VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a
komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Maxim Ilyushchenkov

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

APLIKOVANÁ KRYPTOGRAFIE V IOT

APPLIED CRYPTOGRAPHY IN IOT

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Maxim Ilyushchenkov

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Lukáš Malina, Ph.D.

SUPERVISOR

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Maxim Ilyushchenkov ***ID:*** 211794 ***Ročník:*** 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Aplikovaná kryptografie v IoT

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s kryptografickými schématy poskytujícími zabezpečení přenosu dat a ochranu soukromí uživatelů v rámci systémů a aplikací v prostředí IoT. Proveďte analýzu kryptografických softwarových knihoven (např. mbedTLS, OpenSSL, PionDTLS) a zhodnoťte jejich využitelnost na omezených zařízeních v rámci IoT (např. chytré hodinky, mikropočítače). Proveďte ucelené testování kryptografických knihoven a jejich primitiv/operací včetně měření paměťové a výpočetní efektivity na zvolených platformách.
V rámci bakalářské práce předložte návrh řešení zajišťující bezpečnost datového přenosu protokolu MQTT a proveďte verifikační implementaci schématu na vybrané platformě. Cílem bakalářské práce je plně funkční implementace řešení poskytující bezpečnost přenášených dat a autentizaci stran. Dalším cílem je testování řešení a měření výpočetní a paměťové náročnosti implementace na zvolených omezených platformách.

DOPORUČENÁ LITERATURA:

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

[2] KOTHMAYR, Thomas, et al. DTLS based security and two-way authentication for the Internet of Things. AdHoc Networks, 2013, 11.8: 2710-2723.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: doc. Ing. Lukáš Malina, Ph.D.

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

Abstrakt

Bakalářská práce je věnována bezpečnosti v systémech IoT. V práci je popsána architektura IoT, jak zajistit její bezpečnost a různé typy útoků na zařízení IoT. Následně je popsána ochrana osobních údajů IoT a řešení pro její ochranu. Dále je v práci vysvětlena technologie MQTT, její princip fungování, architektura a základní prvky. V rámci praktické části je popsána implementace publishera a subscribera, kteří spolu komunikují pomocí šifrovaných zpráv. Následně je popsána knihovna eciespy, která umožňuje šifrování a dešifrování zpráv. Je také popsán způsob komunikace mezi serverem a klientem pomocí protokolu TLS a výměny certifikátů. A na konci je řešení pro implementaci zabezpečené komunikace mezi mikropočítači pomocí MQTT.

Klíčová slova

Bezpečnost IoT, MQTT, Mosquitto, AES, eciespy, TLS/SSL, Rabin, Raspberry Pi

Abstract

The bachelor thesis is devoted to security in IoT systems. The thesis describes the IoT architecture, how to ensure its security and various types of attacks on IoT devices. Subsequently, the protection of personal data IoT and solutions for its protection are described. The work also describes the MQTT technology, its principle of operation, architecture and basic elements. The practical part describes the implementation of the publisher and subscriber, who communicate using encrypted messages. Afterwards, the eciespy library is described, which enables encryption and decryption of messages. The method of communication between the server and the client using the TLS protocol and certificate exchange is also described. Finally, there is a solution for implementing secure communication between microcomputers using MQTT.

Keywords

IoT security, MQTT, Mosquitto, AES, eciespy, TLS/SSL, Rabin, Raspberry Pi

Bibliografická citace:

ILYUSHCHENKOV, Maxim. Aplikovaná kryptografie v IoT [online]. Brno, 2020 [cit. 2020-11-23]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/130404. Semestrální práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Lukáš Malina.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma „Aplikovaná kryptografie v IoT“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si 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.

V Brně dne: ……..... ……

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Lukáši Malinovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: …….....

……
podpis autora

Obsah

Seznam symbolů a zkratek

Zkratky:

IoT	...	Internet of Things
IWF	...	Internet World Forum
M2M	...	Machine to Machine
CIA	...	Confidentiality, Integrity, Availability
DDoS	...	Denial of Service
RAM	...	Random Access Memory
IPSec	...	Internet Protocol Security
ECIOT	...	Elliptical Curve Internet of Things
Ex-OR	...	Exclusive or
RSA	...	Rivest–Shamir–Adleman
AES	...	Advanced Encryption Standard
RFID	...	Radio-frequency Identification
MITM	...	Men-in-the-middle
GPS	...	Global Positioning System
OASIS	...	Organization for the Advancement of Structured Information Standards
MQTT	...	Message Queuing Telemetry Transport
IP	...	Internet Protocol
TLS	...	Transport Layer Security
CBC	...	Cipher Block Chaining
ECB	...	Electronic Codebook
OFB	...	Output-Feedback
CFB	...	Cipher FeedBack
CTR	...	Counter
ECIES	...	Elliptic Curve Integrated Encryption Scheme
ECC	...	Elliptic Curve Cryptography
GCM	...	Galois/Counter Mode
ECDLP	...	Elliptic Curve Discrete Logarithm Problem

ASCII	...	American Standard Code for Information Interchange
TCP/IP	...	Transmission Control Protocol/Internet Protocol
SSL/TLS	...	Secure Sockets Layer / Transport Layer Security
HTTP	...	Hypertext Transfer Protocol
MAC	...	Media Access Control

Seznam obrázků

Seznam tabulek

1. ÚVOD

Tato bakalářská práce se věnuje tématu bezpečnosti v IoT. V současné době se internet věcí (IoT) stal jedním z nejzajímavějších témat mezi vědci a odborníky. Považuje se to za univerzální přítomnost, která umožňuje připojení všech objektů / věcí v našem prostředí přes internet s možností vzájemného propojení bez lidského zásahu. IoT zahrnuje různé objekty, které lze připojit pomocí bezdrátové nebo kabelové sítě. Tyto objekty mají unikátní schéma adresování, které jim umožňuje vzájemně komunikovat a spolupracovat při vytváření nových aplikací a služeb, jako jsou inteligentní domy, inteligentní doprava, propojená auta, inteligentní sítě, inteligentní města, inteligentní řízení dopravy a další.

Sociální přijetí aplikací a služeb IoT silně závisí na důvěryhodnosti informací a ochraně soukromých dat. Protože IoT je složitý, distribuovaný a heterogenní systém, čelí několika výzvám týkající se bezpečnosti a soukromí. V současné době je budování efektivní a spolehlivé bezpečnostní techniky jednou z nejvyšších priorit, které je třeba zvážit. Přestože řada vědců představila několik řešení v oblasti bezpečnosti a ochrany soukromí, stále existuje poptávka po spolehlivé technice zabezpečení pro IoT, aby byly splněny požadavky na důvěrnost údajů, integritu, soukromí a důvěru [1].

V rámci bakalářské práce byla provedena implementaci zabezpečené komunikace mezi mikropočítači pomocí MQTT.

2. ARCHITEKTURA A BEZPEČNOST IOT

2.1 Architektura IOT

Výbor pro architekturu Světového fóra internetu věcí (IWF) vydal v roce 2014 model internetu věcí (Obrázek 2-1 (převzato z [15])). Tento model funguje jako obecný rámec, který pomáhá průmyslovému odvětví urychlit nasazení IoT. Tento model je navržen tak, aby podporoval spolupráci a vývoj nových modelů pro internet věcí. Model se skládá ze sedmi vrstev, kde každá vrstva poskytuje další informace pro stanovení společné terminologie, jak je uvedeno v tabulce 1 [2]. Také kategorizuje, kde se na různých vrstvách modelu IoT provádějí různé druhy procesů. Tento model navíc umožňuje různým výrobcům vyrábět produkty IoT, které jsou vzájemně interoperabilní, což transformuje IoT z koncepčního modelu na skutečné a dostupné schéma [2].

obr. 2-1 Architektura IoT

Tabulka 2-1 Referenční model IoT podle IWF

Vrstva	Název	Popis
7	Spolupráce a procesy	Zapojení lidí a obchodních procesů
6	Aplikační vrstva	Podávání zpráv, analýza a kontrola
5	Vrstva abstrakce dat	Agregace a přístup
4	Akumulace dat	Big data a data o ukládání věcí
3	Výpočetní vrstva	Analýza a transformace datových prvků
2	Komunikační vrstva	Komunikační jednotky, protokoly, sítě, M2M atd.
1	Fyzická vrstva	Zařízení, senzory, ovladače atd.

Připojená zařízení nepřetržitě komunikují mezi sebou navzájem a s cloudem pomocí různých typů bezdrátových protokolů. Taková komunikace umožňuje nejen vytváření flexibilních aplikací IoT, ale také může vytvářet zranitelnosti zabezpečení v systému IoT, otevírá příležitosti pro útočníky a náhodné úniky dat.

Chcete-li chránit uživatele, zařízení a firmy, musíte efektivně zabezpečit svoje zařízení IoT. Jádrem zabezpečení IoT je konfigurace připojení mezi zařízeními, ovládání těchto připojení a jejich efektivní správa. Efektivní ochrana zajišťuje soukromí dat, omezuje přístup k zařízením a cloudovým prostředkům, umožňuje zabezpečené cloudové připojení a kontroluje využití zařízení. Strategie zabezpečení IoT řeší chyby zabezpečení pomocí zásad, jako je správa identit zařízení, šifrování a řízení přístupu [3].

2.2 Jak zabezpečit IoT zařízení

Zabezpečení systému IoT lze posoudit použitím klasických opatření pro zabezpečení a analýzu rizik. V systému IoT by měly být použity typické bezpečnostní požadavky CIA (Confidentiality, Integrity and Availability).

Důvěrnost (Confidentiality) je zajištění toho, že informace je přístupná jen těm, kteří jsou oprávněni k ní mít přístup. A toho lze dosáhnout například šifrováním.

Integrita (Integrity) se používá k zajištění obsahu zpráv vyměňovaných mezi odesílatelem a přijímačem, který je chráněn proti jakékoli manipulaci útočníkem, aniž by příjemce mohl tuto manipulaci sledovat.

Dostupnost (Availability) se používá k zajištění toho, že uživatel se zlými úmysly není schopen narušit nebo škodlivě ovlivnit komunikaci nebo kvalitu služeb poskytovaných zařízeními IoT nebo komunikační sítí [4].

Přestože je CIA pro IoT zásadní, je třeba implementovat další bezpečnostní požadavky pro každou úroveň architektury IoT, jak ukazuje tabulka 2-1 [5]. Ověřování

uzlů je hlavním bezpečnostním problémem fyzické vrstvy, aby se zabránilo neautentizovanému přístupu uzlu a udržovalo komunikační kanál mezi uzly IoT v bezpečí před jakýmkoli typem útoku. Lehký kryptografický algoritmus a protokol je důležitým aspektem k šifrování přenášených dat, zejména u zařízení IoT s omezenými prostředky. Pro síťovou vrstvu jsou nutná opatření zabezpečení komunikace a ověřování identity, aby se zabránilo nelegálním uzlům. Na této úrovni je také běžný útok DDoS (Distributed Denial of Service), takže je potřeba chránit před útokem DDoS v bezbranných uzlech v této vrstvě, zvláště v kontextu IoT je závažnější. Pro data abstrakce, akumulace a výpočetní úrovni je zapotřebí mnoho mechanismů zabezpečení aplikace k zabezpečení dat uložených v cloud computingu. Kromě aktualizovaného antiviru jsou zapotřebí silné šifrovací algoritmy. Pokud jde o úroveň aplikací a spolupráce, je třeba přijmout autentizaci a klíčovou dohodu na ochranu soukromí uživatele. Pro zabezpečení informací na této úrovni je navíc nezbytná správa hesel [5].

Tabulka 2-2 Bezpečnostní požadavky pro každou vrstvu architektury IoT

<table>
<tr><td>Vrstva</td><td>Bezpečnostní požadavky</td></tr>
<tr><td>Spolupráce a procesy</td><td rowspan="2">Autentizace a dohoda o klíči
Ochrana soukromí
Bezpečnostní vzdělávání a management</td></tr>
<tr><td>Aplikační vrstva</td></tr>
<tr><td>Vrstva abstrakce dat</td><td rowspan="3">Bezpečný vícestranný výpočet
Bezpečný cloud computing
Aktualizace antiviru</td></tr>
<tr><td>Akumulace dat</td></tr>
<tr><td>Výpočetní vrstva</td></tr>
<tr><td>Komunikační vrstva</td><td>Autentizace identity
Šifrovací mechanismus
Zabezpečení komunikace</td></tr>
<tr><td>Fyzická vrstva</td><td>Odlehčená šifrovací technologie
Ochrana dat senzoru
Klíčová dohoda</td></tr>
</table>

2.3 Postup pro zabezpečení zařízení IoT

Bezpečnostní problémy spojené se zařízeními IoT vytvářejí potenciální rizika v našem životě. Před vystoupením IoT může narušení zabezpečení vést ke ztrátě vašich peněz, ale s IoT může útok zabezpečení vést doslova ke ztrátě života. Zabezpečení zařízení IoT vyžaduje převzetí sady osvědčených postupů, které zahrnují následující:

• Fyzická ochrana: udržování zařízení IoT v izolaci a pouze někteří lidé k nim mají fyzický přístup, to jsou hlavní kroky k tomu, aby vaše zařízení IoT byla chráněna proti neoprávněné manipulaci. Také posílení zařízení IoT s fyzickým zabezpečením, jako je blokování nevyužitých portů a zakrytí kamery, jsou dobrými body, které zabrání potenciálním útočníkům v dosažení vašich dat [7].

• Aktualizace firmwaru: zařízení IoT musí být opravitelná nebo upgradovatelná se správným digitálním podpisem. Na internetu existuje několik vážných hrozeb, které ovlivňují zařízení IoT. Prodejci a poskytovatelé služeb by měli plánovat budoucí aktualizace softwaru zařízení, aby byl aktuální. Tyto aktualizace musí být provedeny včas nebo v závislosti na důležitosti aktualizace [6].

• Dynamické testování: pro zařízení IoT je zásadní projít testováním a vytvořit nejméně standardní opatření pro bezpečnost. K otestování zabezpečení zařízení IoT existují dva typy; statické a dynamické. Na rozdíl od statického testování, které se zabývá odhalováním hrozeb v softwaru, může dynamické testování zkoumat hrozby a slabá místa v hardwaru i softwaru [7].

• Silné ověřování: mnoho uživatelů internetu věcí stále používá slabá a výchozí hesla bez jakékoli aktualizace. Výrobci by měli před použitím zařízení požádat zákazníka o aktualizaci výchozího hesla se silnými hesly. Kromě toho jsou nutné alternativní způsoby rozpoznání identity a důvěryhodnosti zařízení, protože uživatelské jméno a heslo jsou není realistické pro každé zařízení, zejména pro komunikaci mezi strojí (M2M), která začíná výrazně růst [7].

2.4 Bezpečnost z hlediska kryptografie

Specifičnost zařízení IoT ukládá omezení používání kryptografických primitiv, pokud jde o paměť a výpočetní výkon.

Cíle zabezpečení pro IoT zůstávají stejné jako pro ostatní počítačové sítě, ale existuje několik jedinečných výzev, které představuje použití kryptografie veřejného klíče, které je nezbytné pro zabezpečení komunikace mezi uzly senzorů.

Kryptografické algoritmy veřejného klíče jsou obvykle implementovány komplexně, což má za následek významnou režii, pokud jde o čas provedení a spotřebu energie. Senzory jsou hlavními problémy topologie IoT. Tato miniaturní zařízení jsou navržena pro nízkou spotřebu energie, mají omezenou šířku pásma a výpočetní výkon díky 8/16 bitovému mikroprocesoru s velmi nízkou taktovací rychlostí ($\leq$ 10 MHz). Ukládání informací na těchto zařízeních je také omezené, protože většinu úložiště zabírá aplikační software, což ponechává velmi omezený prostor pro bezpečnostní protokoly: RAM má jen několik kilobajtů a flash paměť má až 256 kB pro ukládání programového kódu.

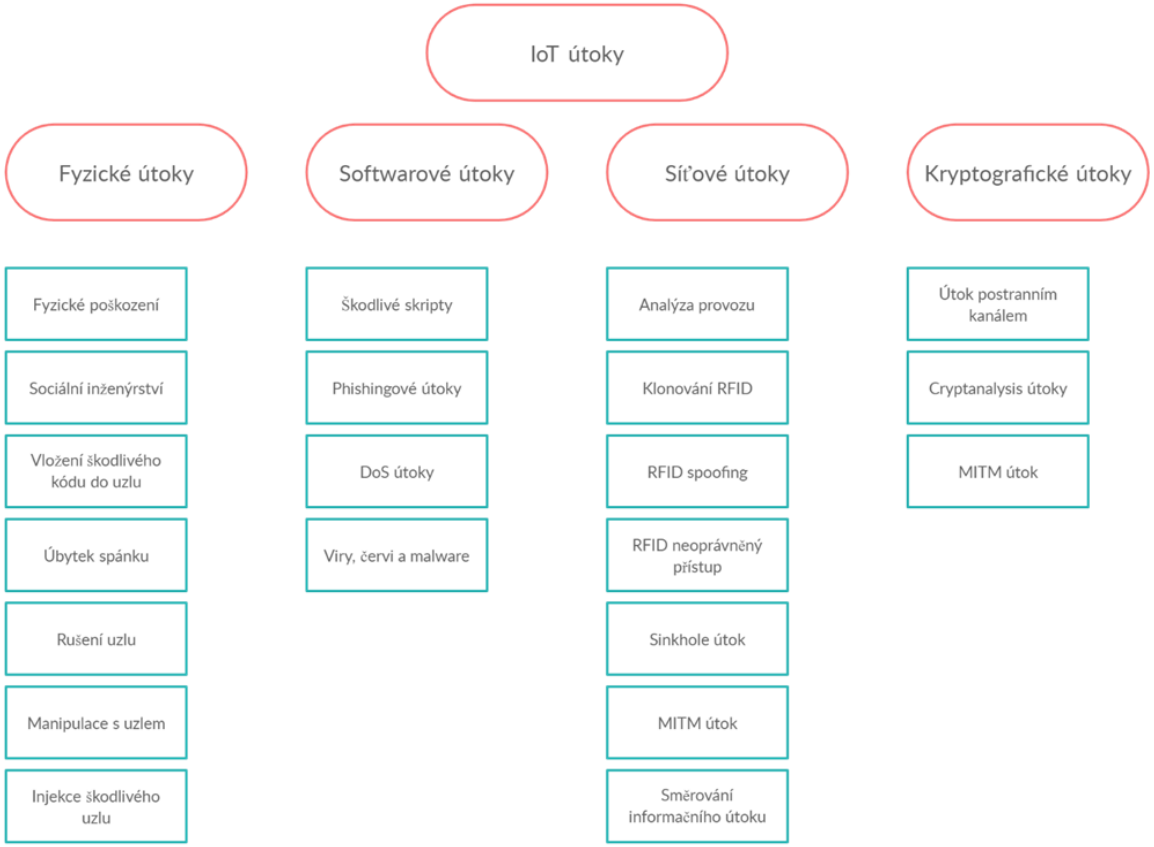
Ochrana a bezpečnostní ověření IoT by mělo být prováděno po celou dobu životního cyklu zařízení, od počáteční konfigurace až po operační prostředí, které zahrnuje bezpečné spuštění prostřednictvím ověření digitálního podpisu, řízení přístupu, ověřování zařízení, brány firewall a IPsec atd [8].

Jeden z kryptografických protokolů, který lze implementovat do zařízení IoT−Elliptical Curve Internet of Things(ECIOT) [9].

ECIOT se používá k šifrování a dešifrování zpráv na základě eliptické křivky pro komunikaci mezi bránou a zařízením IoT. Zařízení IoT budou k šifrování a dešifrování zpráv používat symetrické klíče Ex-OR. V protokolu Ex-OR musí být klíče odesílatele a příjemce stejné, protože se jedná o protokol symetrického klíče. Výměna tohoto klíče přes nezabezpečený kanál je ale riskantní kvůli možnosti, že jej zachytí hacker. Proto zařízení využívající technologii Internet věcí ve schématu ECIOT musí k vytvoření páru klíčů se serverem použít Diffie-Hellman Elliptic Curve Protocol. Je nejvýhodnější použít tento protokol na uzlech bezdrátových senzorů, protože menší velikost klíčových zařízení výrazně sníží požadavky na šířku pásma a úložiště ve srovnání s RSA [9].

3. ÚTOKY NA IOT ZAŘÍZENÍ

Systém IoT s distribuovanou a dynamickou povahou vytváří slabé komunikační kanály, které jsou využívány škodlivými objekty k zneužití a otevření nových hrozeb týkajících se sledování, monitorování a hlášení akcí uživatelů. Nárůst zařízení IoT v naší komunitě představil sadu bezpečnostních útoků, které je třeba řešit. V systému IoT existují čtyři hlavní typy útoků: fyzické, softwarové, síťové a kryptografické. Tato část poskytuje krátkou diskusi o každém typu útoku a jeho běžných příkladech v systémech IoT. Na obrázku 3-1 (překreslen z [16]) jsou shrnuty různé bezpečnostní útoky v systému IoT [10].



obr. 3-1: IoT útoky

3.1 Fyzické útoky

Tyto typy útoků se týkají hardwarových prvků systému IoT, ve kterých útočník vyžaduje fyzickou blízkost systému IoT, aby mohl útok spustit. Těchto útoků je relativně obtížné dosáhnout, protože vyžadují drahé látky. Fyzické útoky mohou mít různé formy, které zahrnují následující [10]:

• Manipulace s uzlem: tento útok se zaměřuje na uzel senzoru fyzickým poškozením nebo dokonce nahradí celý uzel nebo část jeho hardwaru, aby získal přístup k citlivým informacím.

• Fyzické poškození: tento typ útoku je podobný manipulaci s uzly, při které útočník fyzicky poškozuje zařízení IoT. Tento typ útoku je obtížné dosáhnout, protože vyžaduje, aby se útočník dostal do oblasti nebo budovy obsahující zařízení IoT, aby jej zničil. Hlavní rozdíl mezi tímto útokem a útokem na manipulaci s uzly spočívá v tom, že se útočník pokouší poškodit systém IoT přímo a ovlivnit tak dostupnost systému a kvalitu služby.

• Úbytek spánku: většina senzorů je provozována prostřednictvím baterií, které fungují podle spánkové rutiny a prodlužují tak jejich životnost. Útok deprivace spánku udržuje neustále běžící uzly, což vede k většímu hodování energie, které má za následek vypnutí uzlů po spotřebování energie baterie.

• Sociální inženýrství: útočník využívá nedostatek bezpečnostního povědomí uživatelů k manipulaci a získávání přístupu k systému IoT ke shromažďování citlivých informací nebo k provádění konkrétních činností sloužících jeho cílům.

• Injekce škodlivého uzlu: útočník získá přístup k citlivým informacím fyzickým ovládáním nového škodlivého uzlu mezi komunikujícími uzly systému IoT, který útočníkovi umožňuje kontrolovat veškerý tok dat mezi různými uzly.

3.2 Softwarové útoky

Softwarové útoky jsou hlavní příčinou většiny bezpečnostních hrozeb téměř ve všech softwarových systémech. Zaměřují se na slabiny a hrozby zjištěné při implementaci systému pomocí jeho komunikačních rozhraní. Existuje sada softwarových útoků, které zahrnují následující [10]:

• Škodlivé skripty: vzhledem k tomu, že systém IoT je propojen k internetu, útočník používá toto zařízení k vytváření škodlivých skriptů, jejichž cílem je získat přístup k citlivým datům nebo narušit dostupnost systému. Tyto škodlivé skripty jsou spouštěny uživateli systému omylem.

• Phishingové útoky: jedná se o druh útoku sociálního inženýrství, který se zaměřuje na přihlašovací údaje uživatele a další citlivé informace prostřednictvím infikovaných e-mailů nebo phishingových webů.

• DoS Attack: útočník může v systému IoT provádět Denial of Service (DoS) napříč aplikační vrstvou, která ovlivňuje všechny uživatele sítě IoT. Tento typ útoku také blokuje legální uživatele a poskytuje útočníkovi plný přístup k citlivým datům.

• Viry, červy a malware: tyto typy útoku jsou uzavřeny před útokem na škodlivý kód, při kterém útočník napíchne do systému škodlivý software, aby získal přístup do systému, ukradl citlivé informace nebo narušil dostupnost systému.

3.3 Síťové útoky

Systém IoT je kombinací sítí vzájemně propojených za účelem přenosu dat mezi různými zařízeními IoT. Síťové útoky se týkají sítě IoT, ve které útočník v zásadě nevyžaduje, aby byl blízko sítě, aby útok mohl fungovat. Existuje sada síťových útoků, které zahrnují následující [10]:

• Útok analýzou provozu: tento typ útoku zahrnuje zachycení citlivých dat a dalších typů dat kvůli jejich bezdrátové funkci. Ve většině útoků navíc musí útočník před provedením jakýchkoli útoků shromáždit určité informace o síti, čehož je dosaženo útokem analýzou provozu.

• RFID spoofing: tento typ útoku se týká spoofingu signálů RFID za účelem získání dat uložených na štítku RFID. Poté útočník použije původní ID značky k odeslání vlastních dat, která se zdají být z původního zdroje, což útočníkovi umožňuje přístup k celému systému jako legální uzel.

• Klonování RFID: tento typ útoku cílí na RFID tag kopírováním jeho vlastních dat do jiného RFID tagu. Ačkoliv mají dva RFID tagy identická data, neduplikuje původní ID RFID.

• Neoprávněný přístup RFID: kvůli nedostatku vhodných mechanismů ověřování ve většině systémů RFID může k tagům přistupovat kdokoli. To automaticky znamená, že útočník může číst, upravovat nebo dokonce mazat data v uzlech RFID.

• Sinkhole útok: tento typ útoku se zaměřuje na důvěrnost dat a naruší síťové služby tím, že zlikviduje všechny pakety namísto jejich předávání do požadovaného cíle.

• Man-In-The-Middle (MITM) útok je provozován umístěním uzlu mezi dvěma komunikujícími uzly, které mu umožňují zachytit a sledovat veškerý provoz odesílaný mezi komunikujícími uzly. V závislosti na síťových komunikačních protokolech systému IoT nemusí být útočník fyzicky blízko sítě, aby mohl útok spustit.

• Útok na informace o směrování: informace o směrovací tabulce používá síťový směrovač k předávání dat do správných cílů. V důsledku toho tento typ útoku cílí na tuto tabulku spoofingem nebo změnou jejího obsahu, což narušuje síť a většina provozu bude zrušena a budou odeslány chybové zprávy.

3.4 Kryptografické útoky

Systém IoT spojuje všechny objekty prostřednictvím různých komunikačních kanálů. K ochraně komunikačního procesu se používají kryptografické algoritmy. Nic však není nerozbitné. Šifrovací útoky jsou zaměřeny na porušení šifrovací struktury používané v systému IoT. Mezi ně patří sada kryptografických útoků následující [10]:

• Útok postranním kanálem: tento útok cílí na šifrovací zařízení v systému IoT pomocí určitých technik k dosažení šifrovacích a dešifrovacích klíčů použitých v procesu šifrování dat.

• Útoky na dešifrování: pokud předpokládáme, že útočník již má šifrovaný nebo prostý text, cílem útočníka je najít šifrovací klíč porušením šifrovací struktury systému.

Existuje několik forem útoků na dešifrování, například vybraný šifrovací text, známý prostý text, pouze šifrovací text a vybraný prostý text.

• Men in the middle útok: aby dva uzly vzájemně komunikovaly na zabezpečeném komunikačním kanálu pomocí šifrovacího algoritmu, vyměňují si šifrovací a dešifrovací klíč. Útok MITM se pokouší získat přístup k těmto informacím zachycením signálů odeslaných mezi dvěma uzly a pokusí se provést výměnu klíčů s každým uzlem samostatně, což umožňuje útočníkovi šifrovat a dešifrovat jakékoli budoucí signály mezi komunikujícími uzly.

4. OCHRANA OSOBNÍCH ÚDAJŮ IOT

Růst popularity IoT nadále přidává na internet miliardy nových senzorů a zařízení a generuje obrovské množství informací o lidech, včetně jejich umístění, připojení, záznamů o nákupech, finančních transakcí, obrázků, hlasů, konverzací, zdravotního stavu atd. s nebo bez jejich souhlasu. Díky tomuto obrovskému množství informací je ochrana našeho soukromí obtížným úkolem.

Ochrana osobních údajů v systému IoT může mít mnoho podob, ale nejprve musíme definovat, co znamená soukromí. Podle Westina [11] je soukromí definováno jako „Požadavek jednotlivců, skupin nebo institucí, aby si sami určili, kdy, jak a do jaké míry jsou informace o nich sděleny ostatním“.

V kontextu internetu věcí se ochrana soukromí lidí stala velmi obtížným úkolem. Důvodem je, že proces sběru dat je pasivnější, všudypřítomný a méně rušivý, což vede k tomu, že uživatelé jsou méně informováni o tom, že jsou sledováni. Potenciální riziko ztráty kontroly nad osobními údaji je definováno jako ohrožení soukromí. Tato hrozba je obvykle jedním z klíčových zájmů uživatelů a má významný vliv na úroveň přijetí jakékoli nové technologie.

4.1 Ohrožení soukromí IoT

Jednou z důležitých charakteristik internetu věcí je schopnost objektů vnímat své prostředí. Tato schopnost však vede ke sledování a monitorování akcí a aktivit uživatelů, které narušují soukromí uživatelů a vedou k mnoha problémům, které mohou doslova vést k smrti. Tato část pojednává o běžných hrozbách pro ochranu soukromí v systému IoT [12].

Identifikace

Systém IoT umožňuje zařízením snímat a shromažďovat různé typy dat o uživatelích a jejich interakcích s prostředím. Tyto údaje se obvykle zpracovávají u poskytovatelů služeb, kteří jsou mimo kontrolu uživatelů.

Identifikace je hrozba spojování identifikátoru (např. jména, adresy) se soukromými údaji o jednotlivci. V IoT rozšiřují hrozbu identifikace nové technologie a propojení různých technik. Příkladem takových technik, kde je chování zákazníků zkoumáno pro účely analýzy a marketingu, je použití kamery pro sledování v jiných než bezpečnostních kontextech. K řešení tohoto problému se doporučuje ověřování na základě atributů, aby se minimalizovala data, která může zařízení shromažďovat v IoT, a udržovala se kontrola nad zveřejňováním dat [12].

Lokalizace a sledování

Lokalizace a sledování jsou hrozby určování a zaznamenávání polohy osoby v čase a prostoru pomocí různých prostředků, jako je například umístění mobilního telefonu, internetový provoz nebo data GPS. Dostupnost masivních a úplných prostorových a časoprostorových dat vedla k rostoucímu zájmu o využívání geografických dat a zahrnutí analýzy prostorových informací. Jak se systém IoT vyvíjí, je pravděpodobné, že několik faktorů zhorší lokalizační hrozby, jako je rozšíření a přesnost aplikací založených na poloze, všudypřítomnost technologie sběru dat a interakce se zařízeními IoT, která zaznamenávají identitu, umístění a aktivitu uživatele [12].

Profilování

Profilování je proces sběru a zpracování dat o akcích a akcích lidí po dlouhou dobu, aby je bylo možné klasifikovat podle určitých kritérií. Informace se obvykle shromažďují bez souhlasu uživatelů a integrují se s dalšími osobními údaji za účelem vytvoření úplnějšího profilu. Profilování se v současné době používá v široké škále oblastí, jako je elektronický obchod, cílená reklama a hodnocení úvěrů. Jedním z rizik spojených s profilováním je, že osobní údaje mohou být zpřístupněny dalším uživatelům, protože ostatní uživatelé, kteří používají stejný počítač a prohlížeč, mohou prohlížet cílené reklamy. Mnoho uživatelů se navíc obává pouhé realizace, že jsou sledováni. S rozmachem internetu věcí se počet dat ohromně zvyšuje kvůli explozivnímu růstu datových zdrojů a připojených zařízení. Kromě toho se data také kvalitativně změní, protože data jsou shromažďována z dříve nepřístupných částí soukromí lidí, například data shromážděná pomocí nositelných zařízení a různých zařízení doma [12].

4.2 Rešení pro ochranu IoT

Ochrana soukromí zařízení IoT by měla být jednou z hlavních priorit pro úspěšné přijetí a rozvoj systému IoT. K ochraně soukromí bylo navrženo několik přístupů. Tato část poskytuje krátkou diskuzi o těchto přístupech k řešení problému ochrany osobních údajů v systému IoT.

Privacy by Design: jedním z hlavních aspektů k zachování soukromí v prostředí IoT je privacy „by design“. Zákazníci IoT by měli mít požadované funkce, aby mohli ovládat své vlastní informace a definovat, kdo k nim má přístup. V současné době některé společnosti používají určitý druh dohody, který umožňuje určitým službám přistupovat k datům podle potřeby. Proto je nutné, aby byly integrované nástroje k ochraně soukromí uživatelů vytvořeny jako základní součást jakéhokoli produktu.

Povědomí o ochraně osobních údajů: Jedním z hlavních problémů narušení soukromí je nedostatečné povědomí veřejnosti. Uživatelé IoT si musí být plně vědomi toho, jak se chránit před jakýmikoli typy ohrožení soukromí.

Minimalizace dat: Poskytovatelé služeb IoT by měli využívat koncept minimalizace dat snížením sběru osobních údajů pouze na to, co souvisí se službou, kterou zavádějí. Musí také uchovat údaje, pouze pokud je potřebují pro službu.

Kryptografické techniky: Jedním z hlavních řešení pro zachování soukromí v zařízeních IoT je použití vhodné kryptografické techniky k šifrování dat. S omezeným úložištěm a výpočetními prostředky v zařízeních IoT však může být obtížné dosáhnout tohoto řešení.

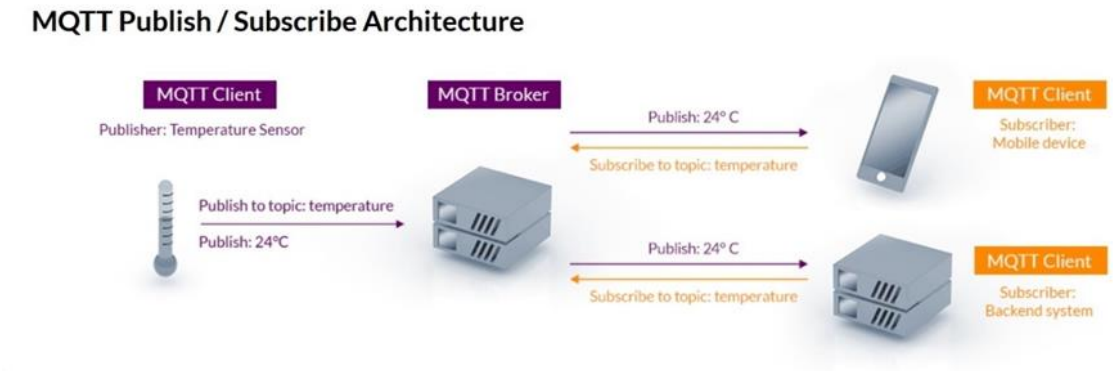
Anonymizace dat: Po sběru dat je nutné odstranit všechny jedinečné identifikátory, jako je číslo sociálního zabezpečení a čísla řidičských průkazů, z datových záznamů, aby se odstranila totožnost jednotlivců v databázích.

Řízení přístupu: Jedním z řešení pro zachování soukromí uživatelů IoT je poskytnutí efektivního modelu řízení přístupu pro systém IoT, který umožní inteligentním věcem poskytovat jemnozrnná rozhodnutí [1].

5. ZPŮSOBY OCHRANY KOMUNIKACE IOT

5.1 MQTT

MQTT je standardní protokol pro zasílání zpráv OASIS pro internet věcí (IoT). Je navržen jako extrémně lehký přenos zpráv pro publikování/odběr, který je ideální pro připojení vzdálených zařízení s malou stopou kódu a minimální šířkou pásma sítě. MQTT se dnes používá v nejrůznějších průmyslových odvětvích, jako je automobilový průmysl, výroba, telekomunikace atd. Klienti MQTT vyžadují minimální zdroje, takže je lze použít na malých mikrokontrolerech. Záhlaví zpráv MQTT jsou malá k optimalizaci šířky pásma sítě. MQTT umožňuje zasílání zpráv mezi zařízeními na cloud a cloud na zařízení. To umožňuje snadné vysílání zpráv skupinám věcí. MQTT se může škálovat a spojit s miliony zařízení IoT. Spolehlivost doručování zpráv je důležitá pro mnoho případů použití IoT. To je důvod, proč má MQTT 3 definované úrovně kvality služeb: 0 - maximálně jednou, 1- alespoň jednou, 2 - přesně jednou. Mnoho zařízení IoT se připojuje přes nespolehlivé mobilní sítě. Podpora MQTT pro trvalé relace zkracuje čas pro opětovné připojení klienta k brokeru [13]. Obrázek 5-1 ukazuje architekturu IoT (převzato z [13]).



obr. 5-1: Architektura MQTT

Hlavní charakteristiky protokolu MQTT[22]:

- Snadná implementace
- Poskytování kvality dat a poskytování služeb
- Efektivní šířka pásma
- Průběžné povědomí o relaci

Hlavní součásti protokolu MQTT:

- MQTT broker
- MQTT publisher
- MQTT subscriber

MQTT broker

MQTT broker je centrální rozbočovač (obvykle v cloudu na veřejném internetu), který spojuje publishera MQTT s subscriberem MQTT. Publisher MQTT odesílá zprávy a subscribery MQTT se přihlašují k odběru zpráv. Může být několik subscriberů MQTT stejného „tématu“.

Zprávy jsou rozděleny do „témat“; zařízení může dané téma buď „publikovat“, nebo se k němu přihlásit. V rámci tématu jsou zprávy vyměňovány tak, jak jsou přijímány brokerem MQTT a poté zasílány na předplacená zařízení.

Zařízení může být současně publisherem pro některá témata (publikuje naměřené hodnoty) a subscriberem pro jiná témata (reaguje na příkazy pro ovládání výstupu) [14].

MQTT publisher

MQTT publisher odesílá zprávy MQTT brokeru MQTT. Klient MQTT může publikovat zprávy, pokud je připojen k brokeru MQTT. Protokol MQTT kategorizuje zprávy podle tématu. Každá zpráva musí obsahovat téma, které může broker MQTT použít k předání zprávy subcriberům MQTT. Každá zpráva má užitečné zatížení, které je doručeno subscriberům tímto způsobem. Může nést jakýkoli obsah [14].

MQTT subscriber

MQTT subscriber přijímá zprávy MQTT od brokera MQTT. Zprávy jsou rozděleny do témat, k jejichž odběru je možné se přihlásit [14].

5.2 MQTT system s AES

MQTT není zpočátku chráněn proto je pro zabezpečení přenosu zpráv mezi publisherem a subscriberem nutné používat kryptografické protokoly.

Šifrování poskytuje zabezpečení na aplikační vrstvě. Šifrování se používá, když vývojář nepoužívá zabezpečení TLS, ale přesto nechce odesílat data jako prostý text. To poskytuje další vrstvu zabezpečení, protože i tímto způsobem jsou všechna data aplikace zabezpečena a šifrována.

Existují různé režimy blokových šifer například cbc, cfb, ofb, ecb, ctr atd.

Electronic codebook (ECB)

Nejjednodušší z režimů šifrování je režim ECB. Zpráva je rozdělena do bloků a každý blok je šifrován samostatně.

Nevýhodou této metody je nedostatečná difúze. Protože ECB šifruje identické bloky prostého textu do identických bloků šifrovaného textu, neskrývá dobře datové vzory. nedoporučuje se ECB používat v kryptografických protokolech [19].

Cipher block chaining (CBC)

V režimu CBC je každý blok prostého textu před šifrováním XORován s předchozím blokem šifrovaného textu. Tímto způsobem každý blok šifrovacího textu závisí na všech blocích prostého textu zpracovaných až do tohoto bodu. Aby byla každá zpráva jedinečná, musí být v prvním bloku použit inicializační vektor.

CBC je nejčastěji používaným režimem provozu. Jeho hlavní nevýhody spočívají v tom, že šifrování je sekvenční (tj. nelze jej paralelizovat) a že zpráva musí být vyplněna na násobek velikosti šifrovacího bloku. Jedním ze způsobů řešení tohoto posledního problému je metoda známá jako krádež šifrovacího textu. Jednobitová změna v prostém textu nebo inicializačním vektoru (IV) ovlivní všechny následující bloky šifrovaného textu [19].

Cipher feedback (CFB)

Režim CFB ve své nejjednodušší variantě využívá celý výstup blokové šifry. V této variantě je velmi podobný CBC, vytváří blokovou šifru do samosynchronizující proudové šifry. CFB dešifrování v této variantě je téměř identické s CBC šifrováním prováděným v opačném pořadí [19].

Output feedback (OFB)

Režim OFB dělá blokovou šifru do synchronní proudové šifry. Generuje bloky klíčového proudu, které se pak XORují s bloky prostého textu, aby získaly šifrovací text. Stejně jako u jiných proudových šifer, i převrácení bitů v ciphertextu vytvoří převrácený bit v prostém textu na stejném místě. Tato vlastnost umožňuje mnoha kódům opravujícím chyby fungovat normálně, i když jsou použity před šifrováním.

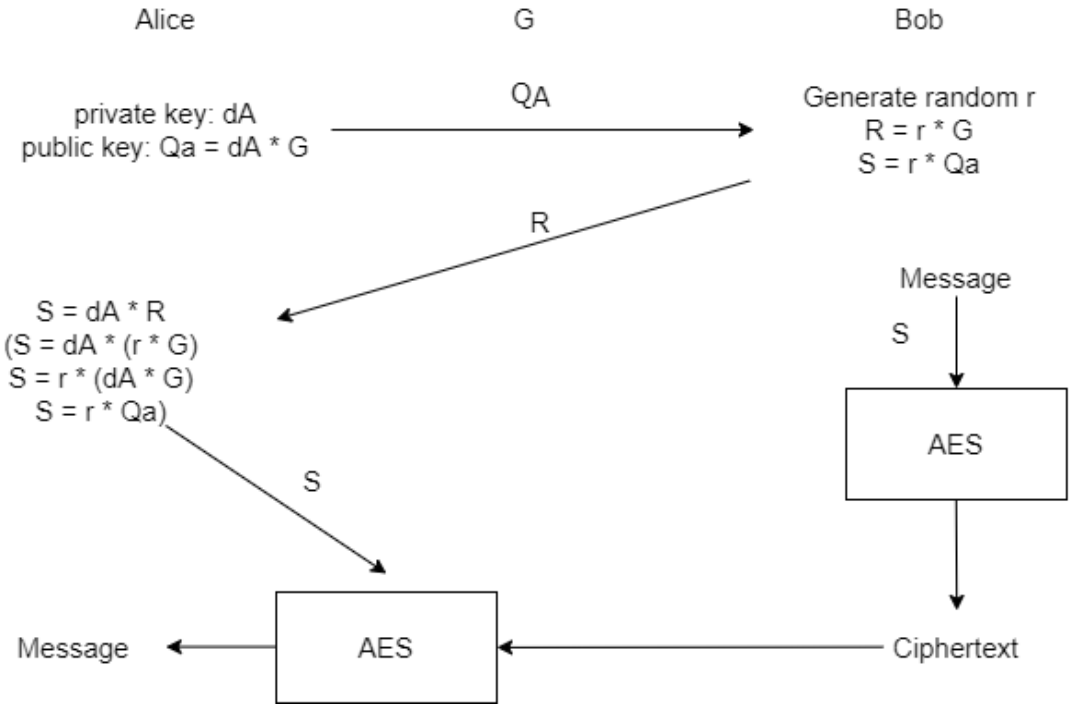
Kvůli symetrii operace XOR jsou šifrování a dešifrování přesně stejné [19].

Counter (CTR)

Režim CTR má podobné vlastnosti jako OFB, ale také umožňuje vlastnost náhodného přístupu během dešifrování. Režim CTR je vhodný pro provoz na víceprocesorovém stroji, kde lze blokovat paralelní šifrování bloků. Kromě toho nemá problém s krátkým cyklem, který může ovlivnit OFB [19].

5.3 ECIES

ECIES je schéma šifrování veřejného klíče založené na eliptických křivkách. ECIES používá veřejný klíč z ECC k odvození symetrického klíče. Dále bude vytvořen symetrický klíč z Elliptic Curve Cryptography a poté bude použít k šifrování pomocí AES ve vybraném módu. Na obrázku 5-2 [20] je schéma, ukazující komunikaci mezi Bobem a Alicí, kde Bob posílá zprávu Alici [21].



obr. 5-2: Schéma ECIES

V této metodě Alice vygeneruje náhodný soukromý klíč (d_A), vezme bod na eliptické křivce (G) a poté určí její veřejný klíč (Q_A): $Q_A = d_A \times G$
G a Q_A jsou tedy body na eliptické křivce. Alice pak odešle Q_A Bobovi. Dále Bob vygeneruje: $R = r \times G$; $S = r \times Q_A$,
kde r je náhodné číslo generované Bobem. Symetrický klíč (S) se poté použije k zašifrování zprávy.

Alice poté obdrží zašifrovanou zprávu spolu s R. Je pak schopna určit stejný šifrovací klíč pomocí : $S = d_A \times R$, který je stejný jako vygenerovány Bobem klíč [20].

Srovnání s jinými algoritmy

Zabezpečení ECIES je založeno na výpočetní složitosti ECDLP. Kryptografické algoritmy mohou být také založeny na výpočetní složitosti faktorizačních problémů (příklad algoritmu: RSA) a diskrétním logaritmu (schéma El Gamal). ECDLP je však považován za nejtěžší ze tří, takže má za následek důležitou výhodu ECIES: menší velikost klíče, což je důležité pro zařízení IoT. Tabulka 5-1 ukazuje tyto rozdíly [21].

Tabulka 5-1: Srovnání velikosti klíčů ECIES a RSA

Úroveň zabezpečení (bit)	Délka klíče RSA (bity)	Délka klíče ECIES (bity)
80	1024	160-223
112	2048	224-255
128	3072	256-283
192	7680	384-511
256	15360	512-571

5.4 Rabin kryptosystém

Kryptosystém Rabin je technika kryptosystému veřejného klíče, jejíž bezpečnost, stejně jako bezpečnost kryptosystému RSA, souvisí s obtížností celočíselné faktorizace. Tento kryptosystém navrhl v roce 1979 Michael Rabin jako variantu kryptosystému RSA, pro který má faktorizace modulo *n* téměř stejnou výpočetní složitost jako získání dešifrovací transformace ze šifrovací transformace. Výhodou je, že bylo prokázáno, že dešifrování Rabinova kryptosystému je stejně obtížné jako celočíselná faktorizace. Nevýhodou je, že každý výstup Rabinova algoritmu může být generován čtyřmi možnými vstupy. Každý výstup, který je blokem šifrovacího textu, je tedy v dešifrovací proceduře představen čtyřem možným vstupům, které představují blok původního prostého textu [23].

Postup navrhování kryptosystému

- Generování klíčů
- Šifrování
- Dešifrování

Generování klíčů

1. Vytvořte dvě velká prvočísla, *p* a *q*, která splňují podmínku:
   $p \neq q, p \equiv q \equiv 3 \pmod 4$,
2. Vypočítejte hodnotu *n:* $n = p * q$
3. Publikujte *n* jako veřejný klíč a uložte *p* a *q* jako soukromý klíč.

Šifrování

1. Vezmete veřejný klíč *n*.
2. Převeďte zprávu na hodnotu ASCII. Poté jej převeďte na binární a vynásobte stejnou binární hodnotou a změňte binární hodnotu zpět na desítkové m.
3. Šifrování pomocí: $C = m^2 \bmod n$
4. Zašlete *C* příjemci.

Dešifrování

1. Přijměte *C* od odesílatele.
2. Určete *a* a *b* pomocí rozšířeného euklidovského algoritmu GCD tak, že
   $a*p + b*q = 1$
3. Vypočítejte *r* a *s:*
   $r = C^{(p+1)/4} \bmod p$
   $s = C^{(q+1)/4} \bmod q$
4. Nyní vypočítáme *X* a *Y*.
   $X = (a*p*r + b*q*s) \bmod p$
   $Y = (a*p*r - b*q*s) \bmod q$
5. Čtyři kořeny jsou, $m1 = X, m2 = -X, m3 = Y, m4 = -Y$
6. Nyní je převeďte na binární a rozdělte je na polovinu.
7. Určete, která levá a pravá polovina jsou stejné. Ponechte tu binární polovinu a převeďte ji na desítkové *m*. Získejte znak ASCII pro desítkovou hodnotu *m*. Výsledný znak dává správnou zprávu odeslanou odesílatelem.

5.5 SSL a TLS

SSL je protokol, resp. vrstva vložená mezi vrstvu transportní a aplikační, která poskytuje zabezpečení komunikace šifrováním a autentizaci komunikujících stran. Následovníkem SSL je protokol TLS.

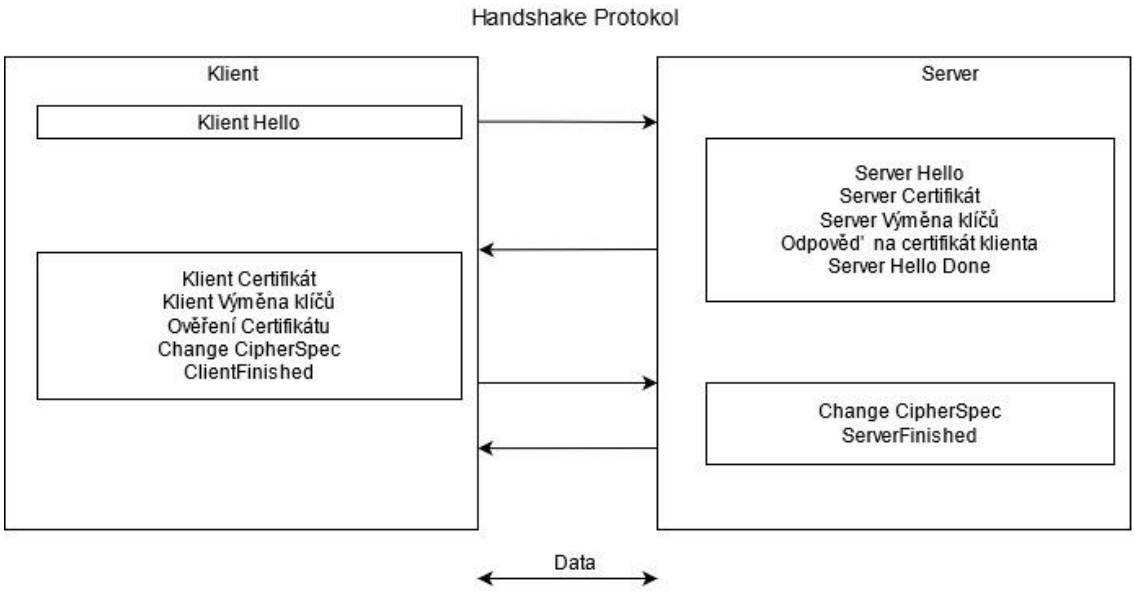
SSL je navržen tak, aby usnadnil komunikační kanál mezi dvěma vrstevníky, poskytuje mechanismy pro bezpečnou výměnu klíčů, autentizaci, šifrování a integritu. Jeho cílem je být odolný vůči útokům typu man-in-the-middle, odposlechu, útokům replay a statistickým útokům. Zatímco SSL může ověřit dvě strany konverzace, v praxi se obvykle autentizuje pouze server. Cíle SSL/TLS zahrnují nejen zabezpečení, ale také interoperabilitu, rozšiřitelnost a relativní účinnost.

SSL / TLS se skládá ze dvou vrstev: záznamové vrstvy a handshake vrstvy. Záznamová vrstva bere data poskytovaná aplikací vyšší vrstvy, fragmentuje data do spravovatelných bloků a provádí kompresi, šifrování pomocí symetrických klíčů a generování MAC. Vrstva handshake provádí zřízení relace a vyjednávání možností, přičemž určuje persistentní symetrické klíče, které hromadně používá vrstva záznamu.

SSL / TLS běží nad protokolem TCP / IP, který řídí přenos a směrování dat přes internet, a pod protokoly vyšší úrovně, jako je HTTP, které pomocí protokolu TCP / IP podporují typické úkoly aplikací, jako je stahování webových stránek. SSL umožňuje serveru s povoleným SSL ověřit se na klienta s povoleným SSL a naopak a umožňuje oběma strojům navázat šifrované připojení. SSL po celou dobu používá protokol TCP / IP jménem protokolů vyšší úrovně [24].

Handshake mezi klientem a serverem v SSL / TLS funguje následovně [25]:

1. Klient odešle zprávu clientHello na server a náhodnou hodnotu klienta a podporované šifrovací sady.
2. Server odešle klientovi zprávu serverHello a náhodnou hodnotu serveru.
3. Server odešle svůj certifikát klientovi k ověření a může požadovat certifikát od klienta.
4. Server odešle zprávu serverHelloDone.
5. Pokud server požadoval certifikát od klienta, klient ho odešle.
6. Klient vytvoří náhodné tajemství Pre-Master a zašifruje ho veřejným klíčem z certifikátu serveru.
7. Klient odešle zašifrované tajemství Pre-Master na server.
8. Server i klient vygenerují hlavní tajemství Master Secret a klíče relace na základě Pre-Master.
9. Klient odešle oznámení ChangeCipherSpec na server, aby začal používat klíče relace pro hašování a šifrování dat.
10. Klient odešle zprávu clientFinished.
11. Server získá ChangeCipherSpec a přepne na symetrické šifrování pomocí klíče relace.
12. Server odešle zprávu serverFinished.
13. Klient a server si nyní mohou vyměňovat data aplikací pomocí symetrického šifrovacího a relačního klíče.



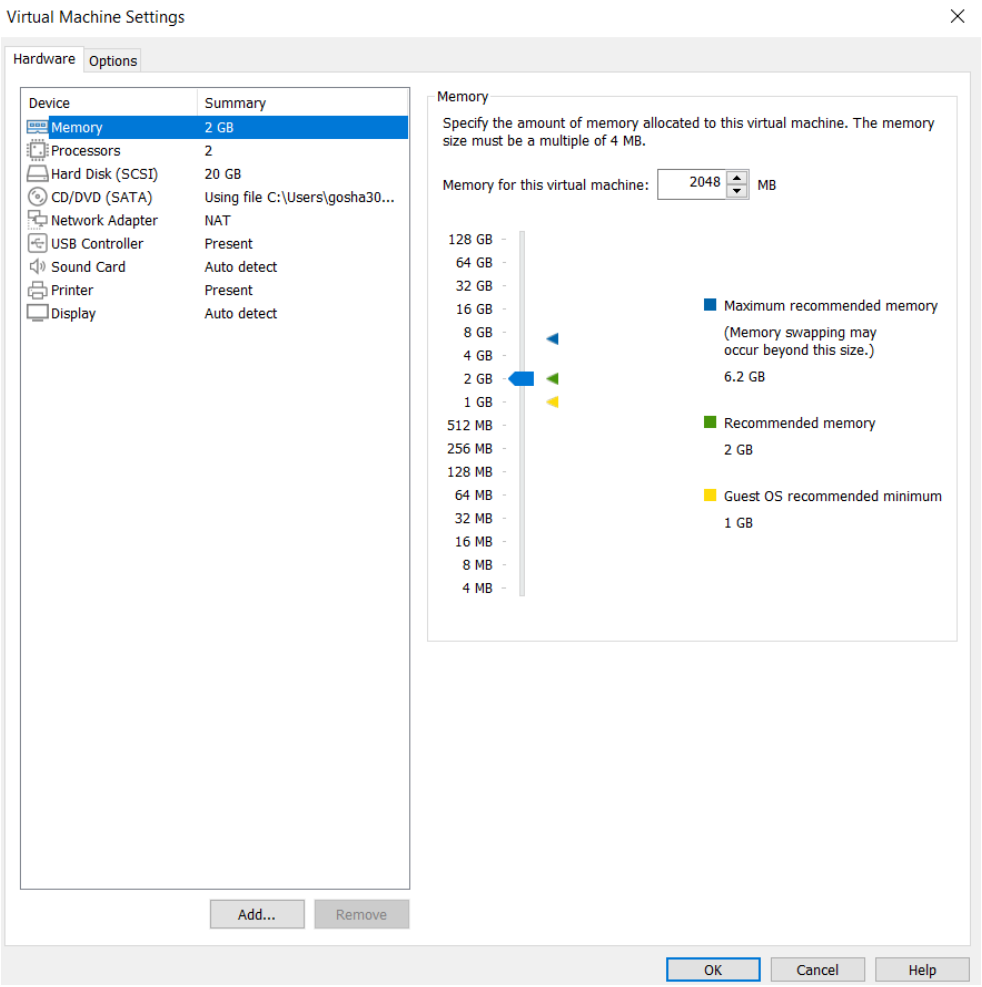
obr. 5-3 Handshake protokol SSL/TLS

6. TESTOVÁCÍ IMPLEMENTACE MOSQUITTO A TESTOVÁNÍ KNIHOVEN

Praktická část této práce bude sestávat z několika částí. První část bude obsahovat implementace jednoduchého publishera a subscriberu, kteří budou navzájem komunikovat. Druhá část bude také obsahovat publishera a subscribera, ale zprávy, které si navzájem posílají, budou šifrovány pomocí AES. Poslední část bude obsahovat popis fungování knihovny ecies.

6.1 Jednoduchý MQTT publisher

Obrázek 6-1 ukazuje nastavení virtuálního stroje, který budeme používat v rámci semestrální práci. Jako software byl použit Ubuntu-20-10.



obr. 6-1: Nastavení virtualního prostředí

Byla použitá knihovna „Mosquitto”, která bude sloužit jako broker. Mosquitto je služba zprostředkovatele zpráv MQTT s otevřeným zdrojovým kódem. Využívá protokol MQTT pro komunikaci zařízení pomocí odesílání a přijímání zpráv.

Mosquitto je založeno na Eclipse, což je lehká serverová implementace protokolu MQTT. Senzory jsou zdrojem a cílem zpráv MQTT. Mosquitto je most, který se připojuje k dalším serverům pro zasílání zpráv, založeným na MQTT. Most má funkce předávání zpráv MQTT ze zdroje do cíle. Aplikace založená na Mosquittu se skládá z komponent připojujících se k serveru pro zasílání zpráv, přihlašování k tématu a publikování k tématu [17].

Často používané funkce [18]:

- mosquitto_lib_init() – inicializace knihovny, musí být voláno před jinými funkcemi Mosquitto.
- mosquitto_connect – připojení k brokeru MQTT.
- mosquitto_new – vytvoření nové instance klienta mosquitto.
- mosquitto_destroy – slouží k uvolnění paměti spojené s instancí klienta mosquitto.
- mosquitto_disconnect – odpojení od brokera.
- mosquitto_lib_cleanup – slouží k uvolnění paměti spojené s knihovnou.
- mosquitto_publish – publikuje zprávu na daná témata.
- mosquitto_subscribe – přihlašení k tématu.
- mosquitto_connect_callback_set – nastavení zpětného volání pro připojení.
- mosquitto_loop_start – toto je část klientského rozhraní.
- mosquitto_loop_stop – toto je část klientského rozhraní.

Byl napsán jednoduchý MQTT publisher v jazyce C ve virtuálním prostředí Linux pro ukázku protokolu MQTT(Příloha 1). Tento publisher bude posílat zprávu „Hello“ pokud bude navázáno spojení s brokerem, v opačném případě vypíše error zprávu. Po napsání kódu publishera musí být program zkompilován pomocí (příkazu 1).

Příkaz 1

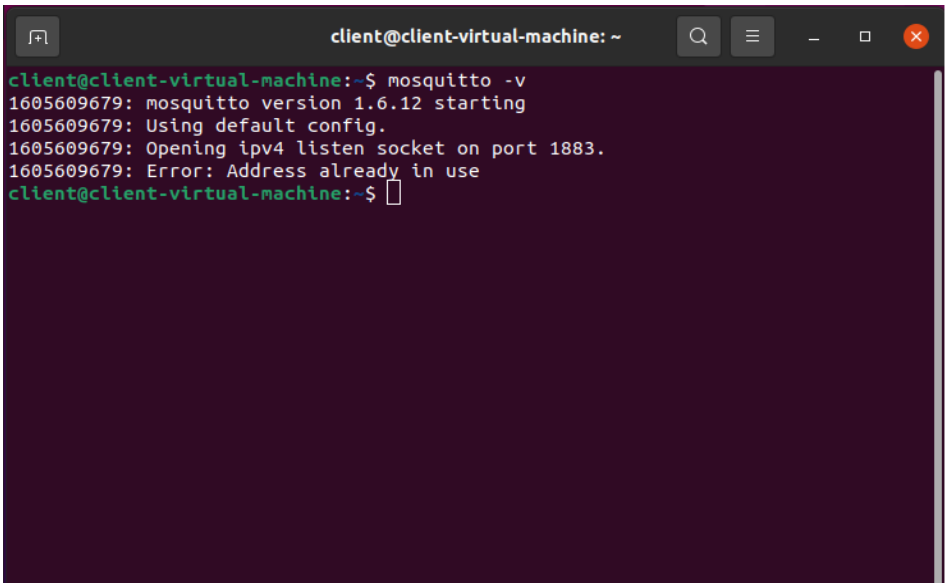
```
gcc mqtt_pub.c -o mqtt_pub -lmosquitto
```

Broker byl povolen v příkazovém řádku ve virtuálním počítači pomocí (příkazu 2).

Příkaz 2

```
mosquitto -v
```

Existuje příležitost narazit na problém, když je IP adresa rezervována něčím jiným (obr. 6-2).



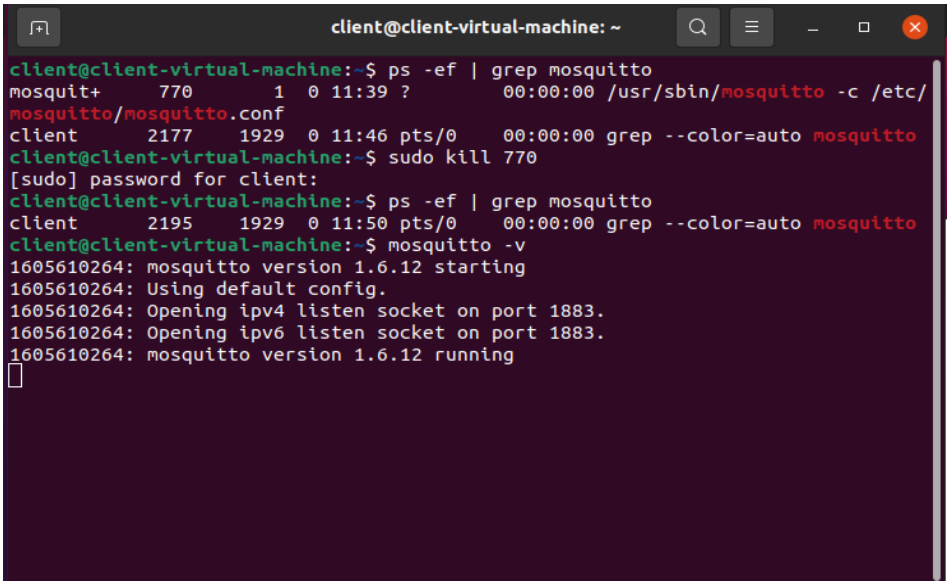
obr. 6-2: Problém s IP adresou

Tento problém lze vyřešit pomocí (příkaz 3).

Příkaz 3

```
ps -ef | grep mosquito
sudo kill { id processu }
```

Připravený broker je na obrázku 6-3.



obr. 6-3: Připravený broker

V novém příkazovém řádku pomocí (příkaz 4) bude spuštěn subscriber.

Příkaz 4

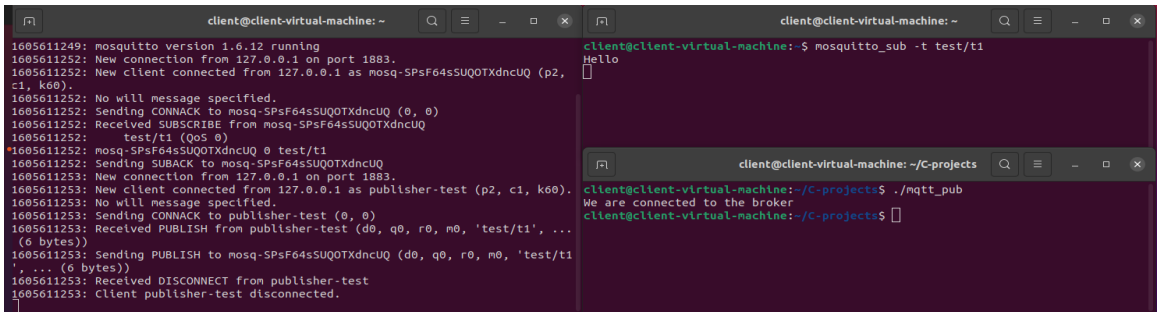
```
mosquitto_sub -t test/t
```

Ve třetím příkazovém řádku bude spuštěn (příkaz 5), který spustí publishera.

Příkaz 5

```
./mqtt_pub
```

Po použití všech předchozích příkazů máme připraveného publishera (obr. 6-4)



obr. 6-4: Připravená komunikace mezi publisherem a subscriberem 1

6.2 Jednoduchý MQTT subscriber

Byl napsán jednoduchý MQTT subscriber v jazyce C (Příloha 2). Po napsání kódu publishera musí být program zkompilován pomocí (příkaz 6).

Příkaz 6

```
gcc mqtt_sub.c -o sub_test -lmosquitto
```

Spustíme subscribera pomocí (příkaz 7).

Příkaz 7

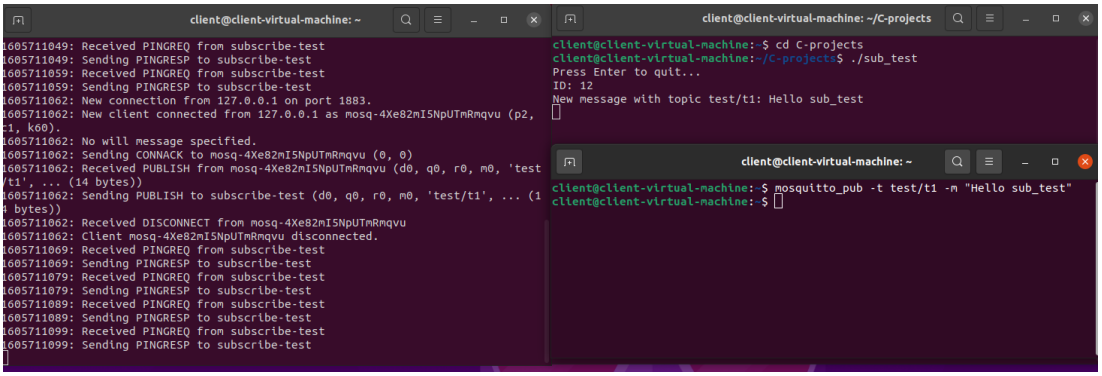
```
./sub_test
```

Stejně jako v bodě 6.1 spustíme MQTT broker (příkaz 2). Ve třetím terminálu bude fungovat publisher, který bude posílat zprávu „Hello_sub“ a bude spuštěn pomocí (příkazu 8).

Příkaz 8

```
mosquitto_pub -t test/t1 -m "Hello_sub"
```

Po použití všech předchozích příkazů máme připraveného subscribera(Obrázek 6-5).



obr. 6-5: Připravená komunikace mezi publisherem a subscriberem 2

6.3 MQTT zabezpečený pomocí AES

V této částí bude vysvětleno, jak lze šifrování aplikovat na užitečný obsah MQTT. Bude použito šifrování typu AES s různými módy.

Byl napsán MQTT publisher v jazyce C, uvnitř kterého byly naimplementovany metody umožňující posílat šifrované zprávy (Příloha 3). Následně byl zkompilovan pomocí (příkaz 9).

Příkaz 9

```
gcc -o enc publisher.c -lmosquitto -lmcrypt
```

Dále byl napsán MQTT subscriber, který má metody umožňující mu dešifrovat zprávy od publishera (Příloha 4). Následně byl zkompilován pomocí (příkaz 10).

Příkaz 10

```
gcc -o enc2 subscriber.c -lmosquitto -lmcrypt
```

Spustíme MQTT broker (příkaz 2). Dále spustíme subscribera pomocí (příkaz 11) a publishera (příkaz 12).

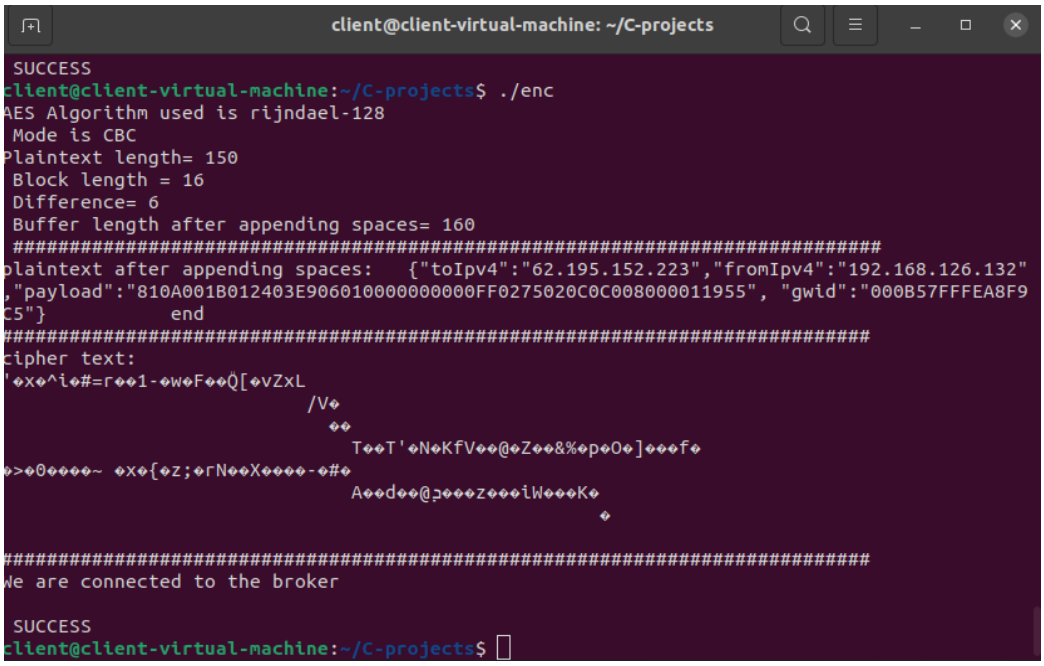
Příkaz 11

```
./enc2
```

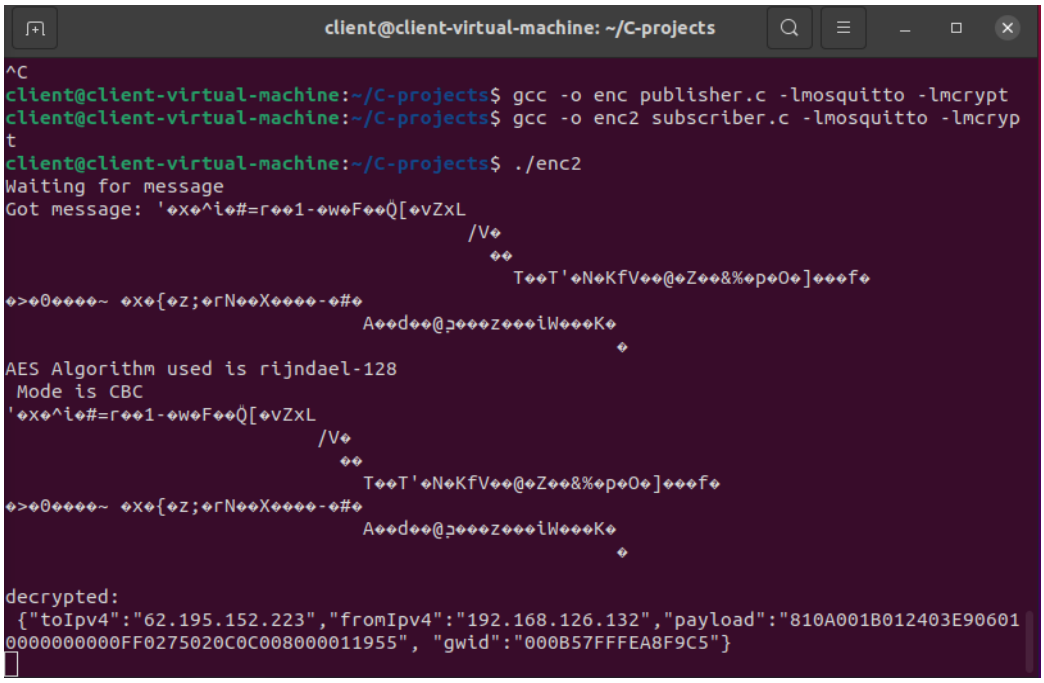
Příkaz 12

```
./enc
```

Na obrázcích 6-6 a 6-7 jsou výsledná komunikace pubslishera a subscribera. Po spuštění publisher zašifruje zprávu, kterou chce odeslat subscriberu, zobrazí plaintext a ciphertext, napíše, že spojení s brokerem bylo úspěšné a pokud během celého procesu nedošlo k chybě, vypíše „SUCCESS“. Subscriber čeká na zprávy v kanálu, zpočátku obdrží zašifrovanou zprávu, kterou pak dešifruje pomocí tajného klíče. Poté pokračuje v čekání na novou zprávu v kanálu.



obr. 6-6: Komunikace ze strany publishera



obr. 6-7: Komunikace ze strany subscribera

Byla provedena analýza různých režimů blokových šifer. Výsledky jsou uvedeny v tabulce 6-1. Lze pozorovat, že je s menší delkou klíče doba trvání šifrování a dešifrování nejmenší. S růstem hodnoty velikosti klíče se čas pro tyto operace zvyšuje.

Tabulka 6-1 Doba trvání šifrování a dešifrování pří různých modéch

Velikost klíče (b)	Mód	Šifrování (ms)	Dešifrování (ms)
128	CBC	1	< 1
192		2	1
256		3	1
128	ECB	2	< 1
256		2	1
128	OFB	1	< 1
256		2	< 1
128	CFB	2	< 1
256		3	< 1
128	CTR	2	< 1
256		2	1

6.4 Šifrování a dešifrování pomocí knihovny eciespy

Knihovna Python eciespy používá ECC přes křivku secp256k1 + šifrování AES-256 v módu GCM.

S ohledem na možné další využití knihovny eciespy v bakalářské práci byly v rámci semestrální práce nastudovány hlavní funkce této knihovny.

Tato část bude popisovat šifrování a dešifrování zpráv. Tabulka 6-3 obsahuje význam všech příkazů, které budou použity v této části.

Tabulka 6-2: Příkazy v eciespy

-g	Vytvoření pár klíčů
-e	Šifrování pomocí veřejného klíče
-d	Dešifrovat pomocí soukromého klíče
-k	Soubor veřejného nebo soukromého klíče
-D	Soubor k šifrování nebo dešifrování
-O	Šifrovaný nebo dešifrovaný soubor

Na začátku byly vygenerovány soukromý a veřejný klíče pomocí (příkaz 13). Na obrázku 6-8 je výsledek tohoto příkazu.

Příkaz 13

```
eciespy -g
```

```
client@client-virtual-machine:~$ eciespy -g
Private: 0xb5fa3cb9a2e588047247dbfa70aae5a4338f655b3aec764eb9edaddcc9360d47
Public: 0xdd3525bc3c081a1860c52a87f7ec290e599706d48742c650b2d5d153d47409edc70fac
65b10ca9461975da9e04e124e983ff7f9df5555ea8bb509c53925cb080
Address: 0x36981bD8F6bBa056b13cb6ACCF0F722573d435c8
```

obr. 6-8: Generování soukromého a veřejného klíčů

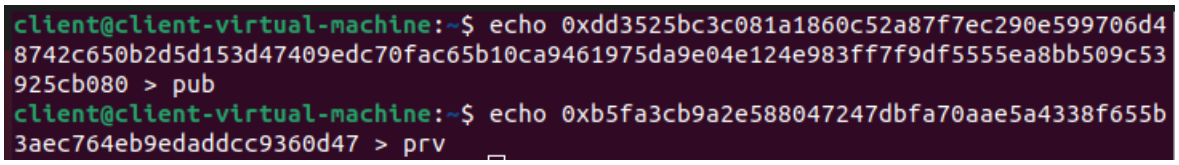
Poté bude veřejný klíč zapsán do souboru s názvem pub pomocí (příkaz 14) a soukromý klíč bude zapsán do souboru s názvem prv pomocí (příkaz 15). Na obrázku 6-9 jsou výsledky těchto příkazu.

Příkaz 14

```
echo
0xdd3525bc3c081a1860c52a87f7ec290e599706d48742c650b2d5d153d
47409edc70fac65b10ca9461975da9e04e124e983ff7f9df5555ea8bb50
9c53925cb080 > pub
```

Příkaz 15

```
echo
0xb5fa3cb9a2e588047247dbfa70aae5a4338f655b3aec764eb9edaddcc
9360d47 > prv
```



obr. 6-9: Zapsání klíčů do souborů

Dále bude zpráva, která má být zašifrována, zapsána do souboru s názvem data (příkaz 16).

Příkaz 16

```
echo 'data to encrypt' > data
```

Poté bude soubor „data“ zašifrován veřejným klíčem pomocí příkazu 17.

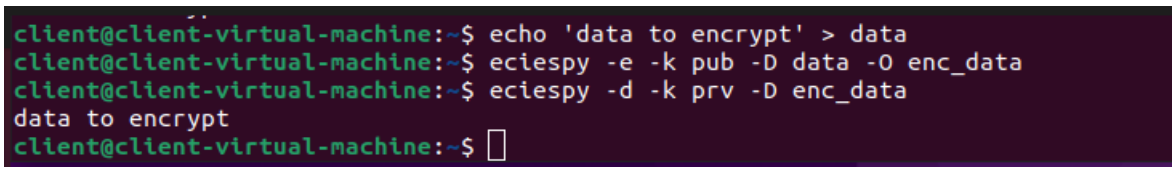
Příkaz 17

```
eciespy -e -k pub -D data -O enc_data
```

Pro dešifrování bude použit příkaz 18. Obrázek 6-10 ukazuje konečný výsledek dešifrování souboru.

Příkaz 18

```
eciespy -d -k prv -D enc_data
```

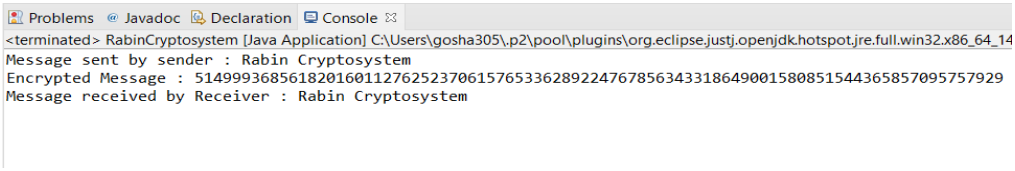


obr. 6-10: Výsledek dešifrování souboru

6.5 Rabin kryptosystém

V práci [22] kryptosystém Rabin se používá jako součást velkého systému k šifrování datového bloku na straně publishera a také k dešifrování na straně subscribera. V této části bude popsán šifrovací a dešifrovací algoritmus v kryptosystému Rabin.

Kód (Příloha 7) byl převzat z [26] a demonstruje jak funguje Rabinův kryptosystém. Celý algoritmus byl popsán dříve v teoretické části, přesněji v 5.4. Obrázek 6-11 ukazuje, že tento algoritmus opravdu funguje.



obr. 6-11 Šifrování a dešifrování pomocí kryptosystému Rabin

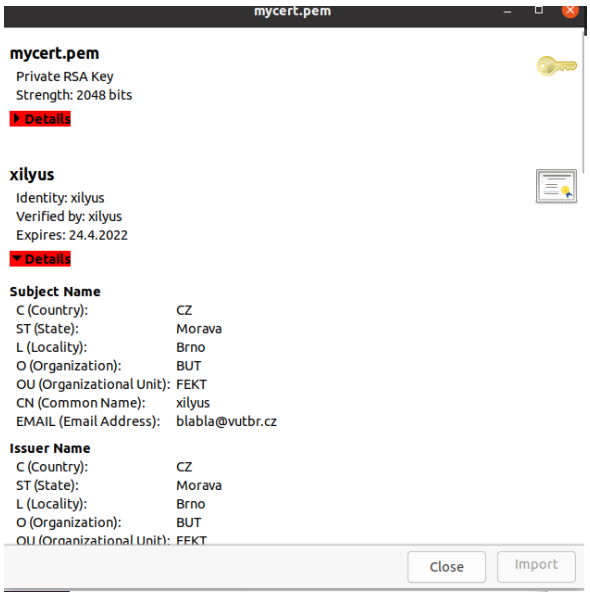
6.6 TLS 1.2 komunikace

V této části bude popsán způsob komunikace mezi serverem a klientem pomocí protokolu TLS a výměny certifikátů. Veškerá práce bude probíhat ve virtuálním stroji, jehož základní parametry jsou na obrázku 6-1.

Prvním krokem je vygenerování certifikátu podepsaného svým držitelem, pro který byl použit příkaz 19. Základní parametry certifikátu jsou následující: certifikát je platný po dobu jednoho roku, klíč RSA je 2048 bitů, uložen v souboru mycert.pem. Informace o certifikátu jsou na obrázku 6-12. Kódy SSL klienta a SSL servera jsou v příloze 5 a 6.

Příkaz 19

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
keyout mycert.pem -out mycert.pem
```



obr. 6-12 Vytvořený certifikát

Nejprve byl zkompilován ssl klient (příkaz 20) a ssl server (příkaz 21).

Příkaz 20

```
gcc -Wall -o client Client.c -L/usr/lib -lssl -lcrypto
```

Příkaz 21

```
gcc -Wall -o server Server.c -L/usr/lib -lssl -lcrypto
```

Poté by měl být nejprve zapnut server (příkaz 22) a teprve poté klient (příkaz 23). Server naslouchá na portu 8081 a klient z localhostu odešle zprávu serveru na stejném portu.

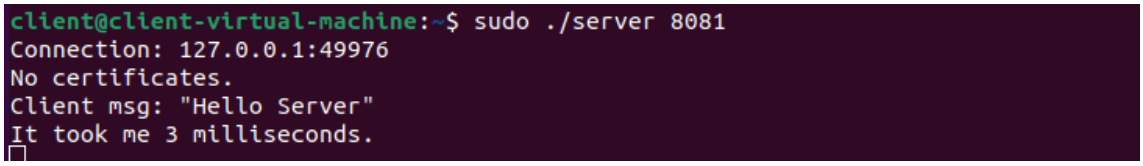
Příkaz 22

```
sudo ./server 8081
```

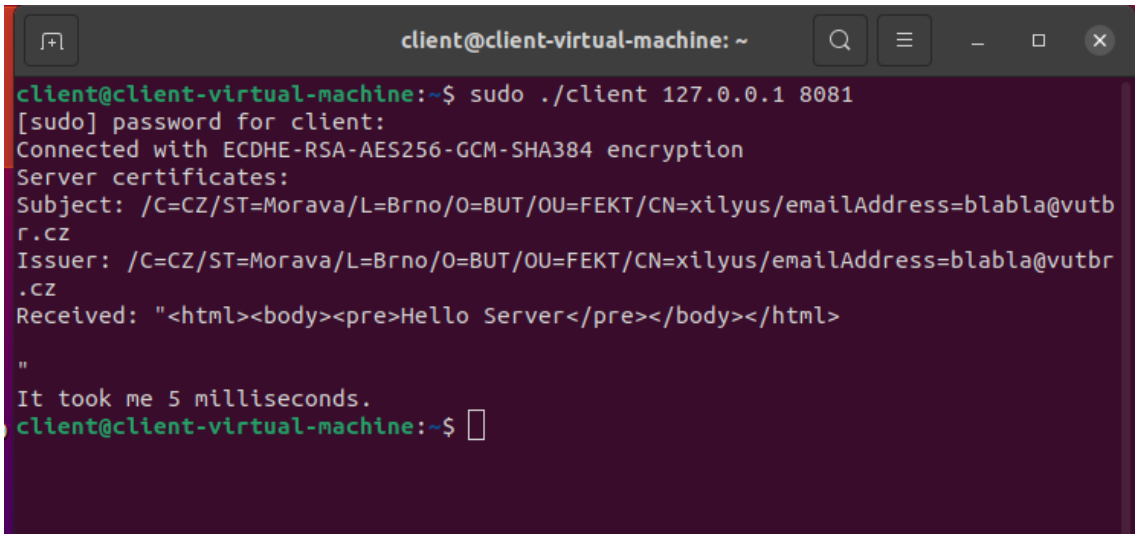
Příkaz 23

```
sudo ./client 127.0.0.1 8081
```

Na obrazcích 6-13 a 6-14 jsou výsledky komunikace mezi klientem a serverem. Lze vidět, že server přijal připojení z adresy 127.0.0.1 a přijal zprávu klienta “Hello Server”. Klient vidí certifikát serveru, všechny jeho základní parametry a to, že jeho zpráva byla serverem přijata.



obr. 6-13 SSL server



obr. 6-14 SSL klient

6.7 Srovnání MQTT s AES a TLS 1.2 komunikace

V této části bude provedeno srovnání dvou komunikačních metod. A zejména MQTT s AES a TLS 1.2. Pro přesnější analýzu bude srovnání probíhat ve stejném OS, jehož základní parametry jsou na obrázku 6-1.

Hlavním účelem srovnání bude změřit čas, který potřebují komunikující strany pro navázání spojení. Měření pro MQTT s AES byly provedeny v 6.3. Pro snadné srovnání bude vzat průměr všech dob trvání šifrování a dešifrování pro klíče 128 a 256 bitů všech modů.

Tabulka 6-3 Průměrná doba trvání šifrování a dešifrování MQTT s AES

Velikost klíče (b)	Šifrování (ms)	Dešifrováni (ms)
128	1,6	< 1
256	2,4	1

Na obrázku 6-14 je posledním řádkem ve výpisu klienta čas, který klient použil pro odesílání zprávy na server a se rovná 5 ms. V tabulce 6-4 je pro 256-bitový klíč čas potřebný k odeslání zprávy brokeru 2,4 ms, což je dvakrát lepší čas než u komunikace SSL / TLS. Pokud jde o přijetí zprávy, buď subscriberem nebo serverem, je tady také rozdíl v době potřebné k přijetí a zpracování zprávy. Na obrázku 6-13 lze vidět, že server SSL / TLS potřebuje k přijetí zprávy 3 ms a z tabulky 6-4 vidíme, že komunikace MQTT potřebuje 1 ms nebo méně. Z toho lze vyvodit závěr, že komunikace na straně serveru / subscriberu u MQTT je třikrát rychlejší.

7. ŘEŠENÍ PRO IMPLEMENTACI ZABEZPEČENÉ KOMUNIKACE MEZI MIKROPOČÍTAČI POMOCÍ MQTT

V této části bude provedena implementace zabezpečené komunikace mezi dvěma mikropočítači pomocí MQTT. Jako mikropočítače budou použita dvě zařízení od Raspberry Pi. Kvůli tomu, že byly použity staré mikropočítače, které nemají vestavěný čip Wi-Fi, je nutné ke každému zařízení připojit Wi-Fi adaptér. Základní parametry těchto mikropočítačů jsou uvedeny v tabulce 7-1.

Tabulka 7-1 Základní parametry mikropočítače

Raspberry Pi verze	Wi-Fi adaptér verze	IPv4 adresa	Paměťová karta (kapacita[GB])
Raspberry Pi 1	tp-link Model WN-722N	192.168.137.180	4
Raspberry Pi 2 Model B V1.1	Alfa Network Model AWUS036H	192.168.137.186	64

7.1 Kominikace prostřednictvím AES

Rasberry Pi 2 byla použita jako broker a publisher. Raspberry Pi 1 jako subscriber. Jako kryptografický protokol byl použit protokol AES-128 v módu CBC, implementace kterého byla podrobně popsána v části 6.3.

Na začátku je potřeba nainstalovat potřebné knihovny do Raspberry Pi, např. Mosquitto. Poté lze zahájit komunikaci mezi publisherem a subscriberem.
Prvním krokem bude spuštění brokera na Raspberry Pi 2 pomoci příkazu 19. Musíte to udělat v samostatném příkazovém řádku.

Příkaz 19

```
mosquito -v
```

Dále na Raspberry Pi 1 bude spuštěn subscriber pomocí příkazu 20, který se přihlásí k tématu test_channel a bude očekávat na zprávu od publishera.

Příkaz 20

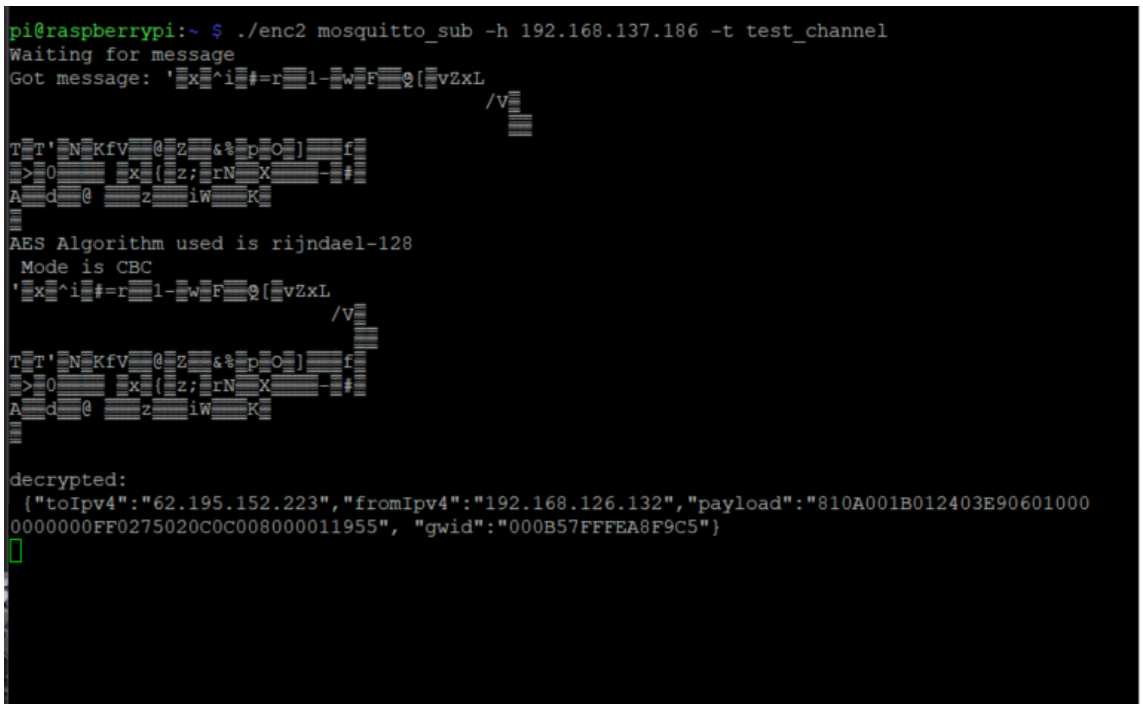
```
./enc2 mosquitto_sub -h 192.168.137.186 -t test_channel
```

Po dvou předchozích příkazech bude spuštěn publisher na Raspberry Pi 2 pomocí příkazu 21.

Příkaz 21

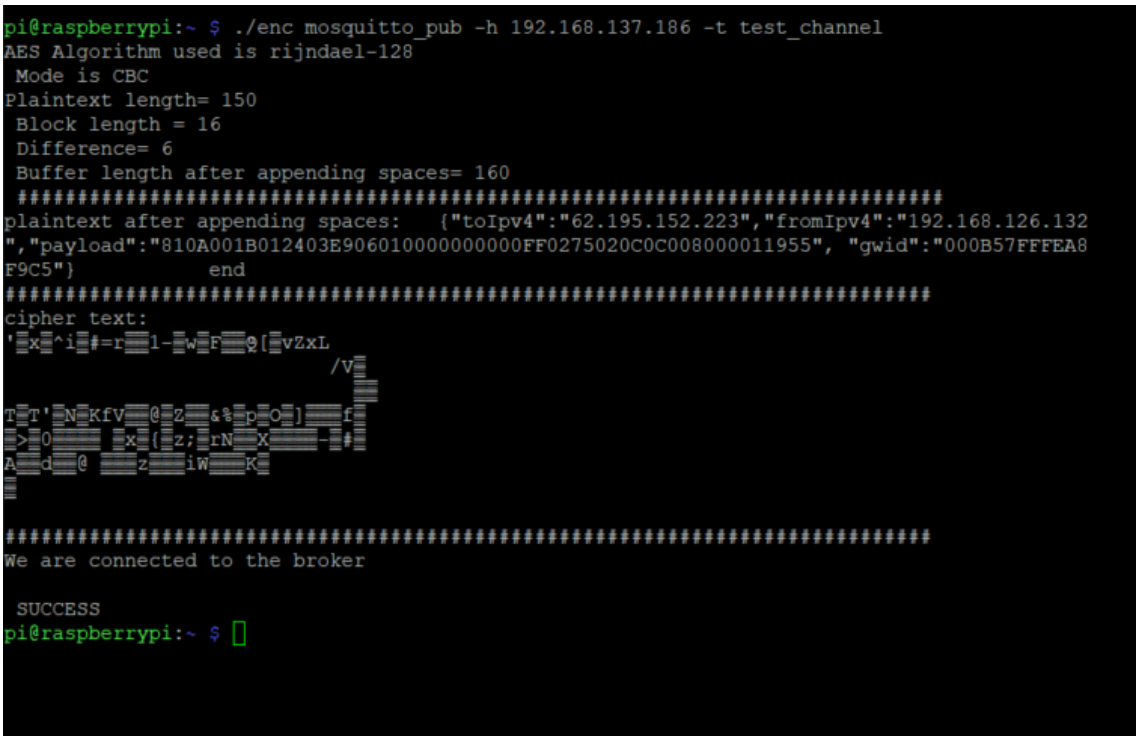
./enc mosquitto_pub -h 192.168.137.186 -t test_channel

Na obrázku 7.1 je subscriber, který je přihlášen k odběru tématu test_channel a který obdržel zašifrovanou zprávu od publishera, kterou se následně dešifroval.



obr. 7-1: Subscriber na Raspberry Pi 1

Na obrázku 7-2 je publisher, který šifruje a posílá zprávy na kanál test_channel.



obr. 7-2: Publisher na Raspberry Pi 2

Byla provedena analýza času potřebného mikropočítačům k šifrování a dešifrování zprávy. Šifrování trvalo déle, protože na tomto mikropočítači byl povolen také broker, který proces zpomalil. Lze vidět, že čas potřebný k zašifrování a dešifrování Raspberry Pi závisí na délce klíče a na použitém módu. Výsledky šifrování pomocí 128-bitového klíče jsou víceméně podobné. Čas potřebný pro dešifrování v módů ECB je menší, protože je méně bezpečný ze všech použitých modů. Výsledky šifrování a dešifrování při použití 256-bitového klíče se navzájem příliš neliší. Všechny výsledky jsou uvedeny v tabulce 7-2.

Tabulka 7-2 Doba trvání šifrování a dešifrování pří různých modéch na Pi 1 a Pi2

Délka klíče (bit)	Mód	Šifrování (ms)	Dešifrování (ms)
128	CBC	20-21	5-6
256		22-24	7-8
128	ECB	19-20	3-4
256		20-23	7-8
128	OFB	20-21	5-6
256		21-22	8-9
128	CFB	20	4-5
256		21-23	6-8
128	CTR	19-20	5-6
256		22-23	7-8

7.2 Komunikace prostřednictvím Rabin

V této části se popisuje bezpečná komunikace mezi mikropočítači pomocí kryptosystému Rabin.

Rabin Cryptosystem je asymetrický kryptografický algoritmus, který je založen na kvadratické shodě. Tato kryptografická technika zahrnuje pár soukromých klíčů *(p, q)* a veřejný klíč *n*. Toto *n* se nazývá „Blum Integer“ a prostý text *x* by měl být vždy: $1 < x < n$. Jeho bezpečnost je určena celočíselnou faktorizací. Matematicky se ukázalo, že kryptosystém Rabin je výpočetně bezpečný proti útoku hrubou silou, pokud útočník není schopen získat správné faktory.

Jak již bylo popsáno v odstavci 5.4, existuje postup, kterým je třeba se řídit při navrhování kryptosystému: generování klíčů, šifrování a dešifrování. Na Raspberry Pi 2 bude publisher, který bude generovat klíče, šifrovat zprávu a posílat ji brokeru. Na Raspberry Pi 1 bude subscriber, který bude přijímat zprávy od publishera a dešifrovat je.

Nejprve byly vybrány dvě velké prvočísla $p = 311$ a $q = 239$, která splňují podmínku $p \neq q$, $p \equiv q \equiv 3 \pmod 4$. Poté bylo nalezeno *n* vynásobením *p* na *q*. Toto číslo publisher publikuje jako veřejný klíč a ulože *p* a *q* jako soukromý klíč.

Po vygenerování potřebných klíčů publisher zašifruje zprávu pomocí funkce encrypter, kterou pošle brokeru. Funkce encrypter (obr. 7-3) má dva parametry, plaintext *m* a veřejný klíč *n*. Vrací celočíselnou hodnotu *c*, kde *c* je ciphertext.

```
int encrypter(int m, int n)
{
int c = (m * m)%n; // c = (m^2) mod n
return c;
}
```

obr. 7-3: Funkce encrypter

Subscriber převezme ciphertext *c* od publishera a dvojice soukromých klíčů *(p, q)*. Poté vypočítá m_p a m_q pomocí následujících vzorců:

$m_p = C^{(p+1)/4} \bmod p$

$m_q = C^{(q+1)/4} \bmod q$

Poté subscriber zjistí *a* a *b* pomocí rozšířeného euklidovského algoritmu

$(a.p + b.q = 1)$

Dále subscriber vypočítá kořeny pomocí následujících vzorků:

$X = (a*p* m_p + b*q*s) \bmod p$

$Y = (a*p* m_q - b*q*s) \bmod q$

$r1 = X, r2 = -X, r3 = Y, r4 = -Y$

Převeďte hodnotu ASCII na text a z těchto 4 kořenů bude jeden plaintext.

Na obrázku 7-4 lze vidět, že zpráva „Hello Subscriber!“ byla zašifrována pomocí kryptosystému Rabin. Na obrázku 7-5 jsou výsledky dešifrování. Čas potřebný pro šifrování a dešifrování závisí na délce zprávy a délce vybraných klíčů. V tomto případě je délka zprávy 17 znaků a klíče nemají zabezpečenou délku, takže doba potřebná k dešifrování je pouze 1 ms. Pokud by byly vybrány zabezpečené klíče, pak by čas potřebný k dešifrování byl delší.

```
pi@raspberrypi:~ $ ./rabpub
Plain text: Hello Subscriber!
Lenght of message: 17
 Encrypted text: 518410201116641166412321102468891368996041322598011299611025960410201129961089O
pi@raspberrypi:~ $
```

obr. 7-4: Rabin publisher

```
pi@raspberrypi:~ $ ./rabsub
Ecrypted text: 518410201116641166412321102468891368996041322598011299611025960410201129961089O
It took me 1 milliseconds.
Decrypted text: H e l l o   S u b s c r i b e r !
pi@raspberrypi:~ $
```

obr. 7-5: Rabin subscriber

ZÁVĚR

Pro uvedení kontextu práce v teoretické části bakalářské práce byl vysvětlen koncept internet věcí, metody možných útoků na zařízení IoT a metody ochrany, z nichž některé byly dále použity v praktické části práce. Následně byl nastudován protokol MQTT, který byl popsán v 5 kapitole. Tento protokol byl poté použit v praktické části.

Cílem praktické části bakalářské práce byla implementace protokolu zabezpečení pro MQTT mezi publisherem, brokerem a subscriberem.

Nejprve byla ukázána nezašifrovaná komunikace mezi MQTT publisherem a subscriberem. To bylo provedeno za účelem pochopení toho, jak funguje protokol MQTT. Poté byl vytvořen model, který umožňuje šifrování a dešifrování zpráv pomocí AES během komunikace mezi publisherem a subscriberem. Různé délky klíčů a režimy blokové šifry byly použity k porovnání času potřebného pro šifrování a dešifrování zpráv. Dále byla nastudována knihovna ecispy, která šifruje a dešifruje zprávy pomocí eliptických křivek. Poté byl popsán způsob komunikace mezi serverem a klientem pomocí protokolu TLS a výměny certifikátů. A na konci byla provedena implementaci zabezpečené komunikace mezi mikropočítači pomocí MQTT.

Literatura

[1] ATLAM, Hany F., Robert J. WALTERS a Gary B. WILLS. Internet of Things: State-of-the-art, Challenges, Applications, and Open Issues. International Journal of Intelligent Computing Research. 2018, 9(3), 928-938. ISSN 20424655. Dostupné z: doi:10.20533/ijicr.2042.4655.2018.0112

[2] IoT Architecture. Iotwf.com [online]. 2014 [cit. 2020-11-22]. Dostupné z: http://cdn.iotwf.com/resources/72/IoT_Reference_Model_04_June_2014.pdf

[3] Security of IoT. Aws.amazon.com [online]. [cit. 2020-11-22]. Dostupné z: https://aws.amazon.com/ru/iot-device-defender/

[4] MAPLE, Carsten. Security and privacy in the internet of things. Journal of Cyber Policy. 2017, 2(2), 155-184. ISSN 2373-8871. Dostupné z: doi:10.1080/23738871.2017.1366536

[5] SUO, Hui, Jiafu WAN, Caifeng ZOU a Jianqi LIU. Security in the Internet of Things: A Review. 2012 International Conference on Computer Science and Electronics Engineering. IEEE, 2012, 2012, 2012, 648-651. ISBN 978-0-7695-4647-6. Dostupné z: doi:10.1109/ICCSEE.2012.373

[6] KVARDA, Lukas, Pavel HNYK, Lukas VOJTECH a Marek NERUDA. Software Implementation of Secure Firmware Update in IoT Concept. Advances in Electrical and Electronic Engineering. 2017, 15(4), 626 - 632. ISSN 1804-3119. Dostupné z: doi:10.15598/aeee.v15i4.2467

[7] CORSER, George, Jared BIELBY a Sukanya MANDAL. Internet Of Things (Iot) Security Best Practices. IEEE Community-led White Paper. IEEE, 2017, 2017, 4-9.

[8] LUKATSKIY, Alexey. Kryptografie v IoT. Https://www.ruscrypto.ru/ [online]. Rusko, 2016 [cit. 2020-11-22]. Dostupné z: https://www.ruscrypto.ru/resource/archive/rc2016/files/05_lukatskiy.pdf

[9] SHAH, Darshana Pritam a Pritam Gajkumar SHAH. Revisting of elliptical curve cryptography for securing Internet of Things (IOT). 2018 Advances in Science and Engineering Technology International Conferences (ASET). IEEE, 2018, 2018, 2018, 1-3. ISBN 978-1-5386-2399-2. Dostupné z: doi:10.1109/ICASET.2018.8376830

[10] SOPORI, Diksha, Tanaya PAWAR, Manjiri PATIL a Roopkala RAVINDRAN. Internet of Things: Security Threats. International Journal of Advanced Research in

Computer Engineering & Technology (IJARCET). ISSN, 2017, 2017(Volume 6, 3), 263-267.

[11] FRAENKEL, Osmond K. ALAN F. WESTIN. Privacy and Freedom. Pp. xvi. New York: Atheneum, 1967. $10.00. The ANNALS of the American Academy of Political and Social Science. 2016, 377(1), 196-197. ISSN 0002-7162. Dostupné z: doi:10.1177/000271626837700157

[12] ZIEGELDORF, Jan Henrik, Oscar Garcia MORCHON a Klaus WEHRLE. Privacy in the Internet of Things: threats and challenges. Security and Communication Networks. 2014, 7(12), 2728-2742. ISSN 19390114. Dostupné z: doi:10.1002/sec.795

[13] MQTT [online]. [cit. 2020-11-22]. Dostupné z: https://mqtt.org/

[14] MQTT Architecture [online]. [cit. 2020-11-22]. Dostupné z: https://www.netio-products.com/en/glossary/mqtt

[15] IoT Architecture [online]. [cit. 2020-11-29]. Dostupné z: https://iottimes.wordpress.com

[16] IoT attacks [online]. [cit. 2020-11-29]. Dostupné z: https://www.researchgate.net/figure/arious-security-attacks-in-the-IoT-system_fig2_332859761

[17] Mosquitto. Https://medium.com/ [online]. [cit. 2020-11-29]. Dostupné z: https://medium.com/@bhagvankommadi/mosquitto-mqtt-2a352bd8f179

[18] Funkce Mosquitto [online]. [cit. 2020-11-29]. Dostupné z: https://mosquitto.org/api/files/mosquitto-h.html

[19] Block cipher mode of operation [online]. [cit. 2020-11-30]. Dostupné z: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[20] ECIES [online]. [cit. 2020-12-06]. Dostupné z: https://asecuritysite.com/encryption/ecc3

[21] ECIES [online]. [cit. 2020-12-06]. Dostupné z: https://ru.wikipedia.org/wiki/ECIES

[22] MALINA, Lukas, Gautam SRIVASTAVA, Petr DZURENDA, Jan HAJNY a Radek FUJDIAK. A Secure Publish/Subscribe Protocol for Internet of Things. *Proceedings of the 14th International Conference on Availability, Reliability and Security*. New York, NY, USA: ACM, 2019, 2019-08-26, , 1-10. ISBN 9781450371643. Dostupné z: doi:10.1145/3339252.3340503

[23] *Journal of Mathematics and Statistics*. 10. 2014. ISSN 1549-3644. Dostupné také z: http://thescipub.com/abstract/10.3844/jmssp.2014.304.308

[24] LEE, Homin K., Tal MALKIN a Erich NAHUM. Cryptographic strength of ssl/tls servers. *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*. New York, New York, USA: ACM Press, 2007, 2007, , 83-. ISBN 9781595939081. Dostupné z: doi:10.1145/1298306.1298318

[25] *Handshake protocol* [online]. [cit. 2021-04-18]. Dostupné z: https://sites.google.com/site/tlssloverview/handshake-process

[26] *Rabin Cryptosystem with Implementation* [online]. [cit. 2021-4-25]. Dostupné z: https://www.geeksforgeeks.org/rabin-cryptosystem-with-implementation/

Seznam příloh

Příloha 1 - **Publisher**

```c
#include <stdio.h>
#include <mosquitto.h>

int main(){
     int rc;
     struct mosquitto *mosq;

     mosquitto_lib_init();
     mosq = mosquitto_new("publisher-test",true,NULL);
     rc = mosquitto_connect(mosq, "localhost", 1883, 60);

     if(rc !=0){
     printf("Client could not connect to broker! Error code: %d\n",rc);
     mosquitto_destroy(mosq);
     return -1;
     } printf("We are connected to the broker\n");
     mosquitto_publish(mosq, NULL, "test/t1", 6, "Hello", 0, false);

     mosquitto_disconnect(mosq);
     mosquitto_destroy(mosq);
     mosquitto_lib_cleanup();
     return 0;
}
```

Příloha 2 - **Subscriber**

```c
#include <stdio.h>
#include <stdlib.h>
#include <mosquitto.h>

void on_connect(struct mosquitto *mosq, void *obj, int rc) {
     printf("ID: %d\n", * (int *) obj);
     if(rc) {
             printf("Error with result code: %d\n", rc);
             exit(-1);
     }
     mosquitto_subscribe(mosq, NULL, "test/t1", 0);
}

void on_message(struct mosquitto *mosq, void *obj, const struct
mosquitto_message *msg) {
printf("New message with topic %s: %s\n", msg->topic, (char *) msg-
>payload);
}

int main() {
     int rc, id=12;

     mosquitto_lib_init();

     struct mosquitto *mosq;

     mosq = mosquitto_new("subscribe-test", true, &id);
     mosquitto_connect_callback_set(mosq, on_connect);
     mosquitto_message_callback_set(mosq, on_message);

     rc = mosquitto_connect(mosq, "localhost", 1883, 10);
     if(rc) {
         printf("Could not connect to Broker with return code %d\n", rc);
         return -1;
     }

     mosquitto_loop_start(mosq);
     printf("Press Enter to quit...\n");
     getchar();
     mosquitto_loop_stop(mosq, true);

     mosquitto_disconnect(mosq);
     mosquitto_destroy(mosq);
     mosquitto_lib_cleanup();

     return 0;
}
```

Příloha 3 - **Publisher s AES**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mosquitto.h>
#include <mcrypt.h>
#include <math.h>
#include <stdint.h>

#define MQTT_HOSTNAME "192.168.126.132"
#define MQTT_PORT 1883
#define MQTT_USERNAME ""
#define MQTT_PASSWORD ""
#define MQTT_TOPIC "simple"

int encrypt (void *buffer, int buffer_len, char *IV, char *key, int key_len){
        MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);

        mcrypt_generic_init(td, key, key_len, IV);
        mcrypt_generic(td, buffer, buffer_len);
        mcrypt_generic_deinit(td);
        mcrypt_module_close(td);

        return 0;
}

int decrypt (void *buffer, int buffer_len, char *IV, char *key, int key_len){
        MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);

        mcrypt_generic_init(td, key, key_len, IV);
        mcrypt_generic(td, buffer, buffer_len);
        mcrypt_generic_deinit(td);
        mcrypt_module_close(td);

        return 0;
}

void display(char *ciphertext, int len){
        int v;
        printf("%s \n",ciphertext);
        printf("\n");
}


void mosquitto_routine(char *buffer){

        struct mosquitto *mosq = NULL;
        mosquitto_lib_init();
        mosq = mosquitto_new (NULL, true, NULL);

        if(!mosq){
```

```
                fprintf(stderr, "Can't initialize Mosquitto library\n");
                exit(-1);
        }


        int ret = mosquitto_connect(mosq, MQTT_HOSTNAME, 1883, 60);

        if(ret != 0){
                fprintf(stderr,"Can't connect to Mosquitto server\n");
                exit(-1);
        } printf("We are connected to the broker\n");

        ret = mosquitto_publish(mosq, NULL, MQTT_TOPIC, strlen(buffer), buffer,
0, false);

        if(ret){
                fprintf(stderr,"Can't publish to Mosquitto server\n");
                exit(-1);
        }

}

int main(){

  char * plaintext =
"{\"toIpv4\":\"62.195.152.223\",\"fromIpv4\":\"192.168.126.132\",\"payload\":\
"810A001B012403E9060100000000000FF0275020C0C008000011955\",
\"gwid\":\"000B57FFFEA8F9C5\"}";
  char* IV = "AAAAAAAAAAAAAAAA";
  char *key = "this_is_my_key03";
  int keysize = 16; /* 128 bits */
  char* buffer;

  int rv;

  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  int plaintext_len = strlen(plaintext);
  int diff = plaintext_len%blocksize;
  buffer = calloc(1, plaintext_len+blocksize-diff);
  strncpy(buffer,plaintext,plaintext_len);

  if (diff!=0){
        memset(buffer+plaintext_len,' ', blocksize-diff);
  }

  int buffer_len = strlen(buffer);

  printf("AES Algorithm used is rijndael-128 \n Mode is CBC \n");
  printf("Plaintext length= %d \n Block length = %d \n Difference= %d \n
Buffer length after appending spaces= %d \n
",plaintext_len,blocksize,diff,buffer_len);

printf("##########################################################################
#######\n");
  printf("plaintext after appending spaces:    %s end\n", buffer);

printf("##########################################################################
#######\n");
```

```
  rv=encrypt(buffer, buffer_len, IV, key, keysize);
  printf("cipher text:  \n");
  display(buffer , buffer_len);

printf("################################################################################
#######\n");

  mosquitto_routine(buffer);

  printf("\n SUCCESS \n");
  return 0;

}
```

Příloha 4 - **Subscriber s AES**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mosquitto.h>
#include <mcrypt.h>
#include <math.h>
#include <stdint.h>

#define MQTT_HOSTNAME "192.168.126.132"
#define MQTT_PORT 1883
#define MQTT_USERNAME ""
#define MQTT_PASSWORD ""
#define MQTT_TOPIC "simple"

int decrypt(
    void* buffer,
    int buffer_len,
    char* IV,
    char* key,
    int key_len
){
  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  mcrypt_generic_init(td, key, key_len, IV);
  mdecrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);
  return 0;
}

void display(char* ciphertext, int len){
  int v;
  printf("%s \n",ciphertext);
  printf("\n");
}

void decrypt_master (char *buffer){
        char *IV = "AAAAAAAAAAAAAAAA";
```

```
        char *key = "this_is_my_key03";
        int keysize = 16;
        int rv;

  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  int buffer_len = strlen(buffer);
  printf("AES Algorithm used is rijndael-128 \n Mode is CBC \n");
  display(buffer , buffer_len);
  rv=decrypt(buffer, buffer_len,IV, key, keysize);
  printf("decrypted:\n %s\n", buffer);

}

void my_message_callback(struct mosquitto *mosq, void *obj,
        const struct mosquitto_message *message){

        char *buffer = (char *)malloc(65536);
        buffer = (char *)message->payload;
        printf("Got message: %s\n", (char *)message->payload);
        decrypt_master(buffer);

}

void mqtt_receive(){
        struct mosquitto *mosq = NULL;
        mosquitto_lib_init();
        mosq = mosquitto_new (NULL,true,NULL);

        if(!mosq){
                fprintf(stderr, "Can't initialize Mosquitto library\n");
                exit(-1);
        }

        int ret = mosquitto_connect (mosq, MQTT_HOSTNAME, 1883, 60);
  if (ret !=0)
    {
      fprintf (stderr, "Can't connect to Mosquitto server\n");
      exit (-1);
    }

    mosquitto_subscribe(mosq, NULL, MQTT_TOPIC,0);
    mosquitto_message_callback_set (mosq, my_message_callback);
    printf("Waiting for message\n");

    mosquitto_loop_forever (mosq, -1, 1);

}



int main()
{

  mqtt_receive();

  return 0;
}
```

Příloha 5 - **SSL server**

```
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <resolv.h>
#include "openssl/ssl.h"
#include "openssl/err.h"

#define FAIL    -1

int OpenListener(int port)
{   int sd;
    struct sockaddr_in addr;

    sd = socket(PF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = INADDR_ANY;
    if ( bind(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0 )
    {
        perror("can't bind port");
        abort();
    }
    if ( listen(sd, 10) != 0 )
    {
        perror("Can't configure listening port");
        abort();
    }
    return sd;
}

int isRoot()
{
    if (getuid() != 0)
    {
        return 0;
    }
```

```
    else
    {
        return 1;
    }

}
SSL_CTX* InitServerCTX(void)
{   const SSL_METHOD *method;
    SSL_CTX *ctx;

    OpenSSL_add_all_algorithms();  /* load & register all
cryptos, etc. */
    SSL_load_error_strings();   /* load all error messages */
    method = TLSv1_2_server_method();  /* create new server-
method instance */
    ctx = SSL_CTX_new(method);   /* create new context from
method */
    if ( ctx == NULL )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    return ctx;
}

void LoadCertificates(SSL_CTX* ctx, char* CertFile, char*
KeyFile)
{
    /* set the local certificate from CertFile */
    if ( SSL_CTX_use_certificate_file(ctx, CertFile,
SSL_FILETYPE_PEM) <= 0 )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    /* set the private key from KeyFile (may be the same as
CertFile) */
    if ( SSL_CTX_use_PrivateKey_file(ctx, KeyFile,
SSL_FILETYPE_PEM) <= 0 )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    /* verify private key */
    if ( !SSL_CTX_check_private_key(ctx) )
    {
        fprintf(stderr, "Private key does not match the public
certificate\n");
        abort();
```

```
    }
}

void ShowCerts(SSL* ssl)
{   X509 *cert;
    char *line;

    cert = SSL_get_peer_certificate(ssl); /* Get certificates
(if available) */
    if ( cert != NULL )
    {
        printf("Server certificates:\n");
        line = X509_NAME_oneline(X509_get_subject_name(cert), 0,
0);
        printf("Subject: %s\n", line);
        free(line);
        line = X509_NAME_oneline(X509_get_issuer_name(cert), 0,
0);
        printf("Issuer: %s\n", line);
        free(line);
        X509_free(cert);
    }
    else
        printf("No certificates.\n");
}

void Servlet(SSL* ssl) /* Serve the connection -- threadable */
{   char buf[1024];
    char reply[1024];
    int sd, bytes;
    const char*
HTMLecho="<html><body><pre>%s</pre></body></html>\n\n";

    if ( SSL_accept(ssl) == FAIL )     /* do SSL-protocol accept
*/
        ERR_print_errors_fp(stderr);
    else
    {
        ShowCerts(ssl);        /* get any certificates */
        bytes = SSL_read(ssl, buf, sizeof(buf)); /* get request
*/
        if ( bytes > 0 )
        {
            buf[bytes] = 0;
            printf("Client msg: \"%s\"\n", buf);
            sprintf(reply, HTMLecho, buf);   /* construct reply
*/
            SSL_write(ssl, reply, strlen(reply)); /* send reply
*/
```

```
        }
        else
            ERR_print_errors_fp(stderr);
    }
    sd = SSL_get_fd(ssl);       /* get socket connection */
    SSL_free(ssl);         /* release SSL state */
    close(sd);          /* close connection */
}

int main(int count, char *strings[])
{   SSL_CTX *ctx;
    int server;
    char *portnum;

    if(!isRoot())
    {
        printf("This program must be run as root/sudo user!!");
        exit(0);
    }
    if ( count != 2 )
    {
        printf("Usage: %s <portnum>\n", strings[0]);
        exit(0);
    }
    SSL_library_init();

    portnum = strings[1];
    ctx = InitServerCTX();        /* initialize SSL */
    LoadCertificates(ctx, "mycert.pem", "mycert.pem"); /* load
certs */
    server = OpenListener(atoi(portnum));    /* create server
socket */
    while (1)
    {   struct sockaddr_in addr;
        socklen_t len = sizeof(addr);
        SSL *ssl;

        int client = accept(server, (struct sockaddr*)&addr,
&len);  /* accept connection as usual */
        printf("Connection: %s:%d\n",inet_ntoa(addr.sin_addr),
ntohs(addr.sin_port));
        ssl = SSL_new(ctx);              /* get new SSL state
with context */
        SSL_set_fd(ssl, client);      /* set connection socket
to SSL state */
        Servlet(ssl);         /* service connection */
    }
```

Příloha 6 - **SSL klient**

```
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <sys/socket.h>
#include <resolv.h>
#include <netdb.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#define FAIL    -1

int OpenConnection(const char *hostname, int port)
{   int sd;
    struct hostent *host;
    struct sockaddr_in addr;

    if ( (host = gethostbyname(hostname)) == NULL )
    {
        perror(hostname);
        abort();
    }
    sd = socket(PF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = *(long*)(host->h_addr);
    if ( connect(sd, (struct sockaddr*)&addr, sizeof(addr)) !=
0 )
    {
        close(sd);
        perror(hostname);
        abort();
    }
    return sd;
}

SSL_CTX* InitCTX(void)
{   const SSL_METHOD *method;
    SSL_CTX *ctx;

    OpenSSL_add_all_algorithms();  /* Load cryptos, et.al. */
```

```
    SSL_load_error_strings();   /* Bring in and register error
messages */
    method = TLSv1_2_client_method();  /* Create new client-
method instance */
    ctx = SSL_CTX_new(method);   /* Create new context */
    if ( ctx == NULL )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    return ctx;
}

void ShowCerts(SSL* ssl)
{   X509 *cert;
    char *line;

    cert = SSL_get_peer_certificate(ssl); /* get the server's
certificate */
    if ( cert != NULL )
    {
        printf("Server certificates:\n");
        line = X509_NAME_oneline(X509_get_subject_name(cert),
0, 0);
        printf("Subject: %s\n", line);
        free(line);       /* free the malloc'ed string */
        line = X509_NAME_oneline(X509_get_issuer_name(cert),
0, 0);
        printf("Issuer: %s\n", line);
        free(line);       /* free the malloc'ed string */
        X509_free(cert);     /* free the malloc'ed certificate
copy */
    }
    else
        printf("Info: No client certificates configured.\n");
}

int main(int count, char *strings[])
{   SSL_CTX *ctx;
    int server;
    SSL *ssl;
    char buf[1024];
    int bytes;
    char *hostname, *portnum;

    if ( count != 3 )
    {
        printf("usage: %s <hostname> <portnum>\n",
strings[0]);
```

```
        exit(0);
    }
    SSL_library_init();
    hostname=strings[1];
    portnum=strings[2];

    ctx = InitCTX();
    server = OpenConnection(hostname, atoi(portnum));
    ssl = SSL_new(ctx);      /* create new SSL connection
state */
    SSL_set_fd(ssl, server);    /* attach the socket
descriptor */
    if ( SSL_connect(ssl) == FAIL )   /* perform the
connection */
        ERR_print_errors_fp(stderr);
    else
    {   char *msg = "Hello Server";

        printf("Connected with %s encryption\n",
SSL_get_cipher(ssl));
        ShowCerts(ssl);        /* get any certs */
        SSL_write(ssl, msg, strlen(msg));   /* encrypt & send
message */
        bytes = SSL_read(ssl, buf, sizeof(buf)); /* get reply
& decrypt */
        buf[bytes] = 0;
        printf("Received: \"%s\"\n", buf);
        SSL_free(ssl);        /* release connection state */
    }
    close(server);         /* close socket */
    SSL_CTX_free(ctx);        /* release context */
    return 0;
}
```

Příloha 7 - **Rabin kryptosystém**

```
import java.math.BigInteger;
import java.nio.charset.Charset;
import java.security.SecureRandom;
import java.util.Random;

class Cryptography {
    private static Random r = new SecureRandom();
    private static BigInteger TWO = BigInteger.valueOf(2);
    private static BigInteger THREE = BigInteger.valueOf(3);
    private static BigInteger FOUR = BigInteger.valueOf(4);

    public static BigInteger[] generateKey(int bitLength)
```

```
    {
        BigInteger p = blumPrime(bitLength / 2);
        BigInteger q = blumPrime(bitLength / 2);
        BigInteger N = p.multiply(q);
        return new BigInteger[] { N, p, q };
    }

    public static BigInteger encrypt(BigInteger m,
                                     BigInteger N)
    {
        return m.modPow(TWO, N);
    }

    public static BigInteger[] decrypt(BigInteger c,
                                       BigInteger p,
                                       BigInteger q)
    {
        BigInteger N = p.multiply(q);
        BigInteger p1 = c.modPow(p
                                     .add(BigInteger.ONE)
                                     .divide(FOUR),
                                 p);
        BigInteger p2 = p.subtract(p1);
        BigInteger q1 = c.modPow(q
                                     .add(BigInteger.ONE)
                                     .divide(FOUR),
                                 q);
        BigInteger q2 = q.subtract(q1);

        BigInteger[] ext = Gcd(p, q);
        BigInteger y_p = ext[1];
        BigInteger y_q = ext[2];

        BigInteger d1 = y_p.multiply(p)
                           .multiply(q1)
                           .add(y_q.multiply(q)
                                   .multiply(p1))
                           .mod(N);
        BigInteger d2 = y_p.multiply(p)
                           .multiply(q2)
                           .add(y_q.multiply(q)
                                   .multiply(p1))
                           .mod(N);
        BigInteger d3 = y_p.multiply(p)
                           .multiply(q1)
                           .add(y_q.multiply(q)
                                   .multiply(p2))
                           .mod(N);
        BigInteger d4 = y_p.multiply(p)
                           .multiply(q2)
                           .add(y_q.multiply(q)
                                   .multiply(p2))
                           .mod(N);

        return new BigInteger[] { d1, d2, d3, d4 };
    }
```

```
    public static BigInteger[] Gcd(BigInteger a, BigInteger b)
    {
        BigInteger s = BigInteger.ZERO;
        BigInteger old_s = BigInteger.ONE;
        BigInteger t = BigInteger.ONE;
        BigInteger old_t = BigInteger.ZERO;
        BigInteger r = b;
        BigInteger old_r = a;
        while (!r.equals(BigInteger.ZERO)) {
            BigInteger q = old_r.divide(r);
            BigInteger tr = r;
            r = old_r.subtract(q.multiply(r));
            old_r = tr;

            BigInteger ts = s;
            s = old_s.subtract(q.multiply(s));
            old_s = ts;

            BigInteger tt = t;
            t = old_t.subtract(q.multiply(t));
            old_t = tt;
        }
        return new BigInteger[] { old_r, old_s, old_t };
    }

    public static BigInteger blumPrime(int bitLength)
    {
        BigInteger p;
        do {
            p = BigInteger.probablePrime(bitLength, r);
        } while (!p.mod(FOUR).equals(THREE));
        return p;
    }
}


public class RabinCryptosystem {
    public static void main(String[] args)
    {
        BigInteger[] key = Cryptography.generateKey(512);
        BigInteger n = key[0];
        BigInteger p = key[1];
        BigInteger q = key[2];
        String finalMessage = null;
        int i = 1;
        String s = "Rabin Cryptosystem";

        System.out.println("Message sent by sender : " + s);

        BigInteger m
            = new BigInteger(
                s.getBytes(
                    Charset.forName("ascii")));
        BigInteger c = Cryptography.encrypt(m, n);

        System.out.println("Encrypted Message : " + c);
```

```
        BigInteger[] m2 = Cryptography.decrypt(c, p, q);
        for (BigInteger b : m2) {
            String dec = new String(
                b.toByteArray(),
                Charset.forName("ascii"));
            if (dec.equals(s)) {
                finalMessage = dec;
            }
            i++;
        }
        System.out.println(
            "Message received by Receiver : "
            + finalMessage);
    }
}
```

```
#include <iostream>
#include <vector>
#include <cstring>
#include <sys/time.h>


using namespace std;

int  p =311 , q = 239; //private key pair(p,q) of the form 3 mod
4

int n = p*q;
int len;

int encrypter(int m, int n)
{
    int c = (m * m)%n;     // c = (m^2) mod n
    return c;
}

long mtime()
{
  struct timeval t;

  gettimeofday(&t, NULL);
  long mt = (long)t.tv_sec * 1000 + t.tv_usec / 1000;
  return mt;
}


int mod(int k, int b, int m) //chinese remainder theorem
implementation
```

```
{
int i=0;
int a=1;

vector<int> t;
        while(k>0){
        t.push_back(k%2);
        k=(k-t[i])/2;
        i++;
        }
for(int j=0; j<i; j++){
        if(t[j]==1){
        a=(a*b)%m;
        b=(b*b)%m;
        } else{
b=(b*b)%m;
}
}
return a;
}
int modulo (int a, int b)
{
return a >= 0 ? a % b : ( b - abs ( a%b ) ) % b;
}
vector<int> Extended_Euclid(int a, int b)
{
if (b>a) {
int temp=a; a=b; b=temp;
}
int x=0;
int y=1;
int lastx=1;
int lasty=0;
while (b!=0) {
int q= a/b;
int temp1= a%b;
a=b;
b=temp1;
int temp2 = x;
x=lastx-q*x;
lastx = temp2;
int temp3 = y;
y = lasty-q*y;
lasty=temp3;
}
vector<int>arr(3);
arr[0] = lastx;
arr[1] = lasty;
arr[2] = 1;
```

```
return arr;
}
int decrypter(int c, int p, int q)
{

int mp = mod((p+1)/4, c, p);
int mq = mod((q+1)/4, c, q);
vector<int> arr = Extended_Euclid(p, q);
int rootp = arr[0]*p*mq;
int rootq = arr[1]*q*mp;
double r = modulo((rootp+rootq), n);
if( r < 128)
return r;
int negative_r = n - r;
if (negative_r < 128)
return negative_r;
int s = modulo((rootp-rootq), n);
if( s < 128)
return s;
int negative_s = n - s;
return negative_s;

}

int main()
{
long t;

vector<int>e; //vector to store the encrypted message
vector<int>d; //vector to store the decrypted message

string message = "Hello Subscriber!";
int len = strlen(message.c_str());
printf( "Ecrypted text: ");

for(int i = 0; i <= len; i++)
{
e.push_back(encrypter(message[i], n));
printf("%d", e[i]);
}
printf("\n");

t = mtime();

for(int i = 0; i < len; i++)
{

d.push_back(decrypter(e[i], p, q));
```

```
}
t = mtime() - t;
printf ("It took me %ld milliseconds.\n",t);
printf( "Decrypted text: ");

for(int i=0;i<d.size();i++)
printf( "%c ", d[i]);
printf("\n");

return 0;
}
```

Příloha 8 - **Subscriber Rabin**

```
#include <iostream>
#include <vector>
#include <cstring>
using namespace std;

int p = 311, q = 239; //private key pair(p,q) of the form 3 mod


int n = p*q;

int c= (m * m)%n;;

int len;

int mod(int k, int b, int m) //chinese remainder theorem
implementation
{
int i=0;
int a=1;
vector<int> t;
while(k>0){
t.push_back(k%2);
k=(k-t[i])/2;
i++;
}
for(int j=0; j<i; j++){
if(t[j]==1){
a=(a*b)%m;
b=(b*b)%m;
} else{
b=(b*b)%m;
}
```

```
}
return a;
}
int modulo (int a, int b)
{
return a >= 0 ? a % b : ( b - abs ( a%b ) ) % b;
}
vector<int> Extended_Euclid(int a, int b)
{
if (b>a) {
int temp=a; a=b; b=temp;
}
int x=0;
int y=1;
int lastx=1;
int lasty=0;
while (b!=0) {
int q= a/b;
int temp1= a%b;
a=b;
b=temp1;
int temp2 = x;
x=lastx-q*x;
lastx = temp2;
int temp3 = y;
y = lasty-q*y;
lasty=temp3;
}
vector<int>arr(3);
arr[0] = lastx;
arr[1] = lasty;
arr[2] = 1;
return arr;
}
int decrypter(int c, int p, int q)
{
int mp = mod((p+1)/4, c, p);
int mq = mod((q+1)/4, c, q);
vector<int> arr = Extended_Euclid(p, q);
int rootp = arr[0]*p*mq;
int rootq = arr[1]*q*mp;
double r = modulo((rootp+rootq), n);
if( r < 128)
return r;
int negative_r = n - r;
if (negative_r < 128)
return negative_r;
int s = modulo((rootp-rootq), n);
if( s < 128)
```

```
return s;
int negative_s = n - s;
return negative_s;
}

int main()
{
vector<int>e;
vector<int>d;
int len = strlen(message.c_str());
printf( "Encrypted text: ");
for(int i = 0; i <= len; i++)
{
e.push_back(encrypter(message[i], n));
printf( “%d”,e[i]);
}
printf("\n");
for(int i = 0; i < len; i++)
{
d.push_back(decrypter(e[i], p, q));
}
printf( "Decrypted text: ");
for(int i=0;i<d.size();i++)
printf( "%c",d[i]);
printf("\n");
return 0;
}
```

Příloha 9 - **Publisher Rabin**

```
#include <iostream>
#include <vector>
#include <cstring>

using namespace std;

int p = 311, q = 239; //private key pair(p,q)
int n = p * q; //public key n

int encrypter(int m, int n)
{
int c = (m * m)%n; // c = (m^2) mod n
return c;
}


int mod(int k, int b, int m) //chinese remainder theorem
implementation
{
int i=0;
int a=1;
vector<int> t;
while(k>0){
t.push_back(k%2);
k=(k-t[i])/2;
i++;
}
for(int j=0; j<i; j++){
if(t[j]==1){
a=(a*b)%m;
b=(b*b)%m;
} else{
b=(b*b)%m;
}
}
return a;
}
int modulo (int a, int b)
{
return a >= 0 ? a % b : ( b - abs ( a%b ) ) % b;
}
vector<int> Extended_Euclid(int a, int b)
{
if (b>a) {
int temp=a; a=b; b=temp;
}
int x=0;
```

```
int y=1;
int lastx=1;
int lasty=0;
while (b!=0) {
int q= a/b;
int temp1= a%b;
a=b;
b=temp1;
int temp2 = x;
x=lastx-q*x;
lastx = temp2;
int temp3 = y;
y = lasty-q*y;
lasty=temp3;
}
vector<int>arr(3);
arr[0] = lastx;
arr[1] = lasty;
arr[2] = 1;
return arr;
}


int main() {
vector <int> e;
string message ="Hello Subscriber!“


printf( "Plain text: %s \n", message.c_str());
int len = strlen(message.c_str());
printf("Lenght of message: %d\n", len);


for(int i = 0; i <= len; i++)
{
e.push_back(encrypter(message[i], n));
printf( "%d",e[i]);
}
printf("\n");


}
```