19] je napsána v programovacím jazyce C a jedná se o milti platformovou kryptografickou knihovnu s různými algoritmy a funkcemi. Je to open source knihovna pod licencí Apache license a BSD licencí.
Taktéž se jedná o nejvíce používanou kryptografickou knihovnu, je ovšem příliš velkou pro zařízení v internetu věcí. Obsah knihovny OpenSSL je v následujících třech tabulkách.

Tab. 3.1: Šifry obsažené v OpenSSL

AES	Blowfish	Camellia	Chacha20	Poly1305
DES	IDEA	RC2	RC4	RC5
3DES	GOST 28147-89	SEED	SM4	CAST-128

Tab. 3.2: Hashovací funkce obsažené v OpenSSL

MD5	MD4	MD2	SHA-1
SHA-3	RIPEMD-160	MDC-2	GOST R 34.11-94
BLAKE2	Whirlpool	SM3	SHA-2

Tab. 3.3: Kryptografie s veřejným klíčem obsažená v OpenSSL

RSA	DSA	Diffie-Hellman key exchange	Eliptic curve	X25519
Ed25519	Ed448	X448	GOST R 34.10-2001	SM2

3.2 WolfSSL

Je napsána v jazyce C. WolfSSL [13] je open source a je licencována pod licencí GNU General Public License GPLv2. Tato knihovna je ideální pro zařízení v internetu věcí, díky její malé velikosti a nízkému využití paměti. Spotřeba paměti této knihovny se pohybuje mezi 1-36 kB. WolfSSL je zaměřena k použití ve vestavěných zařízeních, operačních systémech v reálném čase a prostředích s nedostatkem dat a omezenými výpočetními zdroji. WolfSSL podporuje zlib kompresi, IPv4 a IPv6 spolu s integrací MySQL. Podporuje vývoj multiplatformních algoritmů a umí generovat klíče a certifikáty. Obsahuje SSL/TLS algoritmy, podporuje několik API, má kompatibilní rozhraní s nejvíce používanými funkcemi knihovny OpenSSL,
a také má podporu OSCP a CRL, jenž se používají pro kontrolu certifikátů. Předchůdce WolfSSL s názvem yaSSL je napsán v C++ a je to taktéž knihovna založená na knihovně SSL pro vestavěná prostředí a operační systémy reálného času s omezenými zdroji. WolfSSL je kompatibilní s předešle zmíněnými operačními systémy TinyOS
a RIOT, samozřejmě je kompatibilní s mnoha dalšími OS, nevyjímaje Microsoft Windows, Linux a MacOS. WolfSSL se skládá ze dvou menších knihoven: WolfCrypt a NTRU.

Tab. 3.4: základní obsah knihovny WolfCrypt

RSA	ECC	Diffie-Hellman	DSS
NTRU	3DES	AES(CBC, CTR, CCM, GCM)	cAMELLIA
ARC4	HC-128	ChaCha20	MD2 až MD5
BLAKE2	RIPEMD-160	Poly1305	EDH
IDEA	SHA-1 až SHA-3		

Dále knihovna WolfCrypt obsahuje generátor náhodných čísel, podporu velkých intigerů, základní 16/64 kódování a dekódování. Dále například obsahuje experimentální šifru Rabbit a podporuje novější algoritmy Curve25519 a Ed25519. WolfCrypt také funguje jako back-end kryptografická implementace několika populárních softwarových balíčků a knihoven, jako je MIT Kerberos, ten lze povolit ve vestavěném nastavení. **NTRU** je druhá pod-knihovna v WolfSSL. Tato knihovna zajišťuje stejnou bezpečnost, jako jiné systémy veřejných klíčů i přes to, že má menší bitovou velikost. NTRU obsahuje AES-256, RC4 a HC-128.

3.3 AvrCryptoLib

Knihovna AvrCryptoLib [20] má implementaci svých algoritmů v 8 bitových mikrokontrolerech AVR, což vede ke zkrácení doby provádění procesů této knihovny. Stejně jako knihovna WolfSSL je přizpůsobena pro oblast internetu věcí,
kde se pracuje s omezenými zdroji. Tato knihovna umožňuje přímý přístup ke klíčům prostřednictvím uložení těchto klíčů ve flash paměti, což vede k efektivnímu využití paměti RAM. Algoritmy AvrCryptoLib knihovny jsou v tabulce 3.5:

Tab. 3.5: obsah knihovny AvrCryptoLib

Typ	**algoritmy**		
proudové šifry	ARC4	Trivium	Mugi
	Mickey	Grain	
Blokové šifry	AES	XTEA	CAST 5
	Threefish-256	Threefish-512	Threefish-1024
	SERPENT	SHABEA	Present
	Noekeon	RC5	RC6
	3DES	EDE-DES	Camellia
	SEED	SKIPJACK	DES
Hashovací funkce	BLAKE	Twister	Shabal
	SHA-1	SHA-256	MD5
	BlueMidnightWish	Skein	Grost 1

3.4 TinyECC

TinyECC je kryptografická knihovna založena na eliptických křivkách. Tato knihovna byla již zmíněna u operačního systému TinyOS, kde je také její hlavní využití,
ale dá se také použít na jiných operačních systémech. Tato knihovna implementuje křivky pouze nad prvočíselnými poli, posuvná okna, Barettovou redukci pro účely ověření, ale mezi její hlavní funkce patří:

1. poskytování digitálního podpisu ECDSA
2. ECDH protokol pro výměnu klíčů
3. ECIES schéma pro šifrování veřejného klíče

3.5 RelicToolKit

Knihovna RelicToolKit [15] má hlavní využití v tom, že si v ní lze sestavit vlastní kryptografické nástroje. RelicToolKit poskytuje vysokou úroveň přizpůsobení, pokud

jde o sestavení a začlenění pouze požadovaných komponent pro použití na požadovaných platformách, nebo o Požadovaný výběr různých matematických optimalizací pro optimální výkon sady nástrojů ve konkrétní platformě. Je licencována LGPL v2.1. Tato knihovna obsahuje aritmetiku celých čísel, bilineární mapy a rozšíření, jenž se vztahují k bilineárním mapám. Také má vestavěné funkce s eliptickými křivkami nad prvočíselnými poli, binárními poli, aritmetikou prvočísel a binárními poli. Další kryptografické algoritmy knihovny RelicToolKit se nacházejí v tabulce 3.6 níže.

Tab. 3.6: kryptografické protokoly knihovny RelicToolKit

ECDSA	RSA
ECIES	ECSS
Rabin	ECMQV

Sakai-ohgishi-Kasahara ID-based authenticated key agreement

Boneh-Lynn-Schacham short signiture

Paillier and Benaloh homomorphic encryptation systems

Boneh-Boyen short signiture

3.6 HACL*

Hacl* [5, 14] (High Assurance Cryptographic Library) je kryptografická knihovna fungující na bázi jazyka C a je distribuovaná pod licencí Apache-2.0. Je napsána v jazyce F* a poté je kompilovaná do jazyka C. Zdrojový kód každého kryptografického primitiva je testován z hlediska paměťové bezpečnosti, omezení proti postranním časovým kanálům a funkční správnosti. Při kompilaci pomocí GCC na 64bitových platformách jsou primitiva této knihovny stejně rychlé jako nejrychlejší implementace v OpenSSL. Tato knihovna také obsahuje API pro práci v kryptografickém prostředí NaCl. HACL* se často požívá jako náhrada knihoven NaCl, jako jsou libsodium a TweetNaCl. Primitiva z HACL* jsou také integrována do kryptografické knihovny NSS společnosti Mozilla.

Tab. 3.7: kryptografická primitiva knihovny HACL*

ChaCha20	Salsa20	SHA-1 až sha-3
Poly1305	HMAC	Blake2
SHA-256	SHA-512	P-256
Curve25519	Ed25519	

3.7 TinyCrypt

Knihovna TinyCrypt [18] poskytuje implementaci pro zařízení
s omezenými prostředky s minimální sadou standardních kryptografických primitiv. Minimalizuje velikost kódu každého kryptografického primitiva, jenž obsahuje. Pomocí knihovny lze implementovat pouze ta kryptografická primitiva, jenž uživatel zrovna požaduje. Podporuje programovací jazyk C.

Tab. 3.8: kryptografická primitiva knihovny TinyCrypt

SHA256	HMAC-SHA256	HMAC-PRNG
AES-128	Módy pro AES: CBC, CTR, CMAC, CCM	
ECC-DSA	CTR-PRNG	ECC-DH
ECDH		

3.8 MbedTLS

Tato kryptografická knihovna je implementace protokolů TLS a SSL a je šířena pod licencí Apache License 2.0. MbedTLS [16, 17] je napsaná v programovacím jazyce C. Knihovna je pro malé a zdrojově omezené zařízení, přičemž minimální kompletní zásobník TLS vyžaduje méně než 60 KB programového prostoru a méně než 64 KB paměti RAM. Knihovna je vysoce modulární, což znamená, že její prvky se dají implementovat samostatně dle potřeby. Knihovna podporuje platformy jako jsou ARM, x86, PowerPC, MIPS. Funkce kryptografické knihovny jsou v tabulce 4.9 níže. Mbed TLS také obsahuje dva módy pro šifrování MAC CMAC a HMAC. Dále obsahuje algoritmus pro odvození klíče HKDF a několik funkcí pro natahování klíče: PBKDF2, PKCS, PKCS #12.

3.9 TinyPairing

V posledních letech se začaly provádět mnohé studie týkající se kryptografie s výpočtem bilineárního párování v senzorových systémech. Knihovna TinyPairing [12] je jako jedna z prvních založena na párování pro bezdrátové senzorové sítě. Knihovna je rychlá, odlehčená a také zabírá malé množství RAM a ROM, což je důvod, proč se bilionarní párování aktivně nepoužívá v malých systémech. Pro docílení malé náročnosti na systém je knihovna odlehčena o některé složitější funkce, týkající se párování. Funkce, jenž TinyPairing obsahuje, jsou následující: T pairing, skupinová operace na eliptických křivkách, aritmetika nad základním polem a rozšiřujícím polem, generování náhodných čísel, hashovací funkce a aritmetické operace s více

Tab. 3.9: Obsah knihovny Mbed TLS

Typ	**Algoritmy**		
Hashovací funkce	MD2	MD4	MD5
	SHA-1	SHA-2	SHA-3
	RIPEMD160		
Šifry	AES	ARIA	Blowfish
	ChaCha	DES	RC4
	XTEA	Camellia	3DES
Šifrové Módy	ECB	CBC	CFB
	OFB	XTS	CTR
Módy pro autentizační	CCM	GCM	NIST Key Wrap
šifrování kryptografii	RSA	Diffie–Hellman	ECC
s veřejným klíčem	ChaCha20-Poly1305		
	ECDH	ECDSA	J-PAKE

přesnostmi. Také má jedno schéma pro šifrování schématu BF identity-based encryption a dvě krátká podpisová schémata: BLS short signiture a short signiture by Boneh and Boyen.

4 Testování kryptografických knihoven

V této kapitole bude řešeno praktické porovnávání kryptografických knihoven. Před použitím konkrétní kryptografické knihovny v praktickém řešení je nutné porovnat různé knihovny a vybrat tu nejvhodnější pro implementaci do systému.

4.1 Měření na zařízení Raspberry Pi

Měření v této podkapitole probíhalo na zařízení Raspberry Pi Zero, protože každá knihovna je optimalizována pro různá prostředí a typy zařízení. Většinu knihoven je jednoduše možné použít na systémech, jenž běží na nějakém typu operačního systému Linux. Raspberry Pi Zero je malý a levný počítač, který slouží k výuce programování, k vytváření různých elektronických projektů a k automatizaci domácnosti. Může být použit jako základní počítač, multimediální centrum, server nebo pro projekty, jenž operují v prostředí internetu věcí.

4.1.1 Velikost knihoven

Prvně je zde porovnání z hlediska místa, jenž knihovny zabírají na disku. Velikosti souborů knihoven jsou na obrázku 4.1. Na první pohled je možné vidět, že se velikostmi knihovny výrazně liší. Největší knihovnou je samozřejmě OpenSSL,
což je díky tomu, že knihovna není dělaná pro prostředí internetu věcí. Pro OpenSSL jsou zde započítány pouze hlavní pod-knihovny, jenž obsahují kryptografické funkce. Přesněji se jedná o dvě pod-knihovny libssl a libcrypto, jenž se navzájem potřebují. Pokud by zde byl započítán hlavní soubor knihovny OpenSSL, hodnota by se zvětšila o asi 500 kB.

MbedTLS je v grafu uvedena dvakrát. Jeden soubor byl zkompilován se všemi funkcemi knihovny a je tedy výrazně větší, než druhá kompilace, v níž jsou zahrnuty pouze více populární algoritmy. Před kompilací knihovny si tedy lze vybrat funkce, které chceme, aby byly v knihovně zahrnuty.

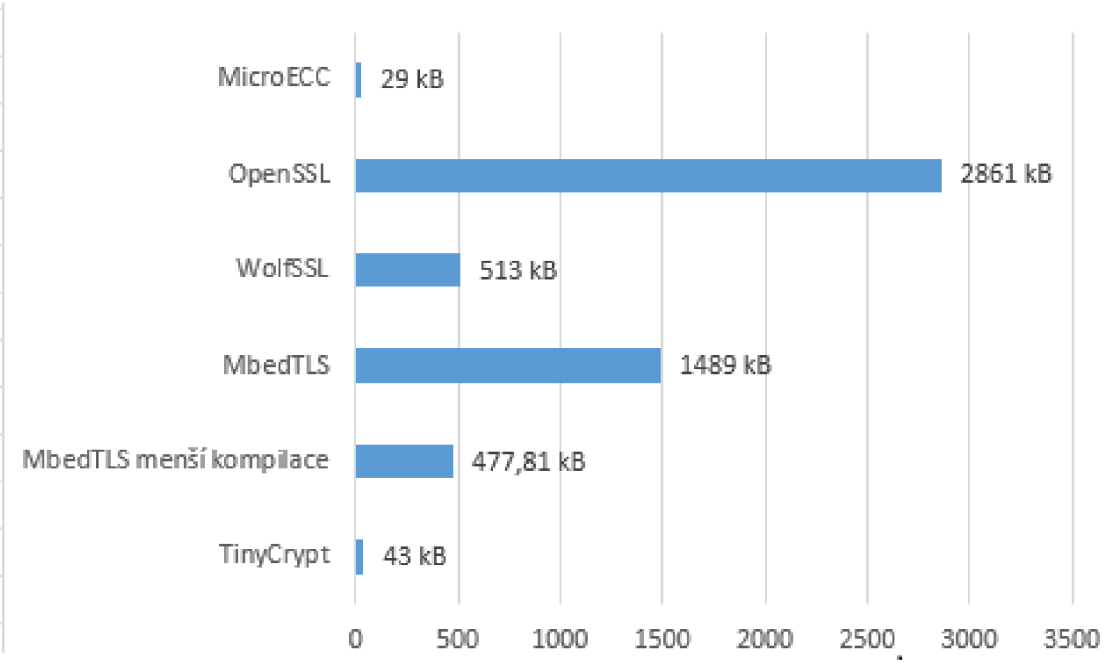
Podobně jako u MbedTLS je to s knihovnou WolfSSL, u které si lze při kompilaci vybrat balíčky funkcí, jenž chceme aby byly v sestavené knihovně zahrnuty. V základní velikosti je knihovna ovšem o něco menší, než MbedTLS.

Dále se zde nachází knihovny MicroECC a TinyCrypt. Ty mají výrazně menší velikost, než ostatní knihovny. To je v důsledku toho, že knihovny poskytují pouze pár funkcí v minimalistickém provedení. Jsou tedy udělány pro využití v specifických případech. MicroECC je knihovna pro eliptické křivky. Obsahuje ECDSA, ECDH

a modulární funkce na eliptických křivkách. TinyCrypt obsahuje pouze pár algoritmů jako je AES, SHA-256, HMAC, ECDSA a ECDH. AES obsahuje pouze s 128 bitovými klíči, takže se jedná o velice odlehčenou knihovnu.

Ovšem zde samozřejmě platí, že každá knihovna má své využití v jiných případech. Záleží na tom, které funkce je potřeba a na parametrech systému. Knihovny jako jsou MbedTLS a WolfSSL je stále možné výrazně zmenšit při kompilaci knihovny, ale i tehdy budou větší, než MicroECC a TinyCrypt.

Obr. 4.1: Velikosti knihoven v kB



4.1.2 Velikost souborů knihoven

V teto sekci budou porovnány velikosti určitých částí knihoven. V internetu věcí je velice důležitá velikost z důvodu omezených zdrojů a většinou je možné přidat pouze určité funkce knihoven. Na grafu nacházejícím se na obrázku 4.2 se nachází vždy záznam pro zdrojový .c soubor a hlavičkový .h soubor určitých funkcí knihoven. Některé knihovny jsou více modulární než jiné. WolfSSL a MbedTLS budou potřebovat více hlavičkových a zdrojových souborů k implementaci určité funkce do programu, ale pro zjednodušení se zde nachází porovnání pouze těch souborů, jenž přímo implementují funkce. Toto se tedy týká pouze dříve zmíněných dvou knihoven MbedTLS a WolfSSL. Zde by tedy velikosti mohly být o něco větší.

V grafu je viditelné, že knihovny potřebují pro funkce ECDSA a ECDH navíc soubory obsahující funkce pro eliptické křivky, které jsou v grafu uvedeny jako ECC.

WolfSSL má oproti ostatním knihovnám výrazně větší zdrojové soubory, to je v důsledku implementace více funkcí do jednoho souboru. Zdrojové soubory pro AES této knihovny obsahují více módů tohoto algoritmu(ECB, CBC, CTR, CCM, GCM). Ostatní knihovny obsahují zvlášť soubory pro jednotlivé módy šifry AES, proto jsou jejich zdrojové soubory výrazně menší. Jak již bylo v předešlé podkapitole zmíněno, WolfSSL je méně modulární, než ostatní zde zmíněné knihovny a je možné pouze zvolit určité balíčky funkcí, jenž chceme použít. Toto je hlavním důvodem, proč jsou soubory této knihovny větší.

Knihovna MicroECC obsahuje pouze jeden soubor s eliptickými křivkami, jenž taktéž obsahuje ECDSA a ECDH. Důležité je také zmínit, že TinyCrypt obsahuje pouze AES se 128 bitovou délkou klíčů a to samozřejmě ovlivňuje velikost souboru. Podobně je tu u ECDSA a ECDH, u kterých podporuje pouze 256 bitové délky klíčů s křivkou secp256r1. To tedy velice limituje možnosti využití této knihovny. Ostatní zde zmíněné knihovny kromě MicroECC podporují délky klíčů 192, 224 256, 384 a 521 bitů pro ECDSA a ECDH. MicroECC podporuje délky klíčů 160, 192, 224 a 256 bitů, přičemž minimální doporučená bezpečná délka klíče je 192 bitů.

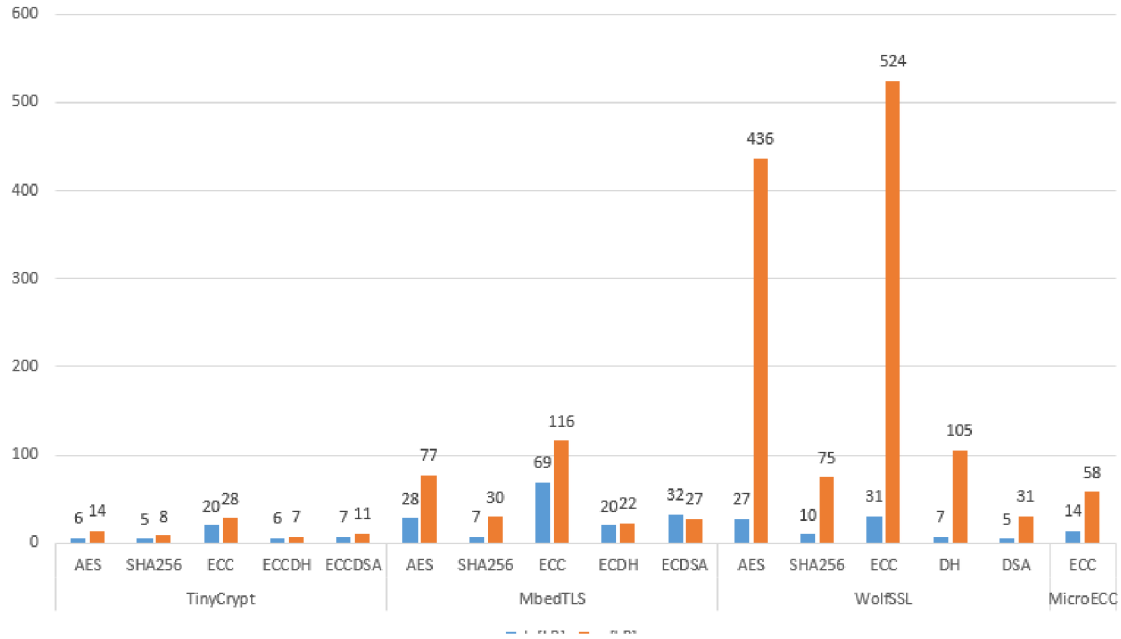
Závěrem tedy je, že pokud je potřeba použít pouze funkce na eliptických křivkách, tak knihovna MicroECC je dobrou volbou pro co nejmenší užití úložného prostoru. TinyCrypt je knihovna, jenž využívá minimální množství úložného prostoru, ale na druhou stranu je limitována na pouze pár algoritmů. Využití a bezpečnost těchto algoritmů je také výrazně omezena. Knihovny MbedTLS a WolfSSL si jsou velice podobné a často mohou být zaměnitelné. MbedTLS je ovšem o něco lepší v tom, že si lze vybrat pouze určité algoritmy a tím výrazně zmenšit velikost knihovny. Oproti tomu knihovna WolfSSL umožňuje vybrat sady funkcí a tím pádem nelze velikost knihovny zmenšit tak drasticky, jako tomu je u MbedTLS.

4.1.3 Doba běhu programu s implementací knihovny

V této sekci je rozebraná tabulka na obrázku 4.3, která obsahuje doby běhu programů s implementovanými algoritmy jednotlivých knihoven. Měření bylo provedeno pro AES, SHA256, ECDH a ECDSA. MicroECC je knihovna implementující pouze funkce týkající se eliptických křivek, a proto neobsahuje AES a SHA256. AES byl použit se 128 bitovou délkou klíče, aby bylo možné sjednotit verzi algoritmu pro všechny knihovny, jelikož TinyCrypt obsahuje AES pouze s touto délkou klíčů. Obdobně je to u ECDSA a ECDH, u nichž byly použity 256 bitové klíče ze stejného důvodu. Pro měření byl použit jednoduchý program napsán v jazyce C, který časy měřil. Časy jsou průměrem 100 opakovaných měření.

Z prvního pohledu je možné vidět, že jsou časy prakticky totožné. To může mít více důvodů. Prvním z nich je zařízení, na němž bylo měření provedeno. Raspberry

Obr. 4.2: Velikosti hlavičkových a zdrojových souborů knihoven



Pi Zero má specifické hardwarové omezení, které může ovlivnit běh kryptografických algoritmů. Jelikož je hardware stejný, časy běhu mohou být podobné i při použití různých knihoven. Taktéž za to částečně může být zodpovědné softwarové prostředí, ve kterém byly testy uskutečněny, přesněji operační systém nebo kompilátor. Nainstalovaným operačním systémem během měření byl Raspberry Pi OS, jenž je také známý jako Raspbian. Taktéž může být měření zkreslené v důsledku běhu procesů na pozadí, nebo také díky překryvu v implementacích algoritmů kryptografických knihoven. Knihovny jsou taktéž dobře optimalizované pro prostředí v internetu věcí, což může taktéž naměřené časy přibližovat.

Tab. 4.1: Časová měření pro různé algoritmy a knihovny

algoritmus	**TinyCrypt**	**MbedTLS**	**WolfSSL**	**OpenSSL**	**MicroECC**
aes-128	0,000533	0,000533	0,000557	0,000558	x
sha256	0,000534	0,000540	0,000577	0,000558	x
ecdh	0,000543	0,000535	0,000545	0,000553	0,000559
ecdsa	0,000541	0,000556	0,000550	0,000566	0,000600

4.2 Měření na zařízení PIC Curiosity Dev Board

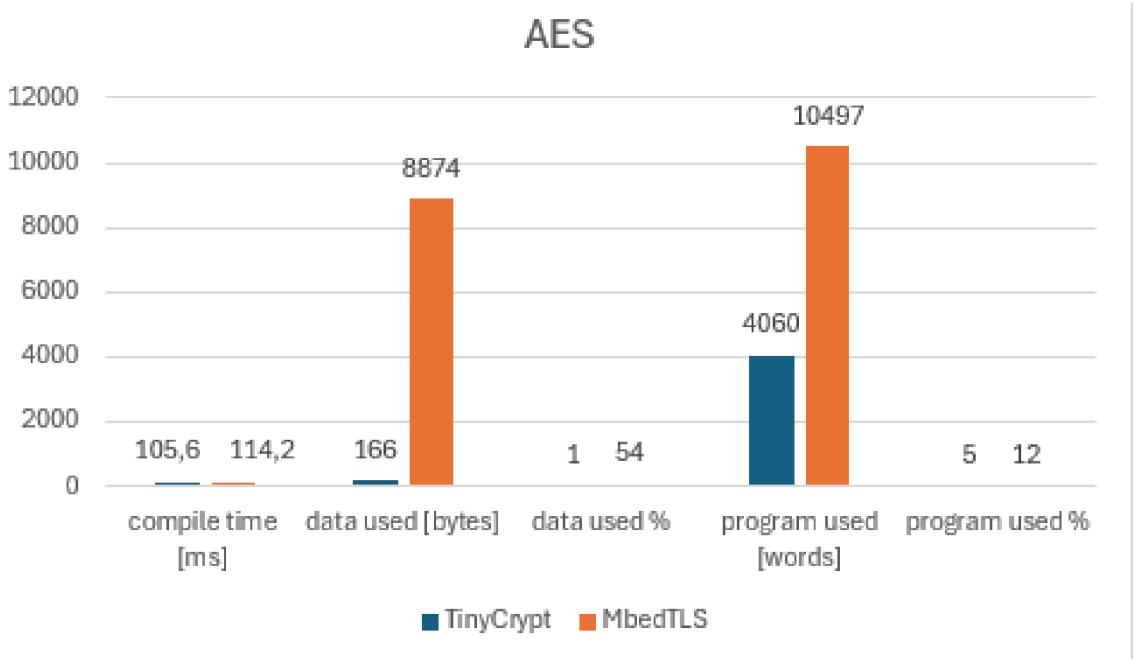
V této sekci budou porovnávány pouze knihovny kompatibilní s 16 bitovými zařízeními. Přesněji se bude jednat o TinyCrypt, MicroECC a MbedTLS.
Měření bylo provedeno na zařízení Curiosity Development Board PIC24F 16BIT MCU. Jedná se o vývojovou sadu s 16 bitovým mikro-procesorem navržen pro testování a vývoj aplikací v omezeném prostředí internetu věcí.

V následujícím textu budou na tomto zařízení testovány určité algoritmu za účelem zjistit jejich výkonnostní parametry. Na obrázku 4.4 lze vidět výsledky pro šifru AES z knihoven TinyCrypt a MbedTLS. V grafu je jako první položka compile time neboli čas délky trvání kompilace programu. Časy jsou průměrem z pouhých deseti měření z důvodu nutnosti čekat poměrně dlouhou dobu mezi testy. To způsobuje prostředí pro práci s tímto zařízením. V tomto se časy velice neliší a takto to je i u ostatních algoritmů v grafech níže. Druhým parametrem je data used což lze přeložit jako využití datové paměti, což se týká úložného prostoru na tomto zařízení v bajtech. Zde je již výrazný rozdíl mezi knihovnami. MbedTLS zabírá 54 % úložného prostoru zařízení. To je zapříčiněno tím, že knihovna není vyvíjena přímo pro takto omezená 16 bitová zařízení. AES této knihovny také obsahuje více délek klíčů, a to přesněji 128, 192 a 256 bitové klíče, zatímco TinyCrypt obsahuje pouze podporu pro 128 bitové klíče. Celkově je AES knihovny MbedTLS obsáhlejší a takto to je i u ostatních algoritmů této knihovny. Samozřejmě toto způsobuje i větší rozdíl v parametru program used. Toto označuje celkovou velikost dostupné programové paměti ve slovech. Ve většině architektur mikrokontrolérů je jedno slovo obvykle 2 bajty.

Na obrázku 4.5 je graf, týkající se hashovací funkce SHA256. Zde jsou rozdíly mezi knihovnami výrazně menší, než tomu bylo u šifry AES. Implementace SHA256 je výrazně jednodušší a menší z hlediska úložného prostoru, ale i přesto využívá MbedTLS více programové paměti, než je tomu u knihovny TinyCrypt.

Dále se v grafu na obrázku 4.6 nachází algoritmus ECDSA. Vše, co je v tomto odstavci napsáno, platí také pro algoritmus ECDH na obrázku 4.7. Jsou zde porovnány knihovny MicroECC a TinyCrypt. Knihovna MbedTLS zde není zařazena z jednoduchého důvodu a to by byla nutnost upravit některé funkce ze zdrojových souborů knihovny kvůli neúplné kompatibilitě s 16 bitovými zařízeními. Také, jak již bylo vidět u předešlých algoritmů, není MbedTLS velice efektivní pro toto zařízení z hlediska velikosti implementace algoritmů. ECDSA těchto dvou knihoven, tedy MicroECC a TinyCrypt, nejsou příliš velikostně rozdílné a náročné. Tyto knihovny jsou velice dobře optimalizované pro tento typ zařízení. MicroECC využívá o něco málo více programové paměti, než TinyCrypt. Ovšem zde je nutné podotknout fakt, že MicroECC podporuje větší množství standardních křivek. Přesněji se jedná o

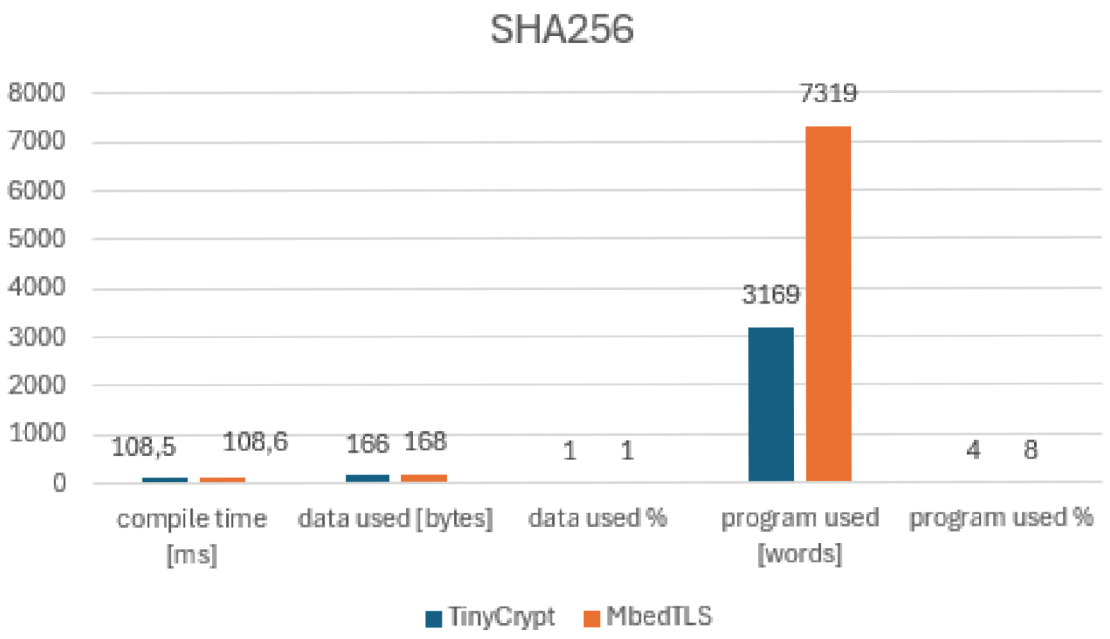
Obr. 4.3: Test šifry AES na vývojové desce



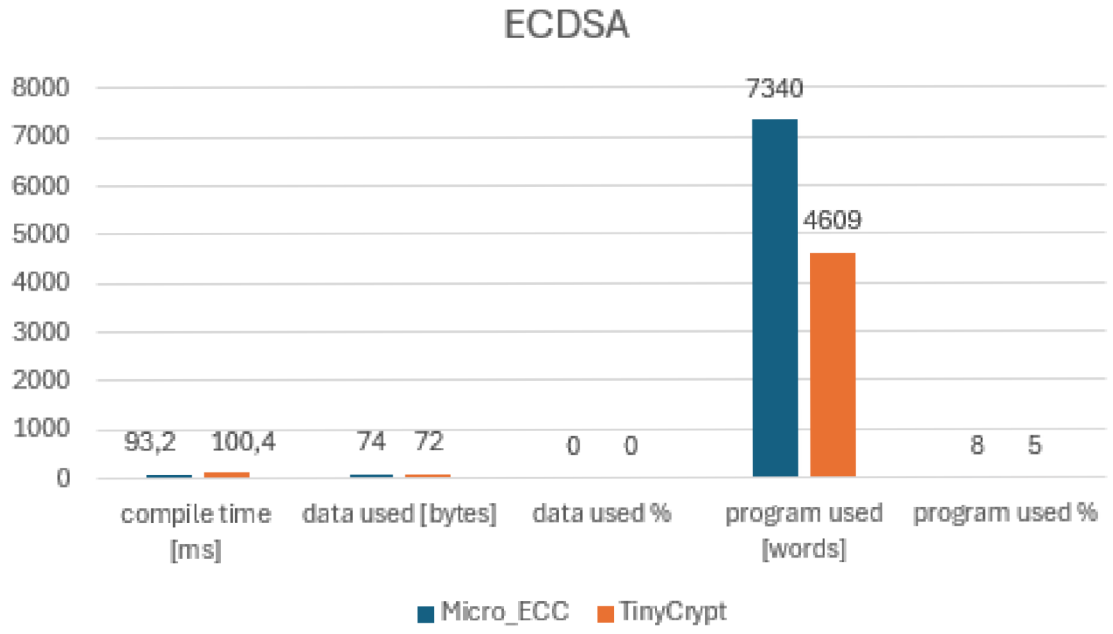
křivky: secp160r1, secp192r1, secp224r1, secp256r1 a secp256k1.
TinyCrypt podporuje pouze křivku secp256r1. Celkově je implementace všech algoritmů knihovny TinyCrypt minimalistická, což velice omezuje využití této knihovny.

Výsledkem tohoto měření je hlavně fakt, že MbedTLS se výrazně liší od TinyCrypt a MicroECC ve velikostech implementací jejích algoritmů. Rozsáhlejší využití více algoritmů z MbedTLS by toto zařízení nezvládlo. Důležité také je,
že i když se výsledky knihoven MicroECC a TinyCrypt velice neliší, tak knihovna TinyCrypt obsahuje výrazně méně funkcí a celkové je podřadná díky minimalistickým implementacím jejích algoritmů. Dále tedy nebude tato knihovna využita.

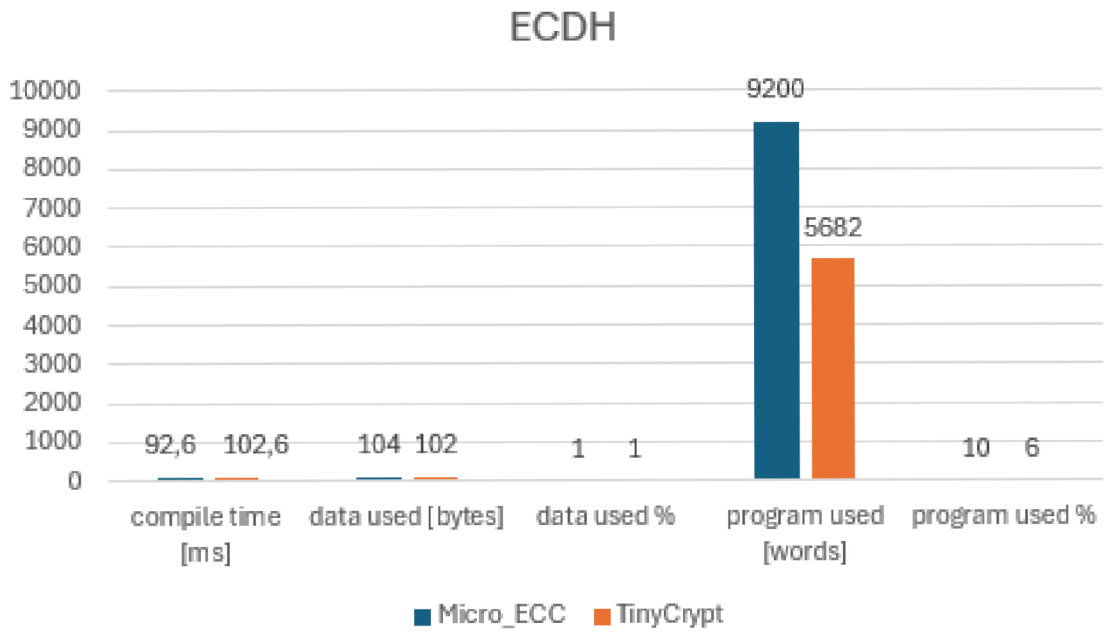
Obr. 4.4: Test hashovací funkce SHA256 na vývojové desce



Obr. 4.5: Test algoritmu ECDSA na vývojové desce



Obr. 4.6: Test algoritmu ECDH na vývojové desce



5 Návrh a implementace systému

V této kapitole budou navrženy a implementovány kryptografické protokoly, jenž využívají předešle zmíněné knihovny. Protokoly byly navrženy a sestaveny na zařízení pracující v oblasti internetu věcí. Přesněji na zařízení použitém v minulé kapitole Curiosity Development Board PIC24F 16BIT MCU. Jelikož cílem práce je minimalizovat systémové zdroje, tak zde nebude implementován žádný operační systém z prostředí internetu věcí. Na zařízení, kde budou protokoly vyvíjeny by to ani nebylo možné.

5.1 Návrh systému

V systému by měla být implementace zabezpečení komunikace zařízení v internetu věcí a hlavního sběrače dat, což bude v tomto případě počítač. V tomto systému bude omezené zařízení simulovat Raspberry Pi Zero z důvodu zjednodušení komunikace, i přesto budou zde vyvinuté kryptografické protokoly kompatibilní a použitelné na omezených zařízeních, jelikož byly vytvořeny na omezeném zařízení Curiosity Development Board PIC24F 16BIT MCU. Zařízení Raspberry Pi Zero bude simulovat zařízení v internetu věcí, které měří teplotu. Tyto data bude dále třeba poslat hlavní stanici (počítači) prostřednictvím zabezpečené komunikace. Bezpečný přenos bude uskutečněn pomocí tří protokolů, mezi kterými se bude možné přepínat. Na počítači se po úspěšné komunikaci zobrazí data sběrače.

V protokolech byly kryptografické algoritmy, jenž se týkají eliptických křivek využívány z knihovny MicroECC, která je dobrou volbou pro úsporu úložného prostoru, který je na 16 bitovém zařízení velice omezený. Přesněji se jedná o modulární funkce na eliptických křivkách nebo případně algoritmy ECDH a ECDSA. Jako hashovací funkce byla využita hashovací funkce SHA256 z knihovny MbedTLS, to je z toho důvodu, že MicroECC neobsahuje hashovací funkce a dále bylo potřeba využít šifru AES v režimu GCM, jenž je obsažen právě v knihovně MbedTLS. SHA256 nebyla čerpána z jiné knihovny, aby nebylo třeba načítat třetí knihovnu. MbedTLS není plně kompatibilní s 16 bitovými zařízeními a proto bylo potřeba na začátku kódu algoritmu specifikovat funkci memcmp. Funkce memcmp je standardní knihovní funkcí v jazyce C, která se používá k porovnání dvou bloků paměti. Je třeba jí definovat, jelikož některé funkce knihovny MbedTLS, jenž poskytují funkcionalitu memcmp, nefungují na 16 bitových zařízeních a je třeba je nahradit.

5.1.1 Protokol 1

Jedná se o symetrický protokol, jenž je navržen pro využití standardizovaných kryptografických algoritmů. Tento protokol zabezpečuje komunikaci mezi Raspberry Pi Zero, dále senzorem a PC, dále kolektorem pomocí šifrování AES-GCM a klíčové derivace SHA-256.

Před komunikací je nutná instalace klíčů. Senzor instaluje klíč KU pro šifrování, který se používá k ochraně dat přenášených ze senzoru. Dále je zde vytvořen jedinečný identifikátor IDU uživatele senzoru. Kolektor provede to samé a instaluje klíč KR a vytváří identifikátor IDR. Také je nutné derivovat klíče skey a ukey pomocí funkce SHA-256 a hlavního klíče KM. Jedná se o šifrovací klíče pro zabezpečení zpráv od senzoru. Kolektor provede to samé a také takto vygeneruje oba klíče skey a ukey pro zabezpečení zpráv od kolektoru. Nyní přichází první krok komunikace která je také vizuálně znázorněná na obrázku 5.1.

1. Senzor odesílá IDU a NU. Nu je nonce, neboli náhodné číslo, které je použito k vytvoření unikátního šifrovacího klíče pro danou komunikaci. Nonce zajišťuje, že každý šifrovaný blok dat je jedinečný, i když jsou použity stejné klíče.
2. Kolektor odesílá zpět IDR, NR, IVR, C1 a TAG1. NR je nonce od kolektoru. IVR je inicializační vektor, který je použit v algoritmu AES-GCM pro šifrování dat. C1 je šifrovaná zpráva a TAG1 je určen pro ověření integrity a autenticity šifrované zprávy. Následuje proces šifrování:

$$C1, TAG1 = ENC_{AES-GCM}(IVR, skey, serverhello)$$

3. Senzor ověřuje a dešifruje zprávu pomocí klíče skey a IVR, integrita je ověřena pomocí TAG1. Proces dešifrovaní:

$$serverhello = DEC_{AES-GCM}(IVR, skey, C1, TAG1)$$

4. Senzor odesílá zpět IVU, C2 a TAG2 kolektoru. IVU je inicializační vektor, který je použit v algoritmu AES-GCM pro šifrování dat. C2 je šifrovaná zpráva. Opět přichází proces šifrování:

$$C2, TAG2 = ENC_{AES-GCM}(IVU, ukey, DATA)$$

5. Kolektor finálně přijímá a dešifruje zprávu:

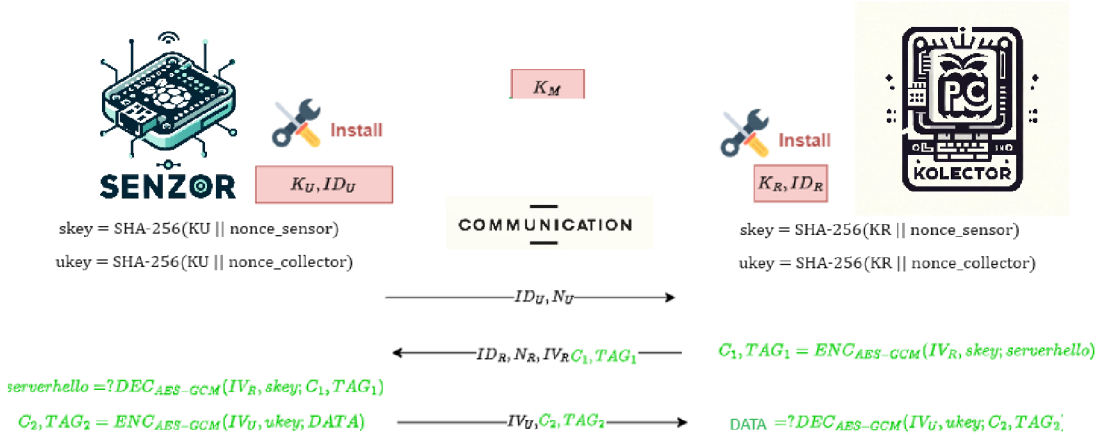
$$DATA = DEC_{AES-GCM}(IVU, ukey, C2, TAG2)$$

Dešifruje samozřejmě pomocí ukey a ověřuje integritu pomocí TAG2.

Tento protokol zajišťuje bezpečnou komunikaci mezi senzorem a kolektorem pomocí klíčů odvozených z hlavního klíče a jedinečných nonce. Každý krok zahrnuje

šifrování a dešifrování dat, což zajišťuje důvěrnost, integritu a autenticitu přenášených zpráv.

Obr. 5.1: Diagram komunikace protokolu 1



5.1.2 Protokol 2

Protokol je navržen pro využití protokolů s nulovou znalostí. Využívá Schnorrův podpis a důkaz znalosti. Schnorrův podpis je digitální podpisový algoritmus založený na problému diskrétního logaritmu v konečných grupách, který je znám svou jednoduchostí a efektivitou. V porovnání s jinými podpisovými schématy, jako je například DSA, nabízí Schnorrův podpis menší velikost podpisu a vyšší výkonnost. Jedná se o proprietární kryptografický návrh, což znamená, že to není to standard. Schnorrův podpis využívá matematické struktury nazývané cyklické grupy.
Základním bezpečnostním předpokladem je, že výpočet diskrétního logaritmu v těchto grupách je výpočetně velmi náročný. Jedná se o asymetrický protokol. Základ protokolu jako zařízení a knihovny je prakticky totožný, jako u protokolu 1, s tím rozdílem,
že se zde navíc používá knihovna MicroECC pro funkce spojené s eliptickými křivkami, přesněji modulární funkce na eliptických křivkách knihovny MicroECC.

Opět se prvně instalují klíče. Pro senzor to jsou: sku - soukromý klíč senzoru, pku - veřejný klíč senzoru a pkS což je známý veřejný klíč kolektoru. Toto provede i kolektor s klíči (sks, pks a pku). Následuje komunikace, jenž je také vizualizována na obrázku 5.2.

1. Senzor odesílá IDU kolektoru, což je identifikátor senzoru.

2. Kolektor odesílá IDR, σ - signatura kolektoru. IDR je opět identifikátor kolektoru. Signaturu je předtím třeba vygenerovat, což provedou následující operace. Prvně se náhodně vygeneruje r_S. Pak se provedou následující operace:

$$t_S = g^{r_S}$$

$$e_S = SHA - 256(t_S)$$

$$s_S = r_S - e_S * sk_S \, mod \, q$$

$$\sigma = (e_S, s_S)$$

3. Nyní senzor ověřuje signaturu σ a vypočítává sdílené tajemství:

$$t'_S = g^{s_S} * pk_S^{e_S}$$

$$e'_S = SHA - 256(t'_S)$$

V_U je náhodně generovaná hodnota. Pro generování sdíleného tajemství jsou následující operace:

$$t = g^{v_U}$$

$$\kappa = t^{r_S}$$

$$e_U = SHA - 256(t,)$$

,

$$s_U = v_U - e_U * sk_U \, mod \, q$$

Následně se provádí šifrování pomocí AES v GCM režimu:

$$C1, TAG1 = ENC_{AES-GCM}(IVU, \kappa, DATA)$$

4. Senzor odesílá , IVU, C1, TAG1 kolektoru, IVU je inicializační vektor pro šifrování. C1 jsou šifrovaná data a TAG1 je autentizační tag.

$$\pi = (e_U, s_U)$$

5. Kolektor finálním krokem ověřuje signaturu a dešifruje zprávu:

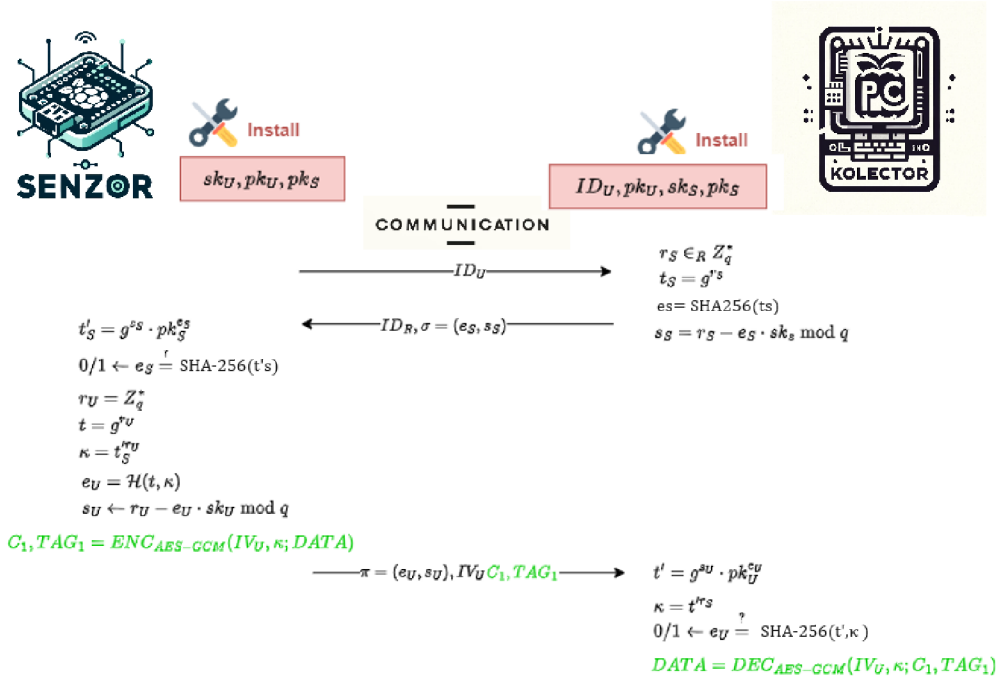
$$t' = g^{s_U} * pk_U^{e_U}$$

$$\kappa' = t'^{r_S}$$

$$e'_U = SHA - 256(t', \kappa')$$

$$DATA = DEC_{AES-GCM}(IVU, \kappa', C1, TAG1)$$

Obr. 5.2: Diagram komunikace protokolu 2



5.1.3 Protokol 3

Jedná se o asymetrický protokol, jenž je navržen pro využití standardizovaných kryptografických algoritmů. Zde se pro algoritmus pro digitální podpisy ECDSA. ECDSA je algoritmus pro digitální podpisy, který využívá kryptografii na eliptických křivkách k zajištění autenticity a integrity zpráv. ECDSA je varianta DSA,
která nabízí vyšší bezpečnost s menšími klíčovými velikostmi, což zvyšuje efektivitu a snižuje požadavky na výpočetní výkon a paměť. Algoritmus by použit z knihovny MicroECC.

Prvně je třeba nainstalovat klíče. Senzor instaluje tři klíče: soukromý klíč skU, veřejný klíč pkU a veřejný klíč kolektoru pkS. Kolektor také instaluje tři klíče: soukromý klíč skS, veřejný klíč pkS a veřejný klíč senzoru pkU. Následuje komunikace, která je na obrázku 5.3 a níže je detailněji popsaná.

1. Senzor odesílá IDU kolektoru, čímž sám sebe identifikuje.
2. Kolektor generuje signaturu a odesílá IDR, tS
   a σS senzoru. Kolektor vygeneruje náhodné číslo r_S a vypočítá:

$$t_S = g^{r_S}$$

   Následně vytvoří signaturu a odešle ji senzoru spolu s identifikátorem IDR

a hodnotou t_S

$$\sigma_S = SIG_{ECDSA}(sk_S, t_S)$$

3. Číslo r_U je náhodně vygenerované. Senzor ověřuje signaturu a vypočítává sdílené tajemství:

$$VERIFY_{ECDSA}(pk_S, \sigma_S)$$

$$t = g^{v_U}$$

$$\kappa = KDF_{ECDH}(t_S, r_U)$$

$$\sigma_U = SIG_{ECDSA}(sk_U, t)$$

4. Senzor šifruje data pomocí AES v GCM režimu a odesílá σ_U, IVU, C1 a TAG1 kolektoru:

$$C1, TAG1 = ENC_{AES-GCM}(IVU, \kappa, DATA)$$

Odesílá signaturu $\sigma_U = (r_U, s_U)$, inicializační vektor IVU, šifrovaná data C1 a autentizační tag TAG1.

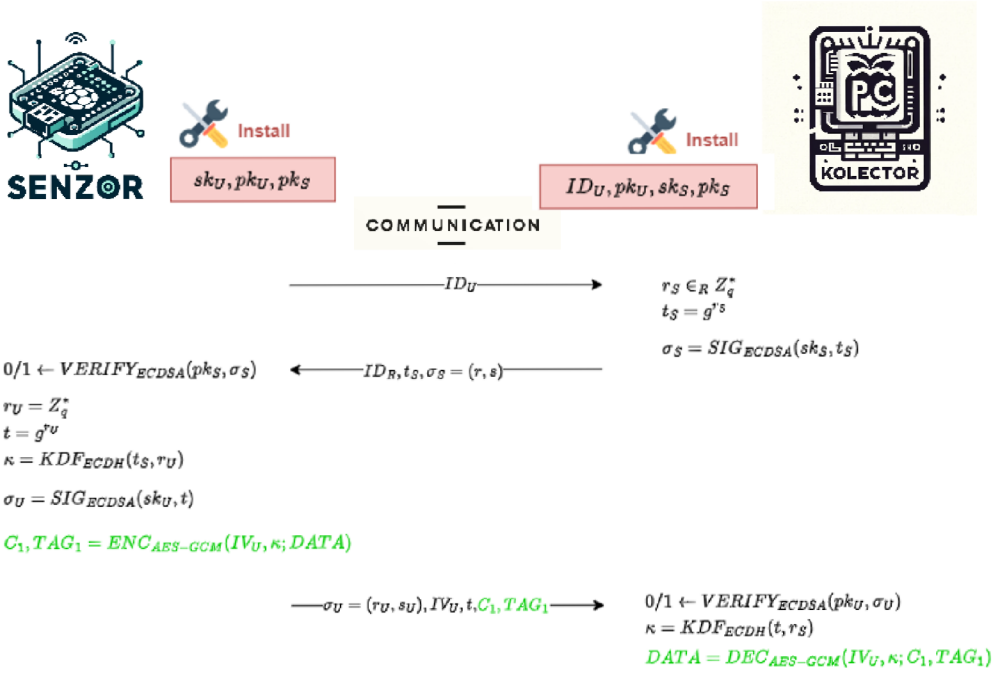
5. Kolektor ověřuje signaturu, vypočítá sdílené tajemství a dešifruje zprávu:

$$VERIFY_{ECDSA}(pk_U, \sigma_U)$$

$$\kappa = KDF_{ECDH}(t, r_S)$$

$$DATA = DEC_{AES-GCM}(IVU, \kappa, C1, TAG1)$$

Obr. 5.3: Diagram komunikace protokolu 3



5.2 Integrace protokolů do systému

V této sekci práce bude vytvořená jednoduchá aplikace, jenž bude umožňovat komunikaci mezi Raspberry Pi Zero, dále uváděno jako senzor a počítačem, dále již uváděno jako kolektor. Prvně bude vytvořeno jednoduché prostředí pro další operace v programovacím jazyce Python. Na kolektoru se vytvoří soubor cryptu_gui.py a na senzoru app.py. Aplikace bude strukturována tak, že se prvně spustí aplikace senzoru a ta bude vyčkávat na inicializaci komunikace od kolektoru.
V aplikaci kolektoru bude velice jednoduché grafické rozhraní, ve kterém budou tři tlačítka, každé pro jeden určitý protokol pro zabezpečení komunikace. Senzor bude v kódu simulovat generování hodnot o teplotě a po vyžádání kolektoru tuto teplotu odešle.

Prvně je v Python aplikacích nutné nalinkovat knihovny, aby je bylo možné použít. Toto lze vidět na obrázku 5.4. Načtení na obrázku je provedeno na senzoru a knihovny protokolů jsou zde zkompilované s příponou .so. Na kolektoru je načtení stejné, až na přípony souborů knihoven, které jsou zde .dll z důvodu jiného operačního systému, kterým je na kolektoru Windows 11.

Obr. 5.4: Načtení knihoven do aplikace na senzoru

```
# Load the shared libraries
so_path_1 = os.path.abspath("crypto_protocol1.so")
so_path_2 = os.path.abspath("crypto_protocol2.so")
so_path_3 = os.path.abspath("crypto_protocol3.so")
lib_1 = ctypes.CDLL(so_path_1)
lib_2 = ctypes.CDLL(so_path_2)
lib_3 = ctypes.CDLL(so_path_3)
```

Komunikace zařízení bude probíhat po Wi-Fi. Toto je uskutečněno pomocí funkce socket, ten v Python aplikacích slouží k vytváření síťových spojení mezi různými zařízeními nebo aplikacemi přes síť. Socket je základní stavební blok pro síťovou komunikaci, který umožňuje přenos dat mezi klientem a serverem. V Pythonu lze sockety použít pro různé typy síťové komunikace, včetně TCP a UDP protokolů. Zařízení musí mít před komunikací definovány IP adresy a porty. Toto lze vidět na obrázku 5.5. IP adresa, na kterou senzor naslouchá je 0.0.0.0, což znamená, že senzor bude přijímat připojení z jakékoli IP adresy, která se může k serveru připojit. Na kolektoru je tato IP adresa nastavena na IP senzoru, což je přesně 10.0.1.16. Kolektor vždy inicializuje spojení a to vždy po zvolení protokolu.

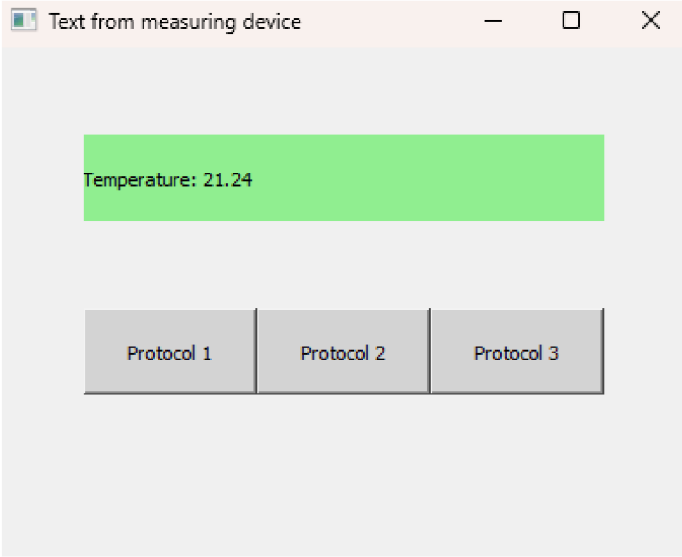
Dále se v kódu aplikace nachází hlavně funkce pro správné provedení protokolů a některé jejich debugovací funkce a to z důvodu neúplné funkcionality protokolů v reálné komunikaci. Aplikace na kolektoru obsahuje samozřejmě kód pro jednoduché

Obr. 5.5: Nastavení IP adresy a portu senzoru

```
# Define server IP and port
SERVER_IP = '0.0.0.0'
SERVER_PORT = 5005
```

grafické prostředí, které je na obrázku 5.6. Kromě tlačítek se zde nachází i pole pro výpis zjištěné teploty.

Obr. 5.6: Grafické rozhraní aplikace na kolektoru



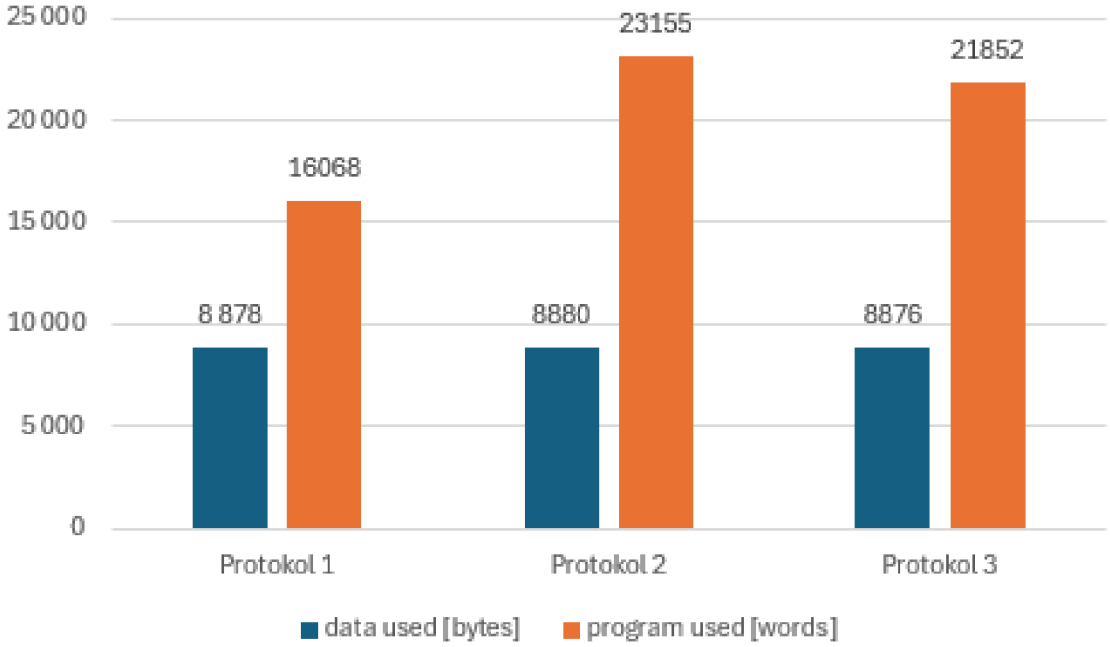
Ve složce aplikace se nacházejí zkompilované kryptografické knihovny, jejich zdrojové soubory a samozřejmě zdrojové soubory knihoven MbedTLS ve složce mbedtls a zdrojové soubory knihovny MicroECC. Struktura souboru je v příloze A.

5.3 Výkonnostní testy protokolů

5.3.1 Testy na zařízení Curiosity Development Board

V této podsekci bude provedeno jednoduché porovnání protokolů na zařízení Curiosity Development Board PIC24F 16BIT MCU, na němž byly základy protokolů sestaveny. Zde bude porovnání velice Krátké a výsledky toho moc nevypoví. To je z důvodu možnosti měření na tomto zařízení. Algoritmy zde nelze pozorovat za chodu díky stavu, v jakém zařízení je. Neobsahuje komponenty důležité pro externí komunikaci. Taktéž velikosti protokolů jsou velice podobné a to hlavně z důvodu použití šifry AES v GCM režimu z knihovny MbedTLS pro všechny protokoly. Jak již bylo uvedeno v předešlých měřeních knihoven, MbedTLS zabírá velké množství

Obr. 5.7: Porovnání protokolů na zařízení Curiosity Development Board



úložiště. Naopak knihovna MicroECC, jenž je využita v protokolech dvě a tři příliš místa nezabírá a přímo kód implementovaných protokolů také neudělá ve velikosti příliš velké rozdíly. Velikost protokolů tedy především závisí na šifře AES. Graf s porovnáním protokolů je na obrázku 5.7. Je zde možné pozorovat, že protokol 1 zabírá
o něco méně programové paměti ve slovech. To je v důsledku jednodušší implementace a také kvůli tomu, že protokol nepoužívá knihovnu MicroECC. Jinak jsou na tom protokoly velice podobně.

5.3.2 Testy na zařízení Raspberry Pi Zero

Zde je hned ze začátku důležité zdůraznit a rozebrat celkovou funkčnost protokolů. Protokol 1 funguje i v opravdové komunikaci v aplikaci. Je tedy možné pomocí něj přenášet zabezpečená data. Protokol 2 ovšem nefunguje vůbec a protokol 3 nefunguje pouze v reálné komunikaci. Problémy může způsobovat například knihovna MicroECC, jenž v procesu debugování vypisovala časté chyby. Dále je možné špatné užití funkcí a algoritmů knihovny. Také zde může hrát roli prostředí, v němž byly protokoly vyvíjeny. Při hledání chyb vše poukazovalo na funkce týkající
se eliptických křivek knihovny MicroECC, chybu se ovšem nepodařilo přímo identifikovat
a napravit.

V tabulce jsou výsledky testování rychlostí provádění protokolů. Opět zde byl pro měření použit jednoduchý program, který provedl měření stokrát a vypočetl aritmetický průměr. Protokol 1 je velice jednoduchý a rychlý symetrický protokol, takže dává smysl, že je výrazně rychlejší. Problém ovšem nastává u protokolu 2 a 3. Třetí protokol by měl bez využití kryptografického jádra výrazně pomalejší, ale není tomu tak. Raspberry Pi Zero samozřejmě kryptografický procesor neobsahuje, jedná se pouze o počítač určený k vývoji a testování. Příčina je to, že testovací aplikace protokolu 3 plně funguje, zatímco protokol 2 nefunguje vůbec. Proto měření pro protokol 2 je nic neříkající.

Tab. 5.1: Měření času pro jednotlivé protokoly

Protokol	**Čas [s]**
Protokol 1	0,002541
Protokol 2	0,094201
Protokol 3	0,046406

Závěr

Cílem bakalářské práce bylo nastudovat problematiku IoT, hlavně analyzovat operační systémy a kryptografické knihovny, jenž se zde používají. Následně měly být testovány kryptografické knihovny. V důsledku zjištění z měření měly být některé knihovny vybrány k testování na paměťově, výkonově a výpočetně omezeném zařízení Curiosity Development Board PIC24F 16BIT MCU. Zde měly být nejvíce efektivní knihovny vybrány pro implementaci v praktickém využití. Následně mělo proběhnout navrhnutí a implementace systému.

V teoretické části byla prvně jednoduše probrána problematika IoT a mírně zde byla řešena i kvantová kryptografie. Následně byly řešeny a porovnány operační systémy a kryptografické knihovny, jež jsou využívány v tomto prostředí. Podrobně byly analyzovány kryptografické knihovny a jejich primitiva.

V praktické části pak byly kryptografické knihovny testovány na co nejvíce jednotné platformě. Jako zařízení, na němž byly knihovny testovány, bylo vybráno zařízení Raspberry Pi Zero. Zde proběhlo podrobné testování a následně byly vybrány knihovny k dalšímu testování již na omezeném zařízení Curiosity Development Board PIC24F 16BIT MCU. Přesněji se jednalo o knihovny MbedTLS, TinyCrypt a MicroECC, jenž byly jako jedny z málo knihoven kompatibilní s 16 bitovými mikroprocesory. Tyto knihovny byly porovnány a do praktické části byly vybrány knihovny MbedTLS a MicroECC z důvodu potřeby funkcí, jenž obsahují. Následně byly sestaveny tři protokoly využívající primitiva těchto knihoven. Poté byly protokoly implementovány v praktické aplikaci.

Při vývoji a testování protokolů bylo objeveno mnoho problému s implementací. Knihovna MbedTLS není plně kompatibilní s 16 bitovými zařízeními a bylo třeba některé části tomuto prostředí přizpůsobit. Z praktického hlediska bylo mnoho problémů i s knihovnou MicroECC, jenž neimplementuje modulární funkce zrovna nejlépe. Taktéž je knihovna velice minimalistická a bylo těžké zjistit, proč některé funkce nefungují tak jak mají.

Ve výsledku byla provedena funkční implementace pouze prvního symetrického protokolu. Oba protokoly, jež obsahovaly funkce na eliptických křivkách pořádně nefungovaly. Protokol dva, jenž obsahoval Schnorrův podpis nefungoval ani v testovacích aplikacích. Vždy vypisoval nějakou chybu v dešifrování nebo nějakou chybu v ověření podpisu. Po hledání příčiny bylo zjištěno, že za to nejspíš může špatná implementace Schnorrova podpisu, chybu se ovšem nepodařilo přímo identifikovat. Protokol tři fungoval v testovacích aplikacích, ale z nějakého důvodu během implementace v opravdové komunikaci selhával. Chyba byla podle všech důkazů opět

v podpisu, i když zde byl použit algoritmus ECDSA. Chyba je tedy nejspíš ve špatném použití funkcí týkajících se eliptických křivek knihovny MicroECC.

Finálně byly provedeny testy protokolů, které byly ovšem velice ovlivněny nefunkčností druhého protokolu, jenž měl být rychlejší a méně náročný na provedení, než protokol tři. To výsledky ovšem neukázaly.

Literatura

[1] DENER, Murat. GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES, GAZI UNIVERSITY, ANKARA, TURKEY. *Comparison of Encryption Algorithms in Wireless Sensor Networks* [.pdf]. 2018. Dostupné z: https://www.itm-conferences.org/articles/itmconf/pdf/2018/07/itmconf_cmes2018_01005.pdf. [cit. 2023-12-10]

[2] *Choose a perfect IoT Operating system for your IoT operation...!!!* [Online]. 2020. Dostupné z: https://dev.to/spectrumcetb/choose-a-perfect-iot-operating-system-for-your-iot-operation-13a0. [cit. 2023-12-10]

[3] K. ROUTRAY, Sudhir, Mahesh K. JHA, Laxmi SHARMA, Rahul NYAMANGOUDAR a Abhishek JAVALI. DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING CMR INSTITUTE OF TECHNOLOGY, BANGALORE, INDIA. *Quantum Cryptography for IoT: APerspective* [.pdf]. 2017. Dostupné z: https://cmapscloud.ihmc.us/rid=1TG41MX26-1VTFQ2Y-1R1/Quantum%20cryptography%20for%20IoT-%20APerspective.pdf. [cit. 2023-12-10]

[4] FREE UNIVERSITY OF BERLIN FRENCH INSTITUTE FOR RESEARCH IN COMPUTER SCIENCE AND AUTOMATION HAMBURG UNIVERSITY OF APPLIED SCIENCES. *Riot-os* [Online]. 2023. Dostupné z: https://www.riot-os.org. [cit. 2023-12-10]

[5] ZINZINDOHOUÉ, Jean Karim, Jonathan PROTZENKO, Karthikeyan BHARGAVAN a Benjamin BEURDOUCHE. *HACL : A Verified Modern Cryptographic Library* [.pdf]. 2017. Dostupné z: https://dl.acm.org/doi/pdf/10.1145/3133956.3134043. [cit. 2023-12-10]

[6] LINUX FOUNDATION, WIND RIVER SYSTEMS. *ZEPHYR* [Online]. 2023. Dostupné z: https://docs.zephyrproject.org/latest/index.html. [cit. 2023-12-10]

[7] KUMAR, Uday, Tuhin BORGOHAIN a Sugata SANYAL. *Comparative Analysis of Cryptography Library in IoT* [.pdf]. 2015. Dostupné z: https://arxiv.org/ftp/arxiv/papers/1504/1504.04306.pdf. [cit. 2023-12-10]

[8] AMAZON WEB SERVICES. *FreeRTOS* [Online]. 2023. Dostupné z: https://www.freertos.org/index.html. [cit. 2023-12-10]

[9] *TinyOS* [Online]. 2023. Dostupné z: https://github.com/tinyos/tinyos-main. [cit. 2023-12-10]

[10] *IoT Operating Systems* [Online]. 2022. Dostupné z: https://www.javatpoint.com/iot-operating-systems. [cit. 2023-12-10]

[11] *Nano-RK* [Online]. 2022. Dostupné z: https://en.wikipedia.org/wiki/Nano-RK. [cit. 2023-12-10]

[12] XIONG, Xiaokang, Duncan S WONG a Xiaotie DENG. DEPARTMENT OF COMPUTER SCIENCE CITY UNIVERSITY OF HONG KONG HONG KONG, CHINA. *TinyPairing: A Fast and Lightweight Pairing-based Cryptographic Library for Wireless Sensor Networks* [.pdf]. 2010. Dostupné z: https://www.wolfssl.com. [cit. 2023-12-10]

[13] WOLFSSL INC. *WolfSSL* [Online]. 2023. Dostupné z: https://www.wolfssl.com. [cit. 2023-12-10]

[14] FREE UNIVERSITY OF BERLIN FRENCH INSTITUTE FOR RESEARCH IN COMPUTER SCIENCE AND AUTOMATION HAMBURG UNIVERSITY OF APPLIED SCIENCES. *HACL* High Assurance Cryptographic Library* [Online]. 2023. Dostupné z: https://api.riot-os.org/group___pkg___hacl.html. [cit. 2023-12-10]

[15] FREE UNIVERSITY OF BERLIN FRENCH INSTITUTE FOR RESEARCH IN COMPUTER SCIENCE AND AUTOMATION HAMBURG UNIVERSITY OF APPLIED SCIENCES. *Relic toolkit for RIOT* [Online]. 2023. Dostupné z: https://api.riot-os.org/group___pkg___relic.html. [cit. 2023-12-10]

[16] *Mbed TLS* [Online]. 2023. Dostupné z: https://en.wikipedia.org/wiki/Mbed_TLS. [cit. 2023-12-10]

[17] THE MBED TLS CONTRIBUTORS. *Mbed TLS documentation hub* [Online]. 2023. Dostupné z: https://mbed-tls.readthedocs.io/en/latest/. [cit. 2023-12-10]

[18] *TinyCrypt Cryptographic Library* [Online]. 2022. Dostupné z: https://docs.zephyrproject.org/latest/services/crypto/tinycrypt.html. [cit. 2023-12-10]

[19] OPENSSL PROJECT AUTHORS *OpenSSL* [Online]. 1999-2023. Dostupné z: https://www.openssl.org/docs/man3.0/man7/crypto.html. [cit. 2023-12-10]

[20] EMILE VAN DER LAAN *AvrCryptoLib* [Online]. Dostupné z: https://www.emsign.nl. [cit. 2023-12-10]

[21] NATIONAL INSTITUTE OF STANDARTS AND TECHNOLOGY. *Report on Lightweight Cryptography* [.pdf]. 2016. Dostupné z: https://csrc.nist.gov/CSRC/media/Publications/nistir/8114/draft/documents/nistir_8114_ [cit. 2023-12-10]

[22] MASANOBU KATAGI, SHIHO MORIAI, SONY CORPORATION. *Lightweight Cryptography for the Internet of Things* [.pdf]. Dostupné z: https://citeseerx.ist.psu.edu/document?repid=rep1type=pdfdoi=9595b5b8db9777d57956258 [cit. 2023-12-10]

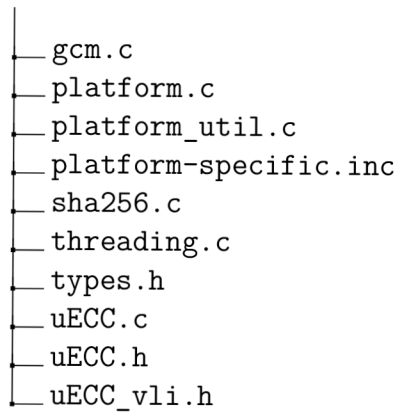
Seznam symbolů a zkratek

WSN	bezdrátová senzorová síť – Wireless sensor networ
RTOS	operační systém reálného času – real-time operating system
SSL	vrstva bezpečných socketů – secure sockets layer
WSN	bezdrátová senzorová síť – Wireless sensor networ
TCP/IP	Protokol řízení přenosu/internetový protokol – Transmission Control Protocol/Internet Protocol
IoT	internet věcí – Internet Of Things
AES	Pokročilý šifrovací standard – Advanced Encryption Standard
SHA-256	Bezpečný hašovací algoritmus 256 bitů – Secure Hash Algorithm 256-bit
ECDH	Diffie-Hellman na eliptických křivkách – Elliptic Curve Diffie-Hellman
ECDSA	Algoritmus digitálního podpisu na eliptických křivkách – Elliptic Curve Digital Signature Algorithm
GCM	Galoisův/čítací režim –Galois/Counter Mode

Seznam příloh

A Obsah přiloženého archivu

- / ..... Root directory of the App_crypto application
  - _pycache_ ..... Directory for Python bytecode files
    - crypto_protocol.cpython-312.pyc ..... Compiled Python file
  - mbedtls ..... Directory for mbedtls library
    - aes.h
    - alignment.h
    - build_info.h
    - cipher.h
    - cipher_wrap.h
    - common.h
    - config_adjust_legacy_crypto.h
    - config_adjust_ssl.h
    - config_adjust_x509.h
    - constant_time.h
    - constant_time_impl.h
    - constant_time_internal.h
    - ctr.h
    - error.h
    - gcm.h
    - mbedtls_config.h
    - platform.h
    - platform_util.h
    - private_access.h
    - sha256.h
    - threading.h
  - aes.c
  - cipher.c
  - cipher_wrap.c
  - constant_time.c
  - crypto_gui.py
  - crypto_protocol1.c
  - crypto_protocol1.dll
  - crypto_protocol1.h
  - crypto_protocol2.c
  - crypto_protocol2.dll
  - crypto_protocol2.h
  - crypto_protocol3.c
  - crypto_protocol3.dll
  - crypto_protocol3.h
  - curve-specific.inc
  - crypto_protocol3.so
  - crypto_protocol2.so
  - crypto_protocol1.so
  - app.py


gcm.c
platform.c
platform_util.c
platform-specific.inc
sha256.c
threading.c
types.h
uECC.c
uECC.h
uECC_vli.h