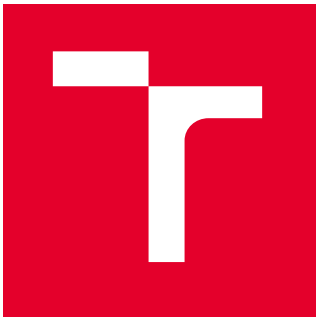


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

BEZPEČNOST AUTONOMNÍ DOPRAVY A VZDÁLENÉHO ŘÍZENÍ

SECURE AUTONOMOUS TRANSPORTATION WITH REMOTE CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Daniel Prachař**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Daniel Prachař ***ID:*** 240969

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

NÁZEV TÉMATU:

Bezpečnost autonomní dopravy a vzdáleného řízení

POKYNY PRO VYPRACOVÁNÍ:

Zaměřte se na kyberbezpečnost v inteligentních transportních systémech (ITS) a v autonomní dopravě. Analyzujte současné bezpečnostní hrozby, útoky a protiopatření v této oblasti. Zhodnoťte a vyberte vhodné kryptografické protokoly pro zajištění bezpečnosti vzdáleného řízení autonomních vozidel. Hlavním cílem bakalářské práce bude demonstrační implementace navrženého protokolu pro bezpečné vzdálené řízení vozidel a výkonnostní zhodnocení daného řešení.

DOPORUČENÁ LITERATURA:

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

[2] LONC, Brigitte, and Pierpaolo CINCILLA. Cooperative its security framework: Standards and implementations progress in europe. 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM). IEEE, 2016.

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

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

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

ABSTRAKT

Tato bakalářská práce se zabývá nastupujícím trendem autonomních vozidel, který se v posledních letech stal jedním z diskutovaných témat v oblasti dopravy a technologií. Autonomní vozidla mají při používání této technologie velké výhody, jako je vyhýbání se nebezpečím na silnici a dopravním podmínkám, nicméně s tímto pokrokem přicházejí nové výzvy a rizika, zejména v oblasti kybernetické bezpečnosti. Teoretická část práce se zaměřuje na představení autonomních vozidel s jejich fungováním. Dále na identifikaci komunikace a analýzu bezpečnostních problémů, které jsou s vozidly spojené. Jedním z hlavních bezpečnostních rizik jsou zranitelnosti těchto vozidel vůči kybernetickým útokům, které mohou ohrozit jak bezpečnost cestujících, tak ostatní účastníky silničního provozu. Hlavním cílem práce je navrhnout vhodné řešení pro bezpečné vzdálené řízení a provést výkonnostní zhodnocení tohoto řešení. Vlastnosti takového řešení by měly garantovat nízké zpoždění v komunikaci a celkovou bezpečnost. Praktická část práce představuje dva protokoly, které toto řešení implementují v aplikaci. Tyto protokoly jsou následně testovány a je provedeno jejich výkonnostní zhodnocení na základě měření zpoždění mezi několika aspekty komunikace. Výsledkem práce je zhodnocení navržených protokolů s jejich doporučením pro praktické využití.

KLÍČOVÁ SLOVA

Autonomní vozidlo, Bezpečná komunikace, Constrained Application Protocol, Message Queuing Telemetry Transport, Teleoperační řízení

ABSTRACT

This bachelor thesis study the emerging trend of Autonomous Vehicle, which has become one of the most discussed topics in the modern field of transportation and technology. Autonomous Vehicle has great benefits using this technology such as avoiding road hazards and traffic conditions however, with these advances, new challenges come and risks especially in the area of cyber security. The theoretical part of this thesis focuses on introducing autonomous vehicles with their operations. Also, the identification of the communication and analysis of the security problems associated with the vehicles. One of the main security risks is the vulnerabilities of these vehicles to cyber attacks, which can threaten the safety of passengers as well as other road users. The main objective of this work is to propose a suitable solution for securing remote driving and to perform a performance evaluation. The features of such solution shall guarantee low communication delay and overall security. Practical part of the thesis presents two protocols that implement this solution in the application. These protocols are then tested and performance evaluation is performed by measuring the delay between several aspects of the communication. The result of the work is an evaluation of the proposed protocols with their recommendation for practical use.

KEYWORDS

Autonomous vehicle, Constrained Application Protocol, Message Queuing Telemetry Transport, Secure communication, Teleoperated driving

Vysázeno pomocí balíčku `thesis` verze 4.09; `https://latex.fekt.vut.cz/`

PRACHAŘ, Daniel. *Bezpečnost autonomní dopravy a vzdáleného řízení*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: doc. Ing. Lukáš Malina, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Daniel Prachař
VUT ID autora:	240969
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Bezpečnost autonomní dopravy a vzdáleného řízení

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

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce doc. Ing. Lukáši Malinovi, Ph.D. za odborné vedení, konzultace, a podnětné návrhy k práci. Rovněž bych chtěl poděkovat všem, kteří mně poskytli podporu při tvorbě bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Rychlý pokrok v oblasti informačních technologií a průmyslové automatizace přivedl automobilový průmysl do nové éry, éry autonomní dopravy. Autonomní vozidla, vybavená pokročilými senzory, umělou inteligencí a komunikačními technologiemi, mají potenciál radikálně transformovat způsob, jakým vnímáme a prožíváme dopravu a zároveň zvýšit, tak její bezpečnost na silnicích. Autonomní vozidla se zrodila z reakce na komplexní výzvy současné dopravy, jako jsou dopravní nehody způsobené lidským faktorem, přetížené dopravní sítě a potřeba efektivnějšího využívání dostupné infrastruktury. Tato bakalářská práce se hlouběji zaměřuje na bezpečnost autonomní dopravy, s důrazem na využití teloperačního řízení autonomních vozidel.

Cílem bakalářské práce je návrh vhodného protokolu pro bezpečné vzdálené řízení vozidel, jeho demonstrační implementace a výkonnostní zhodnocení.

Bakalářská práce je rozdělena na dvě hlavní části, a to první, která je teoretickým východiskem pro druhou navazující praktickou část. První teoretická část uvádí kapitoly, které objasňují pojmem autonomní vozidlo, dále rozdělují autonomii vozidla do úrovní. Rovněž přibližují společnosti, které se zabývají vývojem a provozem autonomních vozidel. Popisují systémy pro detekci okolního prostředí a dělí komunikaci vozidel na vnější a vnitřní. Dále popisují zranitelnosti vnitřní a vnější komunikace, druhy kybernetických útoků a možnosti protiopatření.

Druhá praktická část práce je tvořena popisem teleoperačního řízení autonomních vozidel, návrhem komunikačního modelu a analýzou jeho rizik. Dále je proveden návrh a zhodnocení vhodných kryptografických protokolů pro bezpečné teleoperační řízení. Na základě zhodnocení byly vybrány dva protokoly, a to Message Queuing Telemetry Transport a Constrained Application Protocol, ty byly následně implementovány v programovacím jazyce Java, kde byla vytvořena jejich nezabezpečená i zabezpečená verze. U vytvořených implementací vybraných protokolů byly poté testovány doby navázání komunikace a míry latence. V poslední kapitole bylo provedeno zhodnocení navržených protokolů a bylo představeno jejich doporučení pro praktické využití.

1 Autonomní vozidla

Autonomní vozidla, často označována jako samořídicí vozidla, představují jednu z největších inovací v automobilovém průmyslu a oblasti mobility v moderní době. Tyto vozidla mají potenciál revolučně změnit způsob, jakým se pohybujeme po silnicích a přepravujeme se z místa na místo.

Autonomní vozidlo je speciální typ běžného automobilu, který však je vybaven pokročilými senzory, speciálním softwarem, umělou inteligencí a dalšími potřebnými systémy, jež spolu navzájem komunikují. Při jízdě je vozidlo schopno vnímat své okolí a sbírat tak živá data. Na základě vyhodnocení těchto dat je poté vozidlo schopno převzít řízení nebo částečně vypomoct s některými úkony řidiče.

Tento pojem se může zdát novodobý. Ve skutečnosti sahá do druhé poloviny 20. století, kdy se začaly testovat první autonomní vozidla. Dnes už se s pojmem autonomní vozidlo setkáváme docela běžně, a to nejen v teoretické rovině. Tyto vozidla jsou však stále ve fázi vývoje a testování. Jejich nasazení do běžného provozu závisí na regulacích, technických inovacích a přijetí veřejnosti. V současné době se s plně autonomními vozidly můžeme setkat na území některých státu USA, kde probíhá jejich testování pro komerční užití.

Celou myšlenkou autonomie dopravy je zvýšení její efektivity a bezpečnosti. Jde hlavně o snížení počtu dopravních nehod způsobených lidským selháním. V neposlední řadě jde také o zvýšení komfortu při cestovaní.

1.1 Úrovně autonomie vozidel

Autonomie je fenomén, který zásadně ovlivňuje současný svět a průmyslový sektor. Jedním z klíčových konceptů v oblasti automatizace respektive autonomie je systém hodnocení a klasifikace úrovní autonomie, který pomáhá lidem a organizacím lépe porozumět, do jaké míry je autonomie přítomna v konkrétním procesu nebo systému. Tato klasifikace úrovní autonomie poskytuje rámec pro posouzení, do jaké míry je lidský dohled a řízení nahrazeno strojními systémy a technologiemi. Úroveň autonomie vozidel definuje norma SAE J3016, kterou vytvořila Society of Automotive Engineers (SAE). Tato norma stanovuje klasifikaci autonomních funkcí a vozidel na základě míry automatizace a lidské interakce. Úrovně autonomie jsou rozděleny od úrovně 0 (žádná autonomie) po úroveň 5 (plně autonomní vozidlo). Následující podkapitoly se budou věnovat popisu těchto úrovní autonomie podle normy SAE [1, 2].

1.1.1 Úroveň 0

Autonomie ve vozidle není žádná, tzn. vozidlo je plně ovládáno řidičem a řidič je plně zodpovědný za řízení vozidla a veškeré aspekty s ním spojené. Jedná se stále o mnoho vozidel, které se dnes pohybují na pozemních komunikacích po celém světě. Vozidlo ovšem může být vybaveno prvky a systémy, které napomáhají řidiči vykonávat úkony řízení. Může se například jednat o systém nouzového brzdového asistenta, systém sledování slepého úhlu nebo o adaptivní světlomety [1, 2, 3].

1.1.2 Úroveň 1

Nejnižší úroveň autonomie, tzn. úroveň asistence řidiče. Je taková úroveň, kde ve vozidle jsou přítomny některé adaptivní funkce, které mohou řidiči asistovat. Řidič je zde stejně jako v předchozí úrovni plně zodpovědný za řízení vozidla a má dohled nad vozidlem. V případě potřeby musí řidič být kdykoliv připraven převzít kontrolu nad vozidlem. Do úrovně 1 řadíme systém udržovaní automobilu v jízdním pruhu nebo adaptivní tempomat [1, 2, 3].

1.1.3 Úroveň 2

Druhou úrovní nazýváme částečnou autonomii řízení, to znamená, vozidlo využívá pokročilé asistenční systémy. Oproti předchozí úrovni je zde, ale vyšší míra samostatného rozhodování. Vozidlo je nejen schopno samo řídit, dokáže však regulovat i svou rychlost, a to jak samostatným přidáváním plynu, tak i schopností samostatně brzdit. Jako v předchozích úrovních je i zde nutnost řidiče, který kdykoliv v případě potřeby musí převzít plnou kontrolu nad vozidlem. Mezi systémy úrovně 2 řadíme systém asistenta pro jízdu v koloně či systém automatizované pomoci při práci na silnicích a v přeplněné dopravě [1, 2, 3].

1.1.4 Úroveň 3

Jde o úroveň podmíněné autonomie. Třetí úroveň je oproti druhé značný technologický skok. V této úrovni autonomie je již vozidlo na základě senzorů a kamer schopno detekovat a následně vyhodnocovat okolní prostředí. Můžeme říct, že vozidlo je schopno samo rozhodovat o úkonech řízení. Třetí úroveň je pořád podmíněna přítomností člověka, který v případě problému musí zasáhnout do rozhodnutí vozidla. Právě tuto úroveň testují ve svých vozidlech největší automobilové společnosti. Bohužel při složitosti legislativního procesu a dalších aspektech spojených s běžným použitím, např. problém odpovědnosti v případě nehody, není úroveň 3 ještě

plně používaná. Jako příklad technologií zde najdeme plně automatizovanou jízdu po dálnici [1, 2, 3].

1.1.5 Úroveň 4

Vysoká autonomie řízení. Vozidla ovládají veškeré úkony spojené s řízením. V případě problému dokáže systém řízení vozidla sám zasáhnout. Řízení zde již nevyžaduje jakoukoliv lidskou interakci mimo krizové situace způsobené vlivem výpadku systému nebo nepřízni počasí, ale pokud se člověk rozhodne může řízení nad vozidlem převzít. Klíčovým rozdílem od předchozí úrovně je možnost jízdy se zasunutím ovládacích pedálů a volantu. Úroveň 4 je zatím čistě v testovací fázi, které probíhá zejména v některých státech USA. Samotné testování provází přísné podmínky jako omezení rychlosti těchto vozidel a omezení samotné testovací oblasti. V testovací fázi, to zřejmě nějakou dobu zůstane, dokud se nevyvine vhodná infrastruktura a platná legislativa řešící problémy autonomního řízení. Největší uplatnění této úrovně nyní najdeme v sytému sdílených taxíků známých jako geofencing [1, 2, 3].

1.1.6 Úroveň 5

Úplná autonomie, zatím nejvyšší definovaná úroveň autonomního řízení. Vozidla tohoto typu už vůbec nebudou vyžadovat lidskou interakci. V těchto vozidlech rovněž nenajdeme ani dosud běžné ovládací prvky jako řídící pedály nebo volant. Schopnosti a pohyb těchto vozidel nebude nijak omezen a budou schopna se sama pohybovat v městském prostředí i po dálnici. Interiér samotný bude spíše připomínat kancelář, obývací pokoj či ložnici. Nebudou chybět prvky jako televize, postel či možnost občerstvení. Rovněž se zde uvažuje nad odpadnutím vlastnictví automobilu fyzickou osobou. Auto si bude moci přivolat kdokoliv pomocí telefonu kdykoliv bude potřeba. Dokonale autonomní vozidla jsou spíše vizí budoucnosti, momentálně neexistuje vozidlo dosahující této úrovně autonomie. Problém využití úplné autonomie je hlavně v bezpečnosti, a to jak fyzické, tak hlavně kybernetické. Tyto problémy by se daly shrnout tak, že plně autonomní vozidla nebudou veřejností přijata dokud nebude zajištěna jejich bezpečnost, stejně jako na úrovni komerční letecké a vlakové dopravy [1, 2, 3].

1.2 Společnosti testující autonomní vozidla

Rychlý pokrok v oblasti autonomních vozidel se stal jedním z nejvýznamnějších milníků v automobilovém průmyslu a průmyslu mobility jako celku. V posledních letech se z technologického snu stala realita a díky tomu je možné sledovat zásadní změny

v tom, jaké bude v budoucnosti cestovaní a přeprava. Tato revoluce v oblasti autonomních vozidel by však nebyla možná bez odhodlání a inovací několika význačných společností, které se staly průkopníky v tomto odvětví.

Společnosti, které se věnují vývoji a provozování autonomních vozidel, přicházejí s novými technologiemi a koncepty, které nejen mění způsob, jakým se pohybujeme, ale také ovlivňují naši budoucnost v oblasti mobility. Těchto společností zabývajících se vývojem, testováním a provozováním autonomních vozidel najdeme na trhu desítky. Mnohdy se jedná o samotné automobilky, které přichází se svými projekty v oblasti autonomní dopravy. Nejednou se však na trhu proslavily začínající startupy, které následně odkoupili známí technologičtí giganti jako firma Google či Amazon.

Tato kapitola se podrobněji zabývá některými z těchto společností. Prozkoumává jejich role a příspěvky k rychle rostoucímu odvětví autonomních vozidel. Výběr společností byl však omezen na společnosti, které již vyrobily vozidlo, jež je plně nasazeno v testovací fázi nejméně na 4. úrovni autonomního řízení [4].

Waymo

Waymo[1] je dceřiná společnost přidružená k ALphabet Inc. spadající pod Google. Jde o jednu z prvních společností, která se začala intenzivně zabývat moderním autonomním řízením, a to jak jeho vývojem tak testováním. Společnost Waymo už v roce 2015 provedla čistě autonomní jízdu na běžné pozemní komunikaci, a to jako vůbec první na světě. V roce 2019 pak firma spustila projekt Waymo One. Jedná se o veřejnou autonomní taxislužbu v metropoli Phoenix. V roce 2022 se posunuli u své taxislužby na úroveň 4 autonomního řízení tzn. bez přítomnosti člověka na sedadle řidiče. Nyní společnost poskytuje své služby Waymo One ve třech městech v USA, a to v Phoenixu, Los Angles a San Franciscu. Společnost Waymo rovněž dominuje v oblasti autonomní nákladní dopravy pod hlavičkou projektu Waymo Via, kterou provozuje ve státě Texas pro několik vybraných komerčních zákazníků [5, 6]. Autonomní vozidlo firmy Waymo zobrazuje obrázek1.1

Cruise Automation

Cruise Automation[2], která je od roku 2016 součástí koncernu General Motors se zabývá vývojem a provozem autonomních automobilů na čtvrté úrovni autonomie. V roce 2020 získali povolení provozovat autonomní automobily bez nutnosti řidiče na palubě. První autonomní vozidla této společnosti byla nasazeny ve městě San Franciscu, kde během pandemie Covidu-19 ve spolupráci s potravinářskou společností

[1] Stránky společnosti Waymo: `https://waymo.com`

[2] Stránky společnosti Cruise Automation `https://www.getcruise.com`

Obr. 1.1: Autonomní vozidlo společnosti Waymo. (Zdroj: waymo.com, 2023)

Wallmart rozvážela potraviny nakaženým pacientům. Od roku 2022 společnost rovněž spustila službu noční autonomní taxislužby, a to v centru města Austin a Houston (Texas). Cruise chce mimo osobní automobil přidat do služeb svých autonomních taxi také minibusy. Společnost Cruise však nyní pozastavila veškerá nasazení svých autonomních vozů z důvodu mnoha problémů, které jejich auta během roku způsobila. Mezi problémy patřilo blokovaní vozu integrovaného záchranného systému (IZS), srážka s autobusem, prudké brzdění vozidel, vjezd do uzavřené oblasti a nakonec uvěznění chodkyně pod zadními koly automobilu [7, 8]. Vozidlo společnosti Cruise je vyfotografováno na obrázku 1.2

Obr. 1.2: Autonomní vozidlo společnosti Cruise. (Zdroj: getcruise.com, 2023)

Zoox

Zoox[3] je společnost, nyní ve vlastnictví Amazonu, která se zabývá vývojem autonomních vozidel na úrovni 4. Na počátku roku 2023 získal Zoox povolení pro testování autonomních vozidel bez nutnosti člověka jako řidiče ve městě Froster City v Kalifornii. Zároveň v tomto roce uskutečnili vůbec první komerční jízdu s běžným pasažérem. Firma Zoox se následně rozšířila i do města Los Angeles. Zoox stejně jako Waymo hodlá vést souboj o prvenství v oblasti autonomní dopravy včetně provozování veřejné autonomní a bezpečné taxislužby [9]. Autonomní automobil společnosti Zoox zobrazuje obrázek 1.3.

Obr. 1.3: Autonomní vozidlo společnosti Zoox. (Zdroj: zoox.com, 2023)

Aurora

Aurora[4]je technologickou společností zaměřenou na autonomní řízení, která spolupracuje s výrobci automobilů na vývoji autonomních systémů, včetně vozidel na úrovni 4. Jejím hlavním cílem je automatizace nákladní dopravy. Na její autonomní nákladní vozidla momentálně můžeme narazit na silnicích a dálnicích v americkém státě Texas, kde přepravují zboží pro několik vybraných komerčních zákazníků. Sekundárně se společnost zabývá vývojem autonomní taxislužby, kterou chce provozovat na okresních silnicích a dálnicích na trasách mezi letištěm a městem [6, 10]. Na obrázku 1.4 lze spatřit autonomní kamion společnosti Aurora.

[3]Stránky společnosti Zoox: `https://zoox.com`
[4]Stránky společnosti Aurora Innovation: `https://aurora.tech`

Obr. 1.4: Autonomní kamion společnosti Aurora. (Zdroj: Aurora.tech, 2023)

1.3 Systémy pro detekci okolního prostředí

Pro detekci okolního prostředí využívají autonomní vozidla různé kombinace a druhy systémů a senzorů, aby dosáhla co nejlepší přesnosti a spolehlivosti. Tyto prvky sbírají data o okolním prostředí, které následně odesílají do palubního počítače, který je s pomocí strojového učení a umělé inteligence vyhodnotí a tím určí chování autonomního vozidla. To umožňuje autonomním vozidlům rozpoznávat překážky, sledovat vozovku, číst dopravní značky a reagovat na dynamické situace na silnicích. Některé senzory slouží jako zrak automobilu jiné mají funkci bezpečnostní. Kombinace dat z více senzorů nazýváme tzv. fúzí senzorů. Následující podkapitoly popisují nejzákladnější druhy senzorů, které využívají autonomní vozidla patří zde kamery, radary, LiDARy a GPS [11, 12].

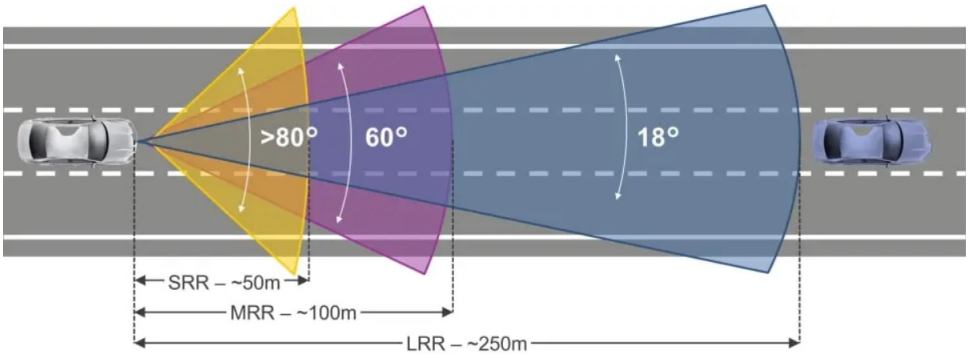
1.3.1 Kamera

Kamera je zařízení, jež pomocí čočky dokáže zachycovat světlo a tím umožňuje snímat obrazová data okolo sebe, čímž poskytuje vozidlu schopnost vnímat a interpretovat své okolí. Světlo procházející čočkou dopadá na snímač, který tyto světelné signály obrazu převede na elektrický signál a odešle palubnímu počítači pro další zpracování. Software v palubním počítači dále používá zpracovaný signál pro různé úkony, jako rozpoznání dopravních značek či ke sledování vozidel a chodců. Pomocí těchto dat vozidlo mění své rozhodování v daných situacích například přidáním nebo snížením rychlosti, změnou jízdního pruhu. Samotné použití kamer jako senzorů nestačí z důvodu změn světelných podmínek například při výjezdu z tunelu, a proto se kamery používají v kombinací s dalšími druhy senzorů [11, 13, 14].

1.3.2 Radar

Jde o elektromagnetický systém složený z vysílače a přijímače, který dokáže do svého okolí vysílat rádiové vlnění. Vlnění se pohybuje vzduchem, dokud nenarazí na objekt od kterého se odrazí zpět do přijímače radaru. Díky schopnosti odrazu těchto vln dokáže radar vyhodnotit pozici i vzdálenost od objektu a zároveň tak může určit rychlost či pohyblivost objektu. Radary jsou konstruovány tak, aby na jejich funkčnost nemělo vliv počasí, vlhkost, teplota nebo střídání dne a noci. Každý radar vysílá rádiové vlny pod jinou frekvencí.

Díky radaru může automobil určit vzdálenost a pozici okolních objektů, nejběžnější umístění radaru je přední a zadní nárazník vozidla. V autonomních vozidlech nalezneme radary krátkého dosahu pro sledování mrtvého úhlu, středního dosahu, který slouží pro detekci překážek i dlouhého dosahu pro určení vzdálenosti a správnou funkci brzdového asistenta [11, 13, 15]. Obrázek 1.5 zobrazuje druhy radaru s jejich průměrným dosahem, které autonomní vozidla využívají.



Obr. 1.5: Druhy radaru a jejich dosah. (Zdroj: semiengineering.com, 2023)

1.3.3 Light Detection and Ranging (LiDAR)

LiDAR je optická technologie podobná radaru. Od radaru se liší tím, že místo rádiových vln vyzařuje světelné paprsky na bázi laserového spektra. Na základě vyslaných paprsků, které se odrazí od objektu zpět do přijímače dokáže za pomocí této technologie skenovat okolní prostředí a vytvořit jeho 3D model a vypočítat vzdálenost od objektů. Princip určení vzdálenosti funguje na výpočtu rozdílu času vyslání paprsku a přijetí odraženého paprsku. Technologie LiDAR je nejspolehlivější pro určení fyzikálních popisů jako poloha, rychlost a tvar objektů. Na rozdíl od radaru technologii LiDAR ovlivňuje nepříznivé počasí jako mlha, sníh a déšť.

Tato technologie je v autonomní oblasti vozidel klíčová pro vnímaní automobilu, kdy díky ní vozidlo získá přehled o stacionárních i pohybujících se objektech.

Pro správnou funkci autonomního vozidla je nejdůležitější zpracovat tyto data: detekce objektu, sledování, rozpoznávání a predikce pohybu, toto vše má na starosti technologie LiDAR [11, 13, 16].

1.3.4 Global Positioning System (GPS)

Systém známý pod zkratkou GPS, umožňuje navigaci pokrývající celý svět. Systém funguje na bázi 31 družic, které se pohybují na oběžné dráze Země a tvoří společnou propojenou síť. Tyto družice vysílají rádiové signály, které na povrchu Země zachytí jednotlivé přijímače a díky tomu můžeme určit přesný čas a polohu kdekoliv na světě. Signál GPS nese základní informace skládající se z pseudonáhodného ID kódu požadavku, dále z efemeridických údajů o pozici družic s aktuálním datem a časem, poslední informací jsou data z almanachu, který poskytuje přijímači informace o následujícím pohybu družice. Aplikuje se zde výpočet rozdílu času mezi přijetím signálu a jeho vysláním. Každý přijímač využívá systému čtyř družic pro určení parametrů zeměpisné šířky, zeměpisné délky, nadmořské výšky a času.

Autonomní vozidla mohou využívat GPS pro vlastní navigaci po trase a určení své vlastní polohy. V operačním středisku tak mohou operátoři sledovat vozidlo v reálném čase a v případě poruchy nebo nehody mohou zasáhnout [14, 17, 18].

1.4 Komunikace vozidel

Základem úspěchu autonomní dopravy je schopnost efektivní a vzájemné komunikace jednotlivých vozidel mezi sebou i s prvky dopravní infrastruktury a ostatními účastníky silničního provozu. Tato komunikace spolu s telematikou hraje klíčovou roli v zajištění bezpečné, efektivní a plynulé integrace autonomních vozidel do dopravní sítě. Komunikace v autonomních vozidlech může být rozdělena do dvou hlavních kategorií vnitřní komunikace (Intra-Vehicle Communication) a vnější komunikace (Inter-Vehicle Communication).

1.4.1 Vnitřní komunikace

Mezi první typ komunikace můžeme zařadit vnitřní komunikaci, která označuje komunikační systémy uvnitř autonomního vozidla. Ta probíhá v rámci jednoho konkrétního vozidla. Vozidlo takto sdílí informace mezi jednotlivými senzory a systémy s elektrickými řídicími jednotkami (Electronic Control Unit), které za pomocí akčních členů a modulů provádí jednotlivé jízdní akce. Mezi jednotky ECU můžeme například zařadit řídicí jednotku motoru, jednotky jízdních charakteristik kam patří

systémy ABS či ARS anebo jednotky určené pro jízdní komfort, jako ovládání klimatizace a polohy sedadel. Na základě tohoto spojení mohou spolu tyto senzory a řídicí jednotky spolupracovat, komunikovat a přizpůsobovat jízdu vozidla, tak aby byla co nejbezpečnější a nejefektivnější. K tomu je zapotřebí využití sběrnic a komunikačních protokolů, které umožňují propojení jednotlivých prvků vozidla. Níže je uveden aktuální přehled sběrnic a protokolů používaných v autonomních vozidlech [20, 21].

Controller Area Network (CAN)

CAN je univerzální standardizovaný protokol sběrnice pro výměnu informací v automobilovém průmyslu. Slouží pro komunikaci mezi elektronickými komponenty s řídícími jednotkami nebo také pro diagnostiku vozidla. Jedná se o síť typu peer-to-peer, kde zprávy jsou zapouzdřeny do rámců. Rámce jsou za pomocí vysílání typu broadcast odesílány do všech uzlů, které komunikují se sběrnicí CAN, tyto rámce následně přijímají jednotlivé řídicí jednotky, které na základě dat provádí další akce. Jakýkoliv uzel může odesílat a přijímat zprávy. Každý rámec obsahuje jedinečné identifikační označení ID, které identifikuje cílový uzel, dále prioritu uzlu a data. Maximální rychlost sběrnice 1 Mb/s. Mezi hlavní výhody nasazení sběrnice CAN patří snížení délky a počtu kabeláže ve vozidle. Mezi systémy využívající sběrnici CAN patří elektronické ovládání sedadel a zrcátek, odpružení, brzdy, řízení systému karoserie, ABS či řízení trakce [20, 21].

FlexRay

Standard pro výměnu informací v automobilovém průmyslu. Vhodný pro kritické systémy a senzory v autonomních vozidlech, díky své spolehlivosti a časové synchronizaci. FlexRay je síť s více hlavními uzly, ty jsou organizovány do clusterů. Každý uzel má schopnost vysílat a přijímat data nezávisle. Jednotlivé clustery spolu spolupracují. Informace jsou zapouzdřeny do rámců. Veškerý komunikační provoz včetně řídicích jednotek je tak synchronizovaný s globálním časem a probíhá na základě nastavených časových intervalů dle potřeby. Oproti CAN disponuje vyšší rychlostí při přenosu a navíc podporuje deterministické řízení a zajišťuje synchronizaci přenosu dat mezi různými uzly v systému. Nevýhodou je výrazně dražší cena a složitost pro vozidlovou síť. Maximální rychlost 10 Mb/s. Technologii FlexRay je možné využít pro hnací ústrojí, adaptivní tempomat, ABS, parkovacích senzory, řízení trakce nebo brzd [20].

Media Oriented Systems Transport (MOST)

MOST je protokol navržený pro přenos multimediálních dat v automobilových systémech. Je často používán pro propojení různých multimediálních zařízení jako audio a video systémy ve vozidle. MOST je zapojen do topologie typu hvězda. Existuje tedy jeden hlavní uzel, ke kterému jsou připojeny ostatní uzly. Data jsou přenášena za pomocí optických vláken od hlavního uzlu k jednotlivým uzlům. MOST podporuje vysoké přenosové rychlosti, což je důležité pro rychlý a bezproblémový přenos multimediálních dat. Maximální rychlost je až 150 Mb/s. Hlavní využití protokolu je pro infotainment systémy jako GPS či virtuální zábavu v podobě audio a video přenosu [20].

Ethernet

Ethernet je standardizovaná sběrnice pro komunikaci na internetu. S postupným vývojem moderních aplikací, které se stávaly náročnější pro svou vysokou šířku pásma, bylo potřeba vymyslet nástupce sběrnic CAN či MOST, které se stávají nedostačujícím. Pro svou širokou dostupnost a vysokou přenosovou rychlost se jim stal Ethernet. Současně je tedy považován za protokol budoucnosti ve vnitřní komunikaci. Nabízí větší šířku pásma a větší přenosovou rychlost. Je schopen přenášet velké objemy dat. Ethernet je nejčastěji implementován jako vysílaní typu broadcast. Každý uzel v síti tak může přijímat data a vysílat je všem ostatním uzlům. Současný standart je Ethernet 100BASE-T1 (IEEE), který poskytuje rychlost až 100 Mb/s. Ten je však stále ve vývoji a předpokládá se, že bude disponovat ještě větší přenosovou rychlostí. Ethernet ovšem má i své nevýhody v podobě vyšší ceny a špatné odolnosti proti vysokofrekvenčnímu rušení. Automobilový Ethernet lze využít pro pokročilé asistenční systémy řidiče, infotainment, diagnostiku a hlavně jako páteřní síť [20].

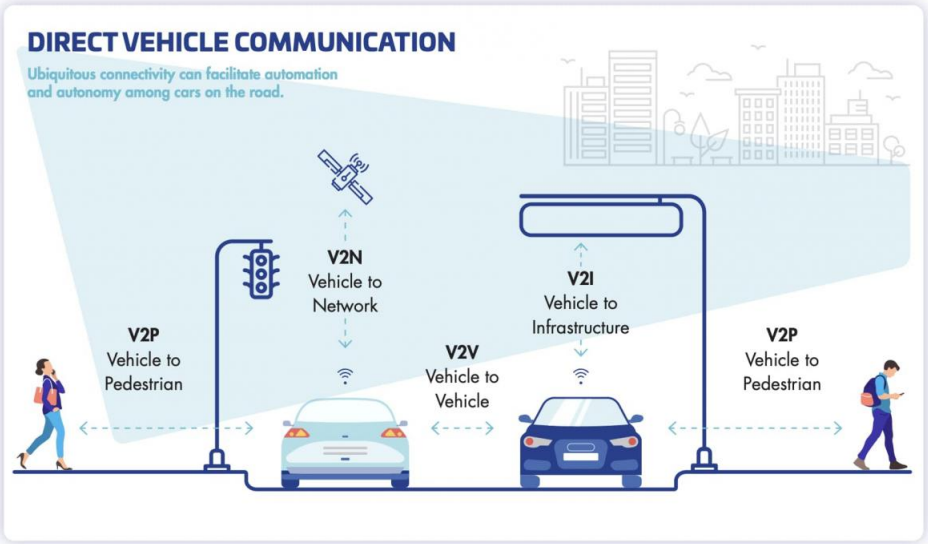
Bezdrátové sítě

V posledních letech se ve vozidlech běžně setkáváme i s bezdrátovým typem vnitřní komunikace, a to prostřednictvím technologií jako je Bluetooth a WiFi. Tyto technologie umožňují řidiči vytvořit ve vozidle lokální síť, kterou ovládá prostřednictvím mobilního telefonu. Pomocí této lokální sítě může řidič komunikovat s vozidlem a jeho funkcemi. Díky tomu si lze vozidlo přizpůsobit dle svých potřeb a dochází tak k zlepšení zážitku z jízdy. Jako příklad zde můžeme uvést systém In-Vehicle Infotainment (IVI). Ten využívá technologie Bluetooth pro spárování mobilního telefonu s vozidlem, díky tomu může řidič uskutečňovat a přijímat hovory pomocí dotykové obrazovky umístěné v palubní desce vozidla. Technologie rovněž umožňuje přehrání audio souborů, zobrazení obrázků či přehrání video souborů, jež jsou uloženy v paměti telefonu. V moderních vozidlech můžeme najít systém detekce únavy,

který pomocí nositelné elektroniky, například fitness náramku nebo chytrých hodinek, monitoruje krevní tlak a zároveň srdeční tep řidiče. V případě naměření hodnot vykazujících únavu může tento systém řidiče upozornit. Technologie WiFi je rovněž ve vozidlech rozšířená. Moderní vozidla disponují WiFi routerem, ke kterému se může posádka vozidla připojit za pomocí telefonů či tabletů a získat tak přístup k internetu [21].

1.4.2 Vnější komunikace

Vnější komunikace je technologický koncept, který umožňuje vzájemnou interakci a výměnu informací vozidla s ostatními vozidly na silnici nebo s prvky dopravní infrastruktury či s dalšími zařízeními a systémy. Tento koncept vznik v době širšího rozšiřování technologie sítí páté generace (5G). Vnější komunikaci autonomních vozidel dále rozdělujeme do čtyř základních typů, a to komunikace: Vozidlo-vozidlo (Vehicle-to-Vehicle), vozidlo-infrastruktura (Vehicle-to-Infrastructure), vozidlo-chodec (Vehicle-to-Pedestrian) a vozidlo-síť (Vehicle-to-Network) [19, 21]. Všechny tyto základní typy tvoří jednu výslednou komunikaci, kterou nazýváme vozidlo-všechno (Vehicle-to-Everything), jenž je vidět na obrázku 1.6



Obr. 1.6: Koncept V2X komunikace. (Zdroj: automatizace.hw.cz, 2023)

Vozidlo-vozidlo (V2V)

Komunikace V2V je založena na síti **Ad-hoc**[5], která umožňuje vzájemnou komunikaci a výměnu informací mezi jednotlivými autonomními vozidly. Vlastností V2V je odstranit závislost na mobilních sítích a na sítích třetích stran z důvodu omezenosti šířky pásma. Tato forma komunikace má potenciál výrazně zvýšit bezpečnost na silnicích, optimalizovat provoz a přispět k efektivnímu řízení dopravy.

V2V funguje na základě komunikace mezi senzory, které jsou zabudovány ve vozidlech. Senzory ve vozidlech sbírají různé informace jako je rychlost, poloha, směr jízdy, stav a typ vozidla. Tyto informace jsou pak v reálném čase odesílány na palubní jednotku On-Board Unit (OBU), která umožňuje sdílet tyto informace s ostatními vozidly ve svém okolí a naopak od ostatních vozidel informace přijímat. Celý tento systém podléhá přísnému omezení kvality služeb (Quality of Service). [19, 21, 22, 23].

Výměnu informací v reálném čase má ve V2V komunikaci na starosti standardizovaná technologie Dedicated Short-Range Communication (DSRC). Jedná se o bezdrátový komunikační protokol pokrývající okruh 360° s dosahem do 1000 metrů, který garantuje dobu zpoždění menší než 5 ms. Tento protokol tedy disponuje vhodnými vlastnostmi pro aplikaci v autonomních vozidlech. Protokol navíc umožňuje v případě nebezpečné situace anebo nehody na silnici šířit nouzové varování. USA se v roce 2020 zavázaly aplikovat tento protokol do všech nových vozidel vyrobených na domácí půdě. Aplikaci tohoto protokolu do svých vozidel podpořily i další státy jako Japonsko, Austrálie a vybrané státy Evropy. Ačkoliv k používání DSRC se zavázaly některé státy, někteří výrobci automobilů začali hledat i alternativní technologie, které by ve svých vozidlech mohly využít, patří mezi ně například Cellular-V2X. To je bezdrátová komunikační technologie, která umožňuje vzájemnou komunikaci mezi vozidly a různými prvky prostředí, včetně infrastruktury, chodců a dalších vozidel. Klíčovým prvkem této technologie je využití mobilních sítí pro vzájemnou výměnu dat. To při nástupu 5G mobilní technologie nabízí výhodnější šířku pásma, datový tok a větší podporu mobility [22, 23].

Pro umožnění komunikace mezi vozidly je tedy nezbytné dosáhnout rychlé a efektivní výměny důležitých informací. Tento požadavek vyžaduje vyřešení několika technických výzev, včetně minimalizace časového zpoždění přenosu dat, zajištění rychlosti přenosu dat a vysoké spolehlivosti komunikace.

Systémy, které využívají tuto komunikaci rozdělujeme na kritické a nekritické. Mezi kritické systémy můžeme například zařadit systém nouzových brzdových svě-

[5]Síť Ad-hoc je bezdrátová nezávislá síť určená pro malou vzdálenost, která se vytváří dynamicky bez potřeby pevné infrastruktury jako je centrální řídicí jednotka nebo přístupový bod. Je charakterizována autonomními uzly, které jsou si v síti rovny a mohou spolu vzájemně komunikovat. Spojení se sestavuje jen v případě potřeby.

tel, systém snímaní před nehodou nebo varování před slepou křižovatkou. Naopak do nekritických systému spadají někteří asistenti, například asistent pro zastavení, asistent pro změnu jízdního pruhu, vyhledávač silničních služeb. Když vozidla vzájemně komunikují mohou sdílet informace o svých pohybech, což umožňuje lepší predikci chování a rychlejší reakci na možné nebezpečí. Například, pokud jedno vozidlo zaznamená náhlé brzdění, může tuto informaci sdílet s okolními vozidly, které pak mohou varovat své řidiče nebo dokonce aktivovat autonomní brzdy pokud je to nezbytné. Dále vzájemná komunikace pomáhá snížit dopravní zácpy [22, 23].

Vozidlo-infrastruktura (V2I)

Je komunikační model umožňující vozidlu komunikovat se silniční infrastrukturou v reálném čase. Vozidlo je schopno komunikovat s prvky inteligentního transportního systému (ITS) jako jsou radary, semafory, dopravní značení nebo parkovací automaty. Jedná se o obousměrnou komunikaci využívající bezdrátového přenosu, která je realizovaná speciálním hardwarem, firmwarem a softwarem. Cílem je vytvořit bezpečnější a efektivnější prostředí pro vozidla na silnici tím, že vozidlům poskytuje klíčové informace o okolí a infrastruktuře. Získané údaje z vozidel jako rychlost a poloha se odešlou na centrální server, který zpracovává aktuální údaje ostatních vozidel a umožní nám tak například určit nejrychlejší cestu z aktuální polohy do námi zvoleného cíle, upozornit na snížení rychlosti v nebezpečných úsecích a ostrých zatáčkách nebo podat informace o vzniku dopravních kolon. V2I dále umožňuje dynamicky řídit světelnou signalizaci čehož můžou využít vozy IZS, pro bezpečné projetí křižovatkou.

Pro komunikaci V2I se používá stejná komunikační technologie jako v komunikaci V2V, a to protokol DSRC nebo technologie Cellular-V2X. Mezi hlavní komponenty, které zpracovávají data nebo poskytují zprávy patří palubní jednotka OBU na straně vozidla. Na straně infrastruktury má výměnu zpráv na starost jednotka infrastruktury RSU. Obě tyto jednotky tvoří radiový vysílač s DSRC protokolem nebo Cellular-V2X technologií [19, 21, 24, 25].

Vozidlo-chodec (V2P)

V2P je forma komunikace v autonomní dopravě, která umožňuje vozidlům komunikovat se speciální skupinou účastníků silničního provozu, jež tvoří převážně cyklisté, chodci a dvoustopá motorová vozidla. Dohromady tyto skupiny lze označit zkratkou Vulnerable Road Users (VRUs). Cílem této komunikace je zvýšit bezpečnost chodců, cyklistů a poskytovat jim informace o pohybu vozidla a jeho záměrech. Toho lze prakticky využít pro bezpečnější přecházení silnice či odbočování cyklistů. Tito účastníci

s vozidly vzájemně komunikují prostřednictvím svých chytrých telefonů a nositelné elektroniky.

V2P komunikace musí být schopna rozlišit rychlost, mobilitu a způsob jízdy. Tím je myšleno, že vozidlo musí být schopno vyhodnotit různé aspekty pohybu, ať už cyklisty, maminky s kočárkem nebo dětí. Například cyklista na silničním kole se může v městském prostředí pohybovat stejně rychle jako motorkář. Jako další příklad lze uvést zastavení cyklisty v křižovatce před semaforem, když svítí červená na semaforu a zároveň přecházení chodců nebo cyklistů vedle kola přes přechod této křižovatky. Dále musíme zohlednit pohyb chodců, ti mohou chodit samostatně nebo ve větších organizovaných či neorganizovaných skupinkách.

Bezpečnost chodců pokrývá tzv. V2P systém prevence před nehodou. Ten je realizovaný sítí Ad-hoc nebo za pomocí mobilních sítí. Výměna informací funguje na systému bezpečnostních zpráv, které nejčastěji obsahují informace jako poloha, rychlost a směr jízdy. Jako komunikační protokol je zde využíván DSRC, který byl převzatý z komunikace V2V. Tomu ovšem může do budoucna konkurovat již zmíněná Cellular-V2X technologie [26, 27].

Vozidlo-síť (V2N)

V2N představuje formu komunikace, ve které autonomní vozidla komunikují s externími sítěmi, jako jsou mobilní sítě, cloudové platformy a centrální řídící systémy. Cílem je poskytovat vozidlům širší kontext informací a optimalizovat jejich chování na základě dat získaných z externích zdrojů. Cloud navíc může vozidlům zpřístupňovat další aplikace, které může posádka vozu užít pro zpříjemnění cesty. Další využití této komunikace je možnost velkého shromažďování dat, které následně lze využít pro správné nastavení intervalů semaforů či železničních signalizačních zařízení, pro nastavení rychlostních limitů při změnách počasí a celkovému zlepšení silniční infrastruktury. V2N pro komunikaci využívá technologii Cellular-V2X. Jako alternativní možnost se v tomto kontextu zkoumala technologie mmWave. Jde o technologii fungující na bázi milimetrových vln, která nabízí dostupnější šířku pásma než mobilní sítě. S nástupem mobilních sítí 5G se však od této možnosti postupně ustupuje. 5G technologie totiž nabízí vysokou rychlost přenosu a nízké zpoždění. Dnes už ve většině moderních městských aglomerací najdeme vysílací věže podporující 5G, a proto se tato komunikace stává stále dostupnější. Výhodou je rovněž poměrně nízká cena a možnost propojení dalších zařízení a infrastruktury do tzv. internetu věcí (IoT) [21, 28, 29].

2 Kybernetická bezpečnost autonomních vozidel

Autonomní doprava, s využitím pokročilých technologií a umělé inteligence, se stává klíčovým prvkem moderního dopravního systému. Zatímco přináší inovativní přístup k mobilitě a slibuje zvýšení bezpečnosti a efektivity, mohou tyto technologie představovat riziko, a to v podobě kybernetické bezpečnosti. Autonomní systémy mají zranitelnosti, které mohou být využity ke kybernetickému útoku. Cílem kybernetické bezpečnosti je tedy zajistit integritu, dostupnost a důvěrnost systémů a minimalizovat rizika spojené s kybernetickým útokem.

První kybernetické útoky se datují k roku 2002, kdy bylo zneužíváno modifikace řídících jednotek motoru. Postupem času byly zaznamenány útoky pomocí technologie Bluetooth či FM rádia. Přelomovým rokem se v oblasti kybernetické bezpečnosti v automobilovém průmyslu však stal rok 2015. V tomto roce se dvěma etickým hackerům (Charlie Miller a Chris Valasek) za pomocí notebooku podařilo vzdáleně převzít kontrolu nad vozidlem Jeep Cherokee, které řídil redaktor časopisu Wired. Dvojice využila zranitelnosti v dotykovém displeji se softwarem Uconnect, který se v automobilu nacházel. Hackeři dokázali nejenom ovládat sílu nebo teplotu klimatizace, přepínat rádiové stanice či zapínat odstřikovače čelního skla. Dokonce mohli autu na dálku vypnout motor nebo naopak přidat plyn a následně odpojit brzdy. Tento demonstrační kybernetický útok následně zveřejnili. Výsledkem útoku bylo zapojení amerického úřadu pro silniční bezpečnost (NHTSA) do vyšetřování celého incidentu. Tento incident měl obrovský dopad na kybernetickou bezpečnost v automobilovém průmyslu a jeho řešení do budoucna. Kybernetické zabezpečení vozidel a souvisejících informačních systémů se tedy stalo jednou z priorit výrobců chytrých vozidel [30].

Hlavními problémy kybernetické bezpečnosti v automobilovém průmyslu jsou zranitelnosti v automobilové komunikaci. Tyto zranitelnosti mohou způsobovat různé aspekty. Aktuální největší zranitelnosti nacházíme v oblastech omezené konektivity vozidla, omezeného výpočetního výkonu a v podobě nových scénářů útoků. Omezená konektivita nám znemožňuje provádět aktualizace softwaru na dálku proti nově vzniklým kybernetickým útokům. Omezený výpočetní výkon představuje problém v zavádění bezpečnostních opatření proti kybernetickým útokům z důvodu nízké výkonosti sběrnic a protokolů. Jelikož je kybernetická bezpečnost v autonomní dopravě poměrně novým tématem, tak je náchylná na nové scénáře útoku. Ty představují vznik nových hrozeb či nových vstupních bodů pro provedení kybernetického útoku, kdy tyto scénáře nedokáží výrobci vozidel nijak předvídat. Následující podkapitoly se zaměřují na výše popsané oblasti zranitelností, typy kybernetických útoku v závis-

losti na typu komunikace a jejich obranu v podobě zavádění bezpečnostních opatření proti těmto útokům [31].

2.1 Vnitřní komunikace vozidla

Kybernetické útoky popsané v této kapitole se zaměřují na komunikační sítě uvnitř vozidel. Komunikační sítě propojují různé elektronické řídící jednotky (ECU), senzory a další komponenty, umožňující jim vzájemnou komunikaci pro efektivní řízení vozidla. Útoky mohou zahrnovat manipulaci či krádež osobních dat, zneužití softwaru a hardwaru, nebo dokonce převzetí ovládacích systémů vozidla. Jako zranitelnosti v těchto útocích se nám představují komunikační sběrnice jako CAN či Ethernet. Hlavním problém těchto komunikačních sběrnic je nedostatek zavedených obraných mechanismů na úkor výkonu a komunikační kapacitě [32, 33].

2.1.1 Zranitelnosti

Sběrnice CAN

Jako první zranitelnosti si zde uvedeme autentizaci a priorita rámců sběrnice CAN. Rámce neobsahují žádné autentizační metody pro označení zdroje. Útočníci mohou využít této zranitelnosti a modifikovat tak data obsažené v rámcích nebo měnit priority jednotlivých uzlů na vyšší, aby docílil nedoručení ostatních rámců. Řídicí jednotky nedokážou rozlišit pravé data od těch modifikovaných či falešných.

Mezi další zranitelnost můžeme zařadit absenci kryptografického zabezpečení rámců pomocí metody šifrování. Absenci šifrování komunikace mohou útočníci využít k analýze nebo odposlechu dat.

Zranitelnost sběrnice CAN můžeme najít i v podobě přístupu ke sběrnici pomocí jiných rozhraní. Jedná se o zranitelnosti rozhraní, které se používají pro zlepšení komfortu při jízdě. Jsou to například přehrávač CD, port USB, OBD port nebo některé telematické systémy. Z výčtu uvedených rozhraní je v dnešní době nejzranitelnější port OBD, který má přímý přístup ke sběrnici CAN. OBD port se používá pro diagnostiku vozidla a přeprogramování firmwaru. Útočník se tak s využitím chytrého telefonu či notebooku může, ať už drátově nebo bezdrátově, připojit na tento port a odposlouchávat tak přenášené zprávy. Zranitelné rovněž v tomto ohledu mohou být i technologie Bluetooth či WiFi, přes které se útočník může ke sběrnici dostat [32, 33].

Sběrnice Ethernet

Automobilový Ethernet ve výchozí implementaci přebírá mnoho již známých zranitelností ze sběrnice CAN. Ethernet neobsahuje zabezpečení pomocí šifrovaní, schází mu autentizace a zajištění integrity rámců. ECU komunikující za pomocí Ethernetu nemohou zajistit nepopíratelnost. Zranitelný se zde stává i samotný přístup k síti, a to v podobě připojení vlastního zařízení nebo získáním kontroly nad stávajícím zařízením. Útočník tak může rozšířit stávající síť o nové zařízení, například v podobě vlastního přepínače nebo přístupového bodu. Ve své novodobé podstatě zavedení Ethernetu do automobilové sítě ještě mnoho zranitelností nebylo odhaleno [34, 35].

2.1.2 Kybernetické útoky

Tabulka 2.1 poskytuje přehled nejznámějších útoků na sběrnice CAN, Ethernet, FlexRay. Podrobnější informace o zranitelnostech, útocích či protiopatřeních lze nalézt ve vědeckých článcích [31, 32, 33, 34, 35].

Název útoku	**Postihnuté sběrnice**
Frame sniffing	CAN
Frame falsifying attack	CAN
Injcetion attack	CAN, Ethernet, FlexRay
Replay attack	CAN, Ethernet, FlexRay
Man-in-the-midle attack	CAN, Ethernet, FlexRay
Denial of Services attack	CAN, Ethernet
Masquerading attack	CAN, FlexRay
ECU impersonation	CAN

Tab. 2.1: Přehled útoků na sběrnice

Snímání rámců (Frame Sniffing)

Jde o metodu, kdy škodlivé uzly mohou odposlouchávat ostatní rámce. Útočníci mohou těchto uzlů využít pro sledování a analýzu provozu, což může vést k odhalení dalších zranitelností. V této metodě dochází k průzkumu prostředí, a proto je základem pro další útoky, jako útok zfalšováním rámců, útok vstřikováním nebo využití dat pro reverzní inženýrství. Snímání rámců je poměrně obtížné odhalit přispívá k tomu pasivní povaha tohoto útoku.

Útok zfalšováním rámců (Frame falsifying attack)

Útočník může vysílané rámce modifikovat a podstrčit jim falešné data nebo vložit nesprávné hodnoty parametrů. Využívá se zde absence autorizace zdroje. Falešné rámce s podvrženými daty mohou oklamat jednotlivé řídicí jednotky. Útok je navíc velmi obtížné odhalit, protože pracuje jen s konkrétními falešnými daty, kterých se tento útok týká. V praxi díky tomuto útoku může útočník zfalšovat údaje na přístrojové desce nebo zobrazit kontrolku falešné poruchy v některém ze systému automobilu.

Útok vkládáním (Injection attack)

V útoku se využívá neposkytnutí ověření sběrnicí. Útočník může pozměnit frekvenci, množství a pořadí odesílaných rámců, což vede k abnormálnímu provozu sběrnice. Ten se projevu zvýšením rychlosti odesílaných zpráv. Výsledný datový provoz sběrnice lze monitorovat a generovat tak falešné zprávy, které způsobí ve vozidle neobvyklé chování ve prospěch útočníka.

Útok opakováním (Replay attack)

Útočník v tomto útoku využívá opakovaného odesílání platných rámců na sběrnici. Tím naruší fungování vozidla, řídicí jednotka navíc nemůže ověřit jestli přijaté rámce byly odeslány z oprávněného zdroje. Takto může útočník například odemknout stojící vozidlo, nastartovat a vozidlo odcizit.

Útok mužem uprostřed (Man-in-The-Midle-attack)

Útočník se připojí k síti, ve které se stává prostředníkem a odposlouchává komunikaci mezi jednotlivými uzly. MITM útok může být použit k odposlouchávání komunikace mezi senzory a řídícími jednotkami nebo k manipulaci s daty, která jsou přenášena mezi různými částmi vozidla. S využitím MITM útok se skupina výzkumníků nabourala do vozů značek BMW či Audi [36].

Útok maskováním (Masquerading attack)

Při útoku je využito zranitelnosti v podobě chybějící metody šifrování a autentizace zdroje. Útočník se tak vydává za legitimní uzel, což mu následně umožňuje odesílat falešné rámce nebo odposlouchávat komunikaci. Útok maskováním také může zahrnovat například falešné přihlášení do systému, aby útočník mohl získat kontrolu nad autonomním vozidlem nebo získat přístup k citlivým systémovým informacím.

Útok odmítnutí služeb (Denial of Services attack)

Vozidlo je díky rámcům sběrnice s nastavením priority zranitelná na útoky typu Denial of Services (DoS). Útočníkovi pouze stačí změnit priority, některých rámců na tu nejvyšší a ostatní rámce s prioritou nižší nebudou sběrnicí přijaty. Útočník rovněž může sběrnici zahltit velkým množstvím rámců a omezit tak její komunikaci nebo změnit, díky znalosti přenosové rychlosti, zvýšení času frekvence, což by mohlo vést k narušení správné činnosti některého ze snímačů.

Vydávání se za ECU (ECU impersonation)

V tomto útoku se využívá získaného přístupu k sběrnici a jejímu vysílání. Útočník, tak může přijímat a analyzovat veškerý provoz sběrnice. Díky informacím získaným z analýzy lze určit chování a informace jako ID nebo přenosovou rychlost konkrétní elektronické řídicí jednotky (ECU). Na základě získaných údajů může útočník simulovat chování ECU, což následně umožňuje vysílat data pod stejnými údaji. Výsledkem útoku je zvýšení frekvence zpráv a vyřazení konkrétní ECU jednotky z provozu schopného stavu.

2.1.3 Možnosti protiopatření

Hlavním faktorem, který je třeba zohlednit je omezená kapacita sběrnic. Proto veškerá opatření nesmí výrazně ovlivnit výkonost sběrnice. Protiopatření spočívá v zavedení mechanismů, které sběrnice CAN nebo Ethernet ve své výchozí implementaci neposkytují, zde patří šifrovaní a autentizace či detekce kryptografický útoků.

Kryptografické metody

Pro zajištění autentizačního mechanismu lze využít Message Authentication Code (MAC), který umožňuje ověřovat zprávy. K šifrování přenášených rámců se používá nejčastěji protokol symetrické kryptografie AES.

Systémy detekce průniku (Intrusion Detection Systems)

Systémy Intrusion Detection Systems (IDS) jsou bezpečnostní nástroje navržené k detekci podezřelého chování v počítačové síti. Cílem těchto systému je zachytit možné kybernetické útoky nebo jiné narušení zabezpečení. IDS rozdělujeme na dva hlavní typy na systém detekce anomálií a na systém detekce signatur. V kontextu autonomních vozidel se používá systém detekce anomálií, a to z důvodu různorodosti možných útoků. Systém detekce anomálií nám umožňuje odhalit přenos abnormálních rámců. Tyto systémy lze umístit na konkrétní řídicí jednotky nebo do komunikační sítě.

2.2 Útoky na komunikaci V2X

V této podkapitole byly prozkoumány zranitelnosti, kybernetické útoky a protiopatření komunikace V2X. Byly zde analyzovány hlavně protokoly, které jsou v této komunikaci využívány jako DSRC, Cellular-V2X s využitím technologie 5G.

2.2.1 Zranitelnosti

Obě technologie se používají pro bezdrátový přenos dat mezi různými komponentami silničního provozu. DSRC zranitelnosti spočívají v nepřítomnosti metody šifrování, nedostatečné odolnosti vůči rušení a útokům typu DoS. Cellular-V2X využívá mobilních sítí (LTE nebo 5G). Přebírá však některé zranitelnosti mobilních sítí jako absenci procesu šifrování, autentizace či narušení integrity.

2.2.2 Kybernetické útoky

Tabulka 2.2 poskytuje přehled nejznámějších útoků na technologie DSRC, Cellular-V2X, které se využívají v komunikaci V2X. Další informace o zranitelnostech, útocích či protiopatřeních lze nalézt ve vědeckých článcích [31, 37, 38].

Název útoku	**Postihnuté technologie**
Jamming attack	DSRC, Cellular-V2X
Man-in-the-midle-attack	DSRC, Cellular-V2X
Denial of Services attack	DSRC, Cellular-V2X
Black hole attack	DSRC
Spoofing attack	DSRC, Cellular-V2X
Malware	DSRC

Tab. 2.2: Přehled útoků na V2X komunikaci

Útok rušením (Jamming attack)

Je cílené rušení signálu anebo komunikačního kanálu, úmyslným vysíláním rušivého signálu na stejném nebo blízkém frekvenčním pásmu, což může mít za následek narušení spojení mezi vozidly nebo vozidly a infrastrukturou.

Útok mužem uprostřed (Man-in-The-Midle-attack)

V kontextu komunikace V2X s autonomními vozidly může útočník odposlouchávat komunikaci mezi komunikujícími vozidly anebo mezi vozidly a infrastrukturou, aby sledoval či modifikoval přenos dat.

Útok odmítnutí služeb (Denial of Services attack)

Vysokorychlostní a vysokokapacitní 5G sítě, kterou využívá Cellular-V2X jsou náchylné k DoS útokům, což může zpomalit nebo zcela znemožnit poskytování služeb. Na útoky typu DoS trpí i protokol DSRC. Protiopatření řešící tento typ útoku spočívá v zavedení firewallu či IDS.

Útok podvržením (Spoofing attack)

Zneužívá slabiny v autentizačních a autorizačních procesech, umožňuje útočníkům simulovat legitimní mobilní zařízení nebo vytvářet falešné komunikační body. Tento útok lze použít i pro falšování GPS provozu. Útočník tak může podvrhnout polohu nebo identitu vozidla. Útočník rovněž může vytvářet falešné bezpečnostní zprávy, které informují ostatní vozidla o neexistujících nebo zmanipulovaných nebezpečných situacích.

Útok pohlcením (Black hole attack)

Jde o typ útoku, kde útočník záměrně přijímá veškerý síťový provoz a odmítá ho předat další části sítě. Útočník může falešně prezentovat sebe jako důvěryhodný uzel v komunikační síti autonomních vozidel. Když vozidlo nebo infrastruktura pošle komunikační zprávu nebo požadavek na informace, útočník tuto komunikaci pohltí anebo přesměruje na jiný zdroj.

Malware

Jde o škodlivý program nebo virus, který je vytvořený s úmyslem poškodit, narušit nebo získat neoprávněný přístup k systému. V kontextu autonomních vozidel může malware negativně ovlivnit integritu a bezpečnost vozidel či komunikaci s infrastrukturou.

3 Teleoperační řízení autonomních vozidel

Teleoperační řízení autonomních vozidel představuje operace, kde řízení vozidla může být prováděno vzdáleně za pomocí technologií dálkového ovládání. Tyto operace může provádět řidič či operátor z hlavního teleoperačního centra za pomocí síťového připojení a ovládacích prvků. Operátor má tak v tomto centru k dispozici aktuální informace o stavu vozidla a s využitím senzorů má možnost sledovat jeho okolí. Tato forma ovládání poskytuje lidským operátorům schopnost převzít kontrolu nad vozidlem v případě komplexních nebo nečekaných situací, kdy by autonomní systém mohl potřebovat pomoc. Problémy autonomní vozidlo může mít například v podobě selhání některého z klíčových senzorů z důvodu špatného počasí či poruchy systému, vozidlo může být rovněž poškozeno ať fyzicky při dopravní nehodě nebo kybernetickým útokem. Autonomní vozidlo samo nemůže učinit rozhodnutí proti pravidlům silničního provozu. Například vozidlo nemůže přejet souvislý oddělovací pruh, tato situace se může stát problémovou při objetí zablokované silnice z důvodu přejetého zvířete nebo jiné překážky. Někdy je potřeba systému v autonomním vozidle pomoci při navigaci v hustě obydlené oblasti anebo při průjezdu stavbou či zónou silničních prací. Budoucností tohoto dálkového ovládání je rovněž i možnost sdílení vozidel, kdy po použití vozidla jedním zákazníkem, může být stejné vozidlo operátorem dálkově odvezeno k zákazníkovi jinému, který jej zrovna bude potřebovat. Teleoperační řízení by mělo tedy poskytnout řešení na problémy jež jsou uvedeny výše a přinést jakousi vizi do sdílení dopravy [39, 40].

Teleoperační řízení však s sebou nese taky mnoho výzev a problémů, kterým budou muset operátoři řídicího centra čelit. Patři zde zvýšení kognitivní zátěže operátora, který převezme řízení nad vozidlem. Toto dálkové řízení se stává ještě těžším, jelikož operátor není schopen vnímat síly a zvuky působící na vozidlo. Mezi další problémy v této oblasti můžeme zařadit zpoždění přenosu informací po síti z vozidla do operačního centra. Celé řízení rovněž vyžaduje přenos živého obrazu z kamer v odpovídající kvalitě, což může být velice náročné na kvalitu služeb (QoS). Otázkou také zůstává, jaké data či informace přenášet. Přenos těchto dat musí být navíc zabezpečený a šifrovaný, aby byla zajištěna jeho důvěrnost, integrita a nepopíratelnost [39, 40].

Vývojem teleoperačního řízení se již zabývá několik firem mezi nejznámější v tomto odvětví patří společnost Vay[1], která působí na Německém území. Svou stopu v teleoperačním řízení má i česká firma Roboauto[2], která má sídlo v Brně.

[1]Společnost Vay: `https://vay.io`

[2]Společnost Roboauto: `https://roboauto.tech`

3.1 Komunikační model teleoperačního řízení

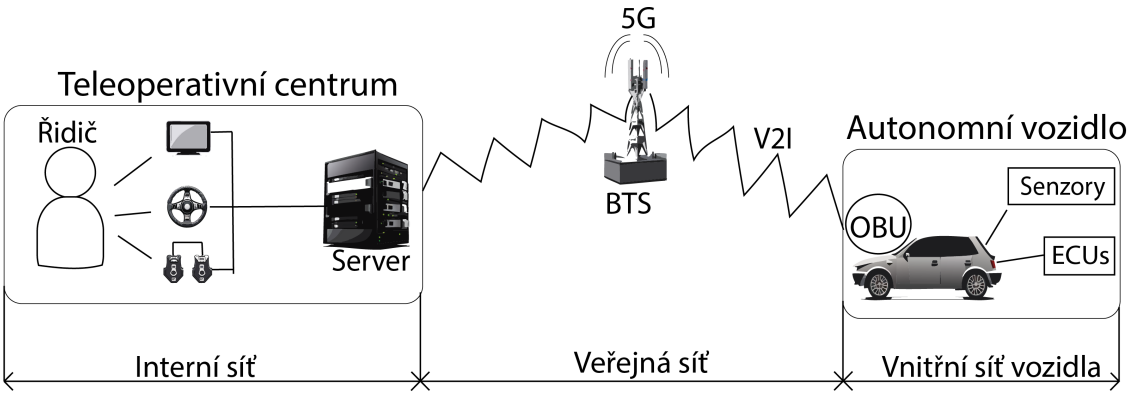
S nárůstem používání autonomních vozidel vzniká potřeba efektivního řízení a monitorování. Komunikační model pro dálkové řízení autonomních vozidel obvykle zahrnuje několik klíčových prvků, včetně dálkového teleoperačního centra, operátora, samotných autonomních vozidel a komunikační sítě, pomocí kterých spolu tyto prvky komunikují. V teleoperačním centru najdeme operátorské stanice, kde operátoři monitorují jednotlivá autonomní vozidla. Operátorské stanice jsou vybaveny různými hardwarovými prvky jako jsou monitory pro sledování vozidel v reálném čase. Tyto prvky umožňují zobrazit aktuální polohu autonomního vozidla, jeho telematická data a informace z různých senzorů. Do operátorské centra jsou rovněž přenášeny signály z kamer a dalších senzorů jako jsou radary či LiDARy, jež jsou umístěné nejčastěji na karoserii autonomního vozidla. Příklad takové teleoperační stanice ukazuje obrázek 3.1. Operátoři (řidiči) v teleoperačním centru tak sledují autonomní vozidlo a v případě potřeby lidské intervence nebo nečekané situace mají možnost nad vozidlem převzít kontrolu. Zároveň operátoři mohou provádět diagnostiku vozidla na dálku. Vozidlo mohou dálkově řídit pomocí aplikace a dalších zařízení k tomu určených. Mezi tyto zařízení patří například volant pro ovládání směru jízdy, pedály umožňující zrychlení a brzdění či monitor, na který je přenášen živý obraz z kamer umístěných na vozidle.

Obr. 3.1: Teleoperační stanice firmy Roboauto. (Zdroj: roboauto.tech, 2024)

3.1.1 Příklad komunikačního modelu

V této kapitole je představen příklad komunikačního modelu pro teleoperační řízení, kdy jednotlivé operátorské stanice komunikují přes interní síť se serverem teleope-

račního centra, který slouží jako centrální uzel pro správu a koordinaci autonomních vozidel. Ze serveru teleoperačního centra odchází data prostřednictvím veřejné sítě Internet, která slouží jako propojení mezi centrem a jednotlivými vozidly. Tyto data mohou být přenášena prostřednictvím různých komunikačních technologií v závislosti na dostupnosti a potřebách konkrétní situace. Komunikace mezi serverem a jednotlivými autonomními vozidly se může odehrávat skrze vysílací stanice (BTS). V případě mobilních sítí, přes bezdrátové WiFi spojení nebo pomocí satelitní komunikace, což umožňuje pokrytí širšího území a překonání geografických překážek. Data dorazí na speciální jednotku označovanou jako OBU, která je umístěna v každém autonomním vozidle. Tato OBU přebírá data z veřejné sítě a řídí komunikaci uvnitř vozidla. Její hlavní úlohou je zajistit přenos informací mezi teleoperačním centrem a jednotlivými systémy vozidla. OBU dále přeposílá požadavky do jednotlivých ECU jednotek a systémů, které mají na starosti samotné aspekty řízení vozidla. Tyto ECU provádějí konkrétní akce na základě instrukcí a dat přijatých z teleoperačního centra [41]. Schéma takového komunikačního modelu je znázorněno na obrázku 3.2.



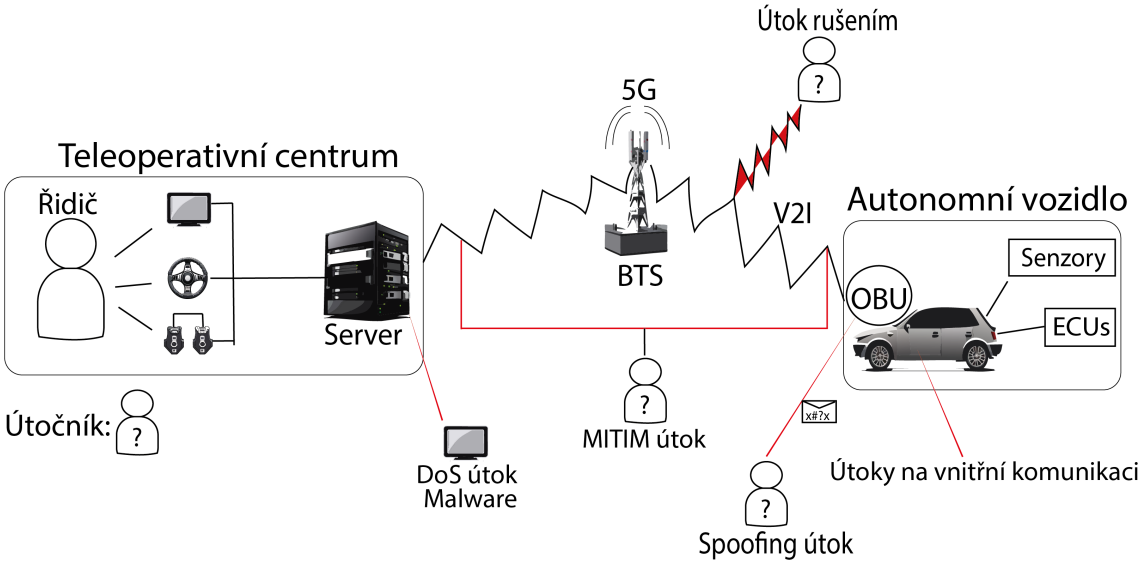
Obr. 3.2: Komunikační model pro teleoperační řízení

3.2 Analýza rizik teleoperačního řízení

V této kapitole jsou analyzovány možné rizika a kybernetické útoky, jež mohou realizovány na představeném komunikačním modelu. Následně je představena možnost ochrany před těmito hrozbami.

3.2.1 Rizika a hrozby komunikačního modelu

Před aplikací komunikačního modelu pro teleopererační řízení autonomních vozidel je nezbytné pečlivě prozkoumat potenciální rizika a možné hrozby kybernetických útoků, které mohou ovlivnit bezpečnost a spolehlivost tohoto řízení. Jedním z hlavních rizik je bezpečnost dat přenášených mezi serverem teleoperačního centra a autonomními vozidly prostřednictvím veřejné sítě Internet. Tento komunikační kanál může být náchylný na různé typy kybernetických útoků. Tyto útoky mohou mít široké spektrum dopadů, včetně neautorizovaného ovládání vozidel, poškození systémů nebo dokonce ohrožení bezpečnosti cestujících a ostatních účastníků silničního provozu. Mohou také umožnit útočníkům přístup k citlivým informacím. Dalším významným aspektem je komunikace mezi OBU a jednotlivými ECU uvnitř vozidla. Nedostatečná ochrana této komunikace by mohla vést k neoprávněnému přístupu a manipulaci s jednotlivými systémy vozidla, což by mohlo mít vážné následky na jeho bezpečnost a integritu. Na obrázku 3.3 jsou znázorněny potencionální kybernetické útoky, které vychází z útoků na komunikaci V2X, jež jsou popsány výše v kapitole 2.2.



Obr. 3.3: Přehled možných kybernetických útoků v komunikačním modelu

3.2.2 Ochrana komunikačního modelu

Pro ochranu před těmito hrozbami je nezbytné implementovat řadu bezpečnostních opatření. Patří sem například použití šifrování dat a bezpečných protokolů komunikace k zabezpečení přenosu dat mezi centrem a vozidly nebo implementace bezpečnostních opatření na úrovni komunikační infrastruktury, jako jsou firewally a detekce anomálií. Důležité je také pravidelné monitorování síťového provozu a rychlá reakce na zjištěné hrozby. To umožní identifikovat a řešit potenciální útoky včas a minimalizovat jejich dopad na bezpečnost a spolehlivost teleoperace autonomních vozidel.

Tato bakalářská práce se zaměřuje právě na analýzu a zvolení vhodného kryptografického protokolu, který umožní rychlý a bezpečný přenos dat mezi teleoperačním centrem a autonomním vozidlem samotným.

4 Analýza vhodných protokolů pro bezpečnou komunikaci

V následujících podkapitolách jsou popsány vhodné protokoly přenosu dat pro vzdálené řízení a zabezpečení komunikace mezi serverem a autonomním vozidlem, které pracují na transportní nebo aplikační vrstvě. Kritéria výběru jsou taková, že za pomocí protokolu budeme moct přenášet telemetrická data mezi serverem a autonomním vozidlem, navíc tento protokolu bude mít bezpečnostní mechanismy, aby byla zajištěna integrita, důvěrnost a autentizace přenosu.

4.1 Analýza protokolů podle modelu TCP/IP

4.1.1 Protokoly transportní vrstvy

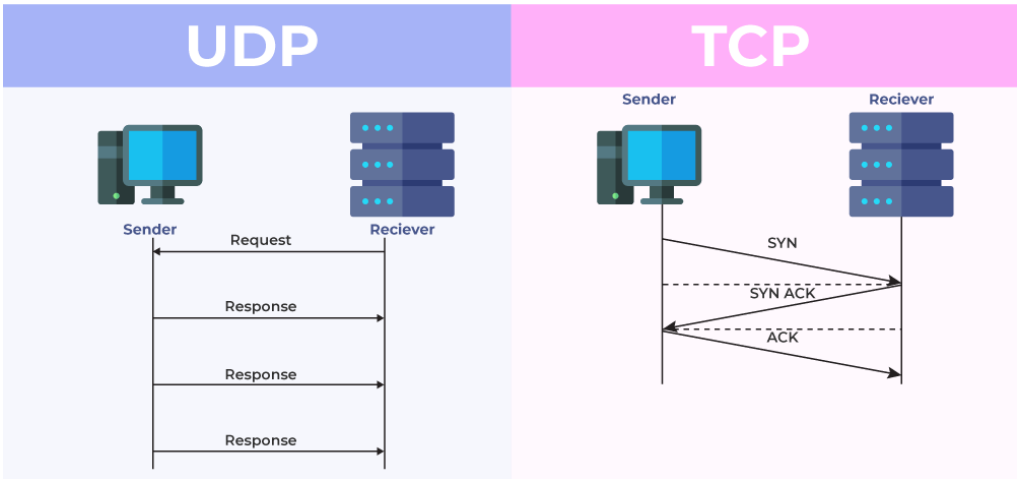
Transmission Control Protocol (TCP)

Jedná se o spojově orientovaný protokol transportní vrstvy. Je navržen pro spolehlivý a řízený přenos dat mezi zařízeními v síti. Jeho největší využití spočívá v navazování a udržování komunikace v síti Internet a spolehlivé doručování paketů. Využívá techniku číslování odesílaných a potvrzovaných bajtů, potvrzování příjmu dat a opakování přenosu v případě ztráty dat. Protokol také umí řídit tok dat a reagovat na zahlcení v síti. TCP je založen na mechanismu nazvaného „Three-Way Handshake“ (třífázové potvrzení) k navázání spojení mezi dvěma koncovými body (odesílatelem a příjemcem). Navázaná komunikace je plně duplexní. Protokol TCP využívá síťových portů, aby rozlišil jednotlivé služby. Jednotka přenosu u TCP se nazývá segment. Jeho maximální velikost je omezena na 65 535 B. Velikost záhlaví TCP je zpravidla 20 B. Mezi nejběžnější použití protokolu patří stahování webových stránek (HTTP), e-mailové přenosy (SMTP, IMAP) či vzdálené připojení (SSH). V kontextu komunikace mezi vzdáleným serverem a autonomním vozidlem může TCP poskytovat spolehlivý a řízený přenos dat. Například autonomní vozidlo může pravidelně posílat telemetrická data (polohu, stav senzorů) na server a přijímat instrukce a aktualizace softwaru zpět. TCP zajistí, že data budou doručena ve správném pořadí a že nedojde k jejich ztrátě [41, 42, 43].

User Datagram Protocol (UDP)

UDP je nespojově orientovaný standardizovaný protokol transportní vrstvy, jež poskytuje jednoduchý mechanismus přenosu dat mezi zařízeními v síti. Tento protokol se značně liší od TCP tím, že se jedná o nespolehlivý protokol, který neposkytuje

záruku doručení dat, kontrolu pořadí, opravu chyb či řízení přetížení. Jeho hlavním úkolem je poskytnout minimální režii a rychlost pro přenos dat. Stejně jako TCP používá síťové porty pro rozlišení služby. Jednotkou přenosu je zde datagram. UDP záhlaví má délku 8 B. UDP neklade žádné omezení na délku zprávy, ale každý paket je odeslán v rámci jednoho datagramu. UDP podporuje multicast a broadcast komunikaci, což umožňuje efektivní komunikaci s více příjemci zároveň. UDP je často využíván v aplikacích, které vyžadují rychlý přenos dat a jsou méně citlivé na ztrátu dat, jako jsou hlasové a video streamy, online hry, DNS dotazy a jiné real-time aplikace. V autonomních vozidlech by se UDP dal použít pro přenos telemetrických dat ze senzorů jako je LiDAR, radar či kamera, zejména pokud je vyžadován rychlý přenos dat a nízká režie. Dále pro data, která nevyžadují spolehlivost, jako jsou neprioritní telemetrické informace. Pro kritické účely v komunikaci mezi vzdáleným serverem a autonomním vozidlem, by však měl být zvážen doplněk o vlastní mechanismy spolehlivosti nebo by měl být použit TCP [41, 42, 44]. Handshake komunikace TCP a UDP protokolu znázorňuje obrázek 4.1.



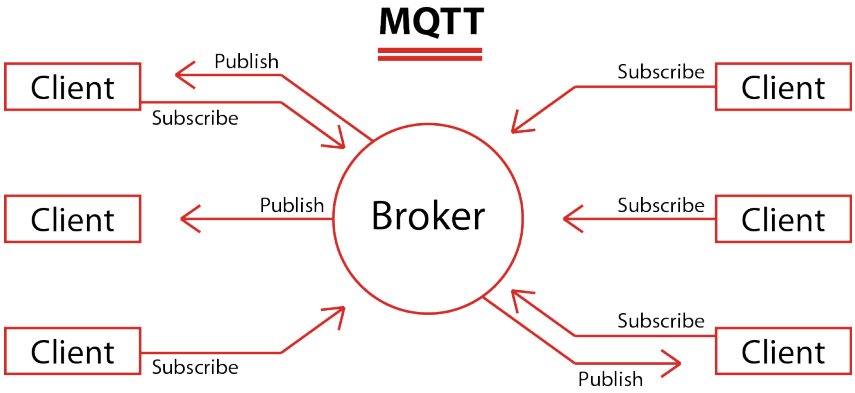
Obr. 4.1: Handshake TCP a UDP protokolu (Zdroj: www.geeksforgeeks.org, 2024).

4.1.2 Protokoly aplikační vrstvy

Message Queuing Telemetry Transport (MQTT)

Jedná se o jednoduchý, lehký protokol určený pro spolehlivý a efektivní přenos dat mezi různými zařízeními v síti a pro komunikaci v reálném čase. Tento protokol se využívá v prostředích s omezeným výpočetním výkonem nebo velmi omezenou šířkou pásma. MQTT využívá architekturu klient-server. Server je v roli tzv. brokera. Ten je centrální komponentou v síti MQTT. Jeho úkolem je přijímat zprávy

od klientů a distribuovat je dalším klientům podle jejich odběrových zájmů. Broker spravuje různá témata (topics), ke kterým se klienti mohou připojovat a odebírat tak zprávy. Klient může nabývat dvou rolí, a to publikovatele (publisher) a odběratele zpráv (subscriber). Publisher tedy publikuje zprávy na určité téma. Když klient publikuje zprávu, broker tuto zprávu distribuuje všem odběratelům připojeným k danému tématu. Zatímco subscriber odebírá zprávy z určitého tématu, které jsou mu zasílány brokerem. Architektura MQTT protokolu je znázorněna na obrázku 4.2. Protokolu můžeme definovat úroveň QoS. Maximální velikost zprávy je standardně definovaná na 256 MB. Režie zprávy je minimálně 2 B. MQTT využivá transportního protokolu TCP. Pro bezpečnou komunikaci lze MQTT zkombinovat s TLS protokolem čímž zajistíme auntetizaci, integritu a šifrování. Pro nezabezpečenou komunikaci běží standardně na portu 1883, pro komunikaci zabezpečenou je to port 8883. Protokol se hojně využívá v aplikacích IoT, kde se stal standardem v této oblasti, a to z důvodu nízké režie a spolehlivému obousměrnému zasílání zpráv. Prakticky se MQTT používá pro přenos telemetrických dat, monitoring či ovládání zařízení na dálku. V oblasti vzdáleného řízení autonomních vozidel může MQTT poskytovat spolehlivý mechanismus pro přenos řídících zpráv, stavů vozidla a dalších informací mezi serverem a vozidlem. Protokol má vhodné vlastnosti pro komunikaci mezi teleoperačním centrem a autonomním vozidlem [45, 46, 47].

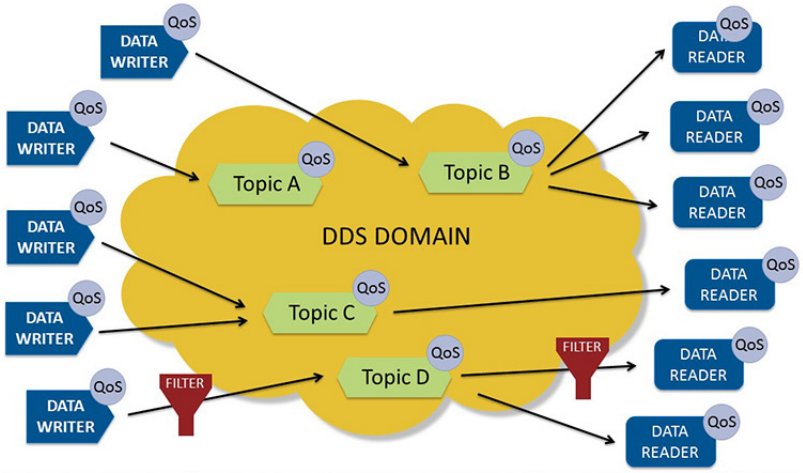


Obr. 4.2: Architektura MQTT protokolu (Zdroj: www.pickdata.net, 2024).

Data Distribution Service (DDS)

Standardní protokol pro distribuovanou komunikaci v reálném čase. Poskytuje spolehlivý, vysokorychlostní a distribuovaný mechanismus pro výměnu dat v reálném čase mezi heterogenními systémy. DDS funguje na základě principu publish-subscribe, kdy publsiher publikuje zprávy na dané téma a subscriber je odebírá. Princip je podobný jako u protokolu MQTT, avšak na rozdíl od něj neobsahuje brokera. DDS

využívá centralizovanou nebo distribuovanou službu distribuce dat, která slouží jako prostředník mezi publishery a subscribery. Tato služba spravuje přenos dat a doručuje je příjemcům na základě jejich odběrových preferencí. DDS také obsahuje dva typy rozhraní dataWriter to je rozhraní umožňující aplikacím publikovat data a dataReader to umožňuje aplikacím data odebírat. Architekturu DDS protokolu znázorňuje obrázek 4.3. DDS je nezávislý na použitém protokolu nižší vrstvy. Velikost záhlaví je různá v závislosti na implementaci. Záhlaví obsahuje metadata pro směrování a správu zpráv. V protokolu můžeme nastavit úroveň QoS. DDS podporuje šifrování a zabezpečení dat pomocí různých mechanismů, jako je TLS nebo proprietární metody zabezpečení. Tento protokol se hojně využívá v aplikacích, které vyžadují spolehlivou a škálovatelnou distribuci dat v reálném čase, jako jsou systémy řízení letových misí, průmyslové automatizační systémy, systémy monitorování a řízení dopravy. Ve smyslu autonomních vozidel DDS umožňuje rychlý a spolehlivý přenos dat mezi různými komponentami vozidla, včetně senzorů, řídících jednotek, navigačních systémů a systémů řízení. Jelikož DDS umožňuje rychlý a spolehlivý přenos dat mezi těmito komponentami, mohl by protokol být použitý pro komunikaci mezi teleoperačním centrem a autonomním vozidlem [48, 49].

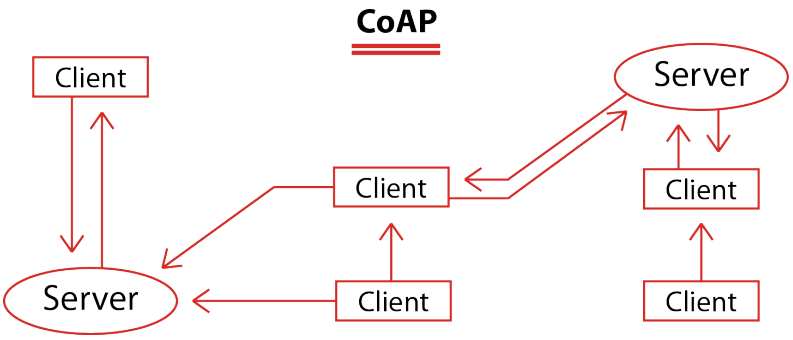


Obr. 4.3: Architektura DDS protokolu (Zdroj: www.dds-foundation.org, 2024).

Constrained Application Protocol (CoAP)

Protokol CoAP je navržený pro komunikaci v omezených prostředích, kde jsou často omezené prostředky jako paměť, energetická spotřeba a šířka pásma. Jedná se specializovaný webový protokol, který je jednoduchý a snadno implementovatelný na omezené zařízení. Umožňuje tak rychlou komunikaci v IoT prostředí. Architektura protokolu je založena na principu klient-server. Klienti zasílají požadavky na server

pomocí CoAP zpráv. Server přijímá požadavky od klientů a poskytuje odpovědi na tyto požadavky. Využívá se zde standardních HTTP metod a to GET, POST, PUT a DELETE. Místo témat využívá URI adresy. Jako protokol transportní vrstvy používá UDP pro snížení režie, navíc poskytuje některé mechanismy pro spolehlivou komunikaci, jako jsou potvrzování doručení a opakování zpráv. Standardně běží na portu 5683. Zabezpečení komunikace lze zajistit pomocí protokolu DTLS. Pro zabezpečenou komunikaci se standardně používá port 5684. Maximální délka zprávy v CoAP je doporučena na 1 152 B v základní konfiguraci. Protokol umožňuje volbu úrovně QoS. Režie zprávy je minimálně 4 B. CoAP je často používán v IoT prostředí pro komunikaci mezi zařízeními a servery, ale také najde uplatnění v dalších aplikacích, jako jsou senzorové sítě, inteligentní města, a řízení budov. Díky svým vlastnostem jednoduchosti, efektivity a zabezpečení je CoAP stále populárnější volbou pro komunikaci v omezených sítových prostředích a rovněž vhodný pro mezi teleoperačním centrem a autonomním vozidlem [45, 50] Architekturu protokolu CoAP popisuje obrázek 4.4.



Obr. 4.4: Architektura CoAP protokolu (Zdroj: www.pickdata.net, 2024).

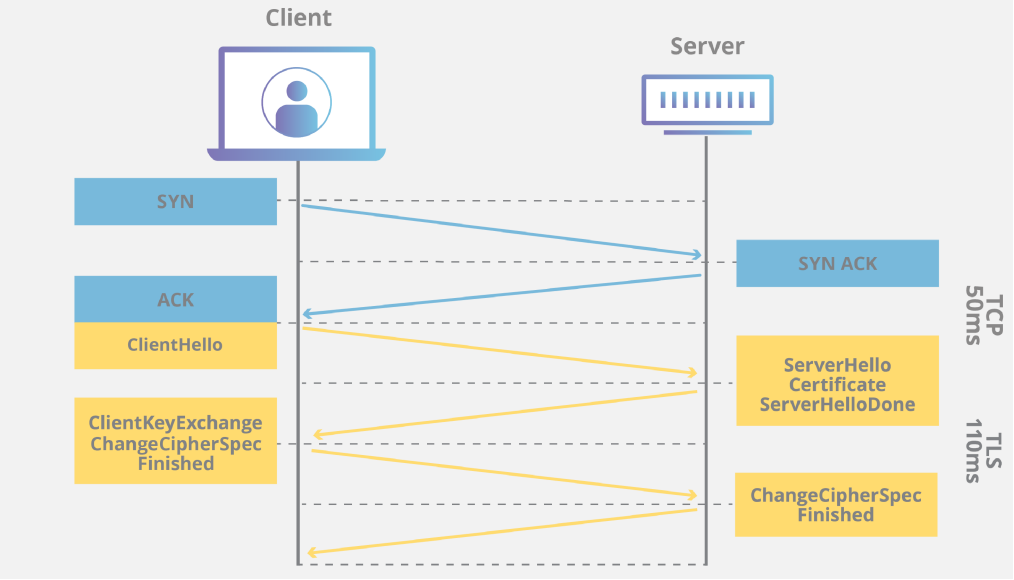
4.2 Analýza kryptografických protokolů

Existuje mnoho kryptografických protokolů, které se používají pro zabezpečení komunikace na internetu či v konkrétních aplikacích. Jejich počet neustále roste s vývojem nových technologií. Co se týče využití pro zabezpečení vzdáleného řízení autonomních vozidel by nebyly všechny vhodné. Při výběru vhodného kryptografického protokolu pro vzdálené řízení autonomních vozidel bylo klíčové zajistit, aby komunikace mezi vozidlem a ovládacím systémem byla chráněna před potenciálními bezpečnostními hrozbami. Před navržením vhodného konkrétního kryptografického protokolu bylo nezbytné identifikovat požadavky, které jsou specifické pro vzdálené řízení vozidel. Mezi klíčové požadavky pro výběr vhodných protokolů patřilo:

- Šifrování dat: zajištění utajení přenášených dat tak, aby byla data chráněna před neoprávněným přístupem;
- Ověření identity: možnost ověřit, že komunikující strany jsou legitimní a že data nejsou odesílána nebo přijímána falešnými entity;
- Ochrana před útoky typu MITM: prevence proti útokům, kde útočník zachytí a mění přenášená data mezi vozidlem a ovládacím systémem;
- Nízká latence: zajištění, že kryptografické operace nezpůsobují výrazné zpoždění, což je klíčové pro řízení vozidel.

Transport Layer Security (TLS)

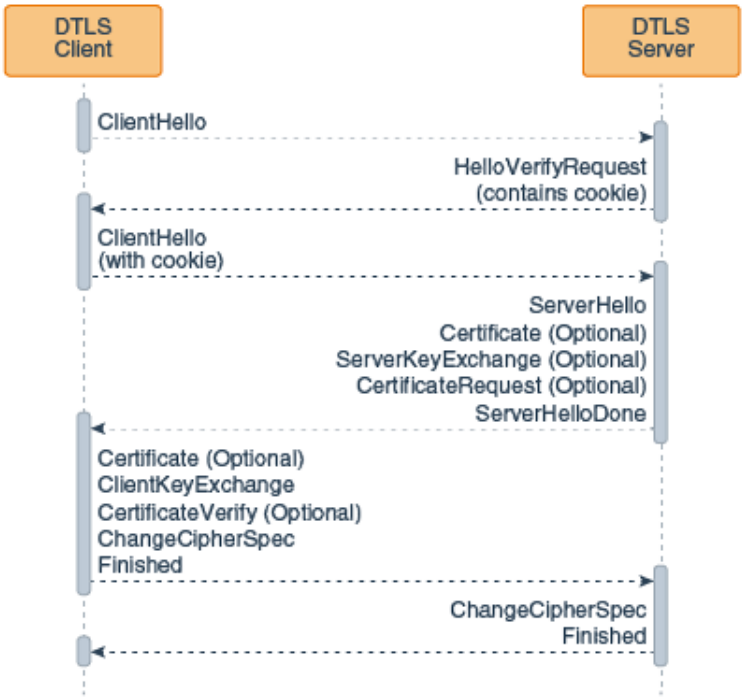
Je kryptografický protokol navržený pro zabezpečení komunikace přes počítačovou síť. Jeho hlavním účelem je poskytnout šifrování, autentizaci a integritu dat přenášených mezi dvěma komunikujícími stranami. TLS je používán pro zabezpečení komunikace mezi webovými servery a prohlížeči, e-mailovými servery či klienty. TLS může být využit pro zabezpečení komunikace mezi různými komponentami autonomních vozidel. Prakticky jej lze také implementovat pro zabezpečení vzdálené komunikace mezi serverem a autonomním vozidlem. Vozidlo naváže spojení s operačním centrem a proběhne tzv. handshake jenž je zobrazen na obrázku 4.5. Během tohoto procesu dojde k výměně informací o šifrovacích klíčích, certifikátech a dalších parametrech. Po úspěšné výměně klíčů se naváže šifrované spojení. TLS také umožňuje autentizaci komunikujících stran pomocí digitálních certifikátů. To znamená, že klient i server mohou ověřit, že komunikují s legitimními partnery [51, 52].



Obr. 4.5: Handshake protokolu TLS (Zdroj: www.cloudfare.com, 2024).

Datagram Transport Layer Security (DTLS)

Je kryptografický protokol, který poskytuje zabezpečení komunikace v prostředích, kde je vyžadovaná nízká latence. Byl vyvinut, aby nahradil TLS v situacích, kdy spolehlivost a zabezpečení jsou prioritou, a kde není vhodné použít TCP jako transportní protokol. To zahrnuje aplikace jako jsou VoIP, online hry, IoT a další. DTLS je často používán pro zabezpečení komunikace v prostředích, kde se kladou vysoké požadavky na rychlost a spolehlivost. Protokol poskytuje veškeré bezpečnostní funkce jako je šifrování dat, ověřování identity pomocí certifikátů a zachování integrity dat. Takto zabezpečený je každý přenášený datagram. Prostřednictvím mechanismů pro opětovné odeslání a detekci ztráty paketů umožňuje DTLS bezpečnou komunikaci i v prostředích s vysokou ztrátovostí dat. Díky schopnosti pracovat s nespolehlivými spojeními může DTLS být vhodným zabezpečením pro komunikaci v autonomních vozidlech, kde je důležité minimalizovat latenci a zajistit bezpečný přenos dat [53]. Na obrázku 4.6 je vyobrazen handshake protokolu DTLS.

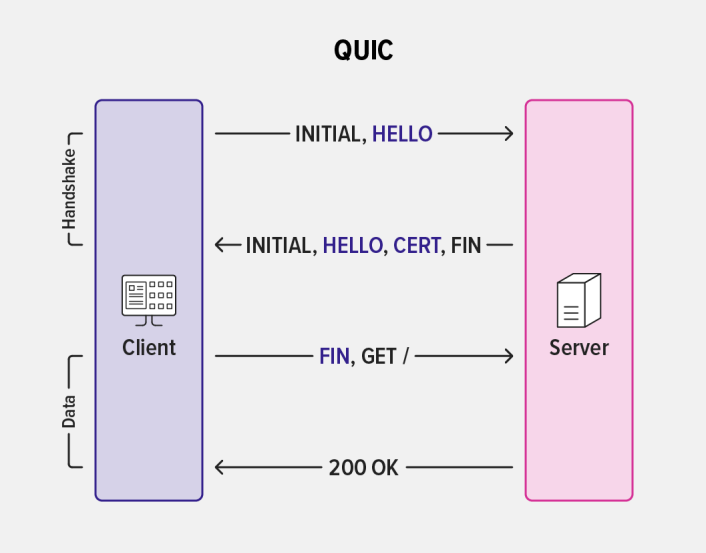


Obr. 4.6: Handshake protokolu DTLS (Zdroj: docs.oracle.com, 2024).

Quick UDP Internet Connections (QUIC)

QUIC je moderní protokol pro zabezpečenou komunikaci přes internet, který byl vyvinut společností Google. Jedná se o protokol pracující na úrovni transportní vrstvy. Byl navržen pro rychlou a bezpečnou komunikaci. Protokol je postavený nad

UDP a přebírá výhody protokolu TLS. QUIC je používán pro různé aplikace, jako jsou webové prohlížeče, mobilní aplikace nebo streamování videa. Je podporován v moderních webových prohlížečích. QUIC umožňuje multiplexování více spojení přes jedno síťové spojení. Protokol umožňuje standardní šifrování přenášených dat a možnost adaptace na změny sítě jako jsou ztráty paketů nebo změny v propustnosti. Protokol QUIC stejně jako DTLS poskytuje nižší režii než zabezpečení pomocí TLS. Díky rychlosti a spolehlivosti může QUIC být vhodným protokolem pro komunikaci mezi senzory, řídícími jednotkami, navigačními systémy a centrálním řídicím systémem, stejně tak pro zabezpečení komunikace mezi vzdáleným serverem a autonomním vozidlem [54, 55]. Handshake protokolu QUIC ukazuje obrázek 4.7.



Obr. 4.7: Handshake protokolu QUIC (Zdroj: www.nginx.com, 2024).

4.3 Zhodnocení a volba vhodných protokolů

V předchozích částech této práce byly zanalyzovány vhodné protokoly, které umožňují přenos v real-time systémech a poskytují nízkou míru latence a zpoždění při přenosu. Dále byly prozkoumány vhodné kryptografické protokoly, pomocí nichž by se dala celá komunikace zabezpečit. Na základě těchto poznatků byly pro komunikaci mezi teleoperačním centrem a autonomním vozidlem vybrány aplikační protokoly MQTT a CoAP, které byly doplněny o kryptografické protokoly TLS a DTLS jež poskytnou zabezpečení celé komunikace. Implementace vybraných protokolů je uvedena v praktické části, kde byly testovány a bylo provedeno jejich zhodnocení.

5 Knihovny a MQTT brokeři

Následující podkapitoly popisují knihovny, jež byly využity, a za pomoci kterých lze implementovat protokol MQTT a CoAP. Dále jaké knihovny nebo balíčky poskytují zabezpečení pomocí protokolů TLS či DTLS. Na závěr kapitoly byl popsán využitý broker Eclipse Mosquitto a dva alternativní open-source MQTT brokeři.

5.1 Knihovna implementující protokol MQTT

Eclipse Paho

Eclipse Paho[1] je open-source knihovna, která je součástí projektu Eclipse IoT. Knihovna umožňuje vytváření klientů MQTT či MQTT-SN v různých programovacích jazycích jako jsou Java, C, C++, Python a další. Tito klienti potom interagují s MQTT brokerem. Podporuje širokou škálu platforem, včetně Linuxu, Windows, macOS, Android. Je distribuován pod open-source licencí Eclipse Public License (EPL) 2.0. a je k dispozici ke stažení z oficiálních repozitářů. Mezi její hlavní výhody patří podpora různých jazyků a platforem, snadná integrace do projektů a možnost volby QoS. Knihovna se hojně využívá pro aplikace v IoT či pro komunikaci typu M2M (Machine-to-Machine), kde umožňuje monitorování a řízení pomocí MQTT protokolu. Zde je odkázáno na GitHub repozitář knihovny[2].

5.2 Balíčky implementující zabezpečení TLS

Implementaci protokolu TLS pro zabezpečení MQTT komunikace mají v jazyce Java na starosti balíčky javax.net.ssl a java.security. Níže byly tyto balíčky popsány.

Javax.net.ssl

Balíček javax.net.ssl obsahuje třídy a rozhraní pro implementaci zabezpečené komunikace pomocí SSL/TLS protokolů v Javě. Tento balíček umožňuje vytvářet SSL/TLS spojení, spravovat certifikáty, šifrování a autentizaci. Je vhodný pro zabezpečení komunikace mezi klientskou a serverovou části programu. Jeho využití v praktické části spočívalo při vytvoření zabezpečených socketů pro klientskou část a server nebo pro vytvoření TLS kontextu a definování konfigurace TLS spojení.

[1] Oficiální stránky Eclipse Paho: `https://eclipse.dev/paho`

[2] GitHub repozitář knihovny Eclipse Paho `https://github.com/eclipse/paho.mqtt.java`

Java.security

Balíček java.security je základní balíček pro správu bezpečnosti v Javě. Obsahuje třídy a rozhraní pro správu kryptografických algoritmů, certifikátů, klíčů, bezpečnostních politik a dalších bezpečnostních mechanismů. Tento balíček poskytuje infrastrukturu pro zabezpečení Javy a je klíčovým komponentem pro implementaci bezpečnostních funkcí. V praktické části byl použit pro uložení a zároveň správu klíčů a certifikátů.

5.3 Knihovna implementující protokol CoAP a DTLS

5.3.1 Eclipse Californium

Eclipse Californium[3] se skládá z několika klíčových knihoven a modulů, které společně tvoří komplexní nástrojovou sadu pro implementaci CoAP protokolu. Je nepsána v programovacím jazyce Java, má multiplatformní vlastnosti a může být použita na většině operačních systémů, včetně Windows, Linux, MacOS. Mezi výhody patří podpora různých režimů provozu a podpora zabezpečené komunikace pomocí DTLS. Je licencována pod open-source licencí EPL 2.0. Odkaz na GitHub repozitář knihovny [4].

Californium Core

Californium Core[5] je základní jádro knihovny, které obsahuje implementaci CoAP protokolu a základní nástroje pro vytváření CoAP klientů a serverů. Uživatelé mohou snadno vytvářet CoAP zprávy, odesílat je, zpracovávat je a komunikovat s CoAP servery a klienty. Knihovna podporuje základní metody jako GET, POST, PUT a DELETE protokolu CoAP.

Element-Connector

Element-Connector[6] jedná se o modul, který je navržen pro usnadnění integrace a rozšíření základního jádra Californium. Tento modul poskytuje abstrakci a nástroje, které umožňují snadné propojení a komunikaci mezi různými částmi aplikace a různými komponentami v Californium. Obsahuje také některé pomocné třídy, jako

[3]Oficiální stránky Eclipse Californium: `https://eclipse.dev/californium`

[4]GitHub repozitář knihovny Eclipse Californium: `https://github.com/eclipse-californium/californium`

[5]GitHub repozitář knihovny Californium Core: `https://github.com/eclipse-californium/californium/tree/main/californium-core#californium-cf---coap-core`

[6]GitHub repozitář modulu Element-connector: `https://github.com/eclipse-californium/californium/tree/main/element-connector#element-connector`

je SslContextUtil pro načítání certifikátů. Tato třída podporuje různé druhy certifikačních formátu jako PKCS#12, JKS či PEM.

Scandium

Scandium [7] je knihovna pro DTLS komunikaci, která je používána jako základ pro Californium DTLS. Tento modul poskytuje podporu pro protokol DTLS, který je používána v CoAP pro šifrování a autentizaci komunikace. Poskytuje DTLS ve verzi 1.2. Implementuje rozhraní Element-connector. Umožňuje autentizaci na základě metody předsdílených klíčů (PSK) nebo certifikátu X509. Podporuje mnoho moderních kryptografických algoritmů včetně eliptických křivek.

5.4 MQTT Brokeři

Eclipse Mosquitto

Eclipse Mosquitto[8] je open-source broker napsaná v jazyce C/C++, který implementuje protokol MQTT. Jedná se o lehký a snadno použitelný broker, který podporuje MQTT verze 3.1, 3.1.1 a 5.0. Pomocí něj můžeme vytvořit decentralizovanou a spolehlivou komunikačních sítí mezi zařízeními IoT. Podporuje zabezpečení zpráv pomocí SSL/TLS. Je k dispozici pro operační systémy Windows, Linux, MacOS či Rasberry PI. Mosquitto dále poskytuje dva klienty možné spustit v příkazové řádce, a to mosquitto_pub a mosquitto_sub. Mosquitto_pub ten simuluje roli klienta publishera. Mosquitto_sub naopak simuluje roli příjemce zpráv subscribera. Zprávy jsou posílány v reálném čase. Mosquitto je distribuován pod licencí EPL 2.0. Své výhody v jednoduchosti nasazení, výborné komunitě a dokumentaci vyvažuje omezení ve škálovatelnosti a pokročilých funkcích jako clustering či nasazení do cloudového prostředí. Využívá se pro senzory, mobilní zařízení či mikrokontroléry [56].

EMQX

EMQX[9] jedná se o výkonný, spolehlivý a škálovatelný MQTT broker. Byl napsán v programovacím jazyce Erlang/OTP, aby byl použitelný pro širokou škálu aplikací, od malých domácích projektů až po velké průmyslové a komerční systémy. Podporuje MQTT 5.0 či MQTT 3.1.1. Umožňuje zabezpečení pomocí protokolu SSL/TLS či protokolu QUIC. Broker je multiplatformní může běžet na operačních systémech

[7] GitHub repozitář knihovny Scandium: `https://github.com/eclipse-californium/californium/tree/main/scandium-core#scandium-sc---security-for-californium`

[8] Oficiální stránky brokera Eclipse Mosquitto: `https://mosquitto.org`

[9] Oficiální stránky brokera EMQX: `https://www.emqx.io`

Windows, Linux či MacOS. EMQX je distribuován pod open-source licencí Apache License 2.0, což znamená, že je zdarma k použití, modifikaci a distribuci bez větších omezení. Mezi jeho výhody patří podpora clusteringu, integrace do cloudu a využití protokolu QUIC. Díky své vysoké výkonnosti je však výrazně náročnější na procesor a paměť. Využíván pro kritické aplikace v IoT [56].

NanoMQ

NanoMQ[10] je jeden z nejnovějších MQTT brokerů. Broker je napsán v jazyce C. Plně podporuje MQTT verze 3.1.1 a 5.0 a zabezpečení protokolem SSL/TLS či QUIC. Mezi jeho vlastnosti patří lehkost a rychlost. Mezi výhody patří nízká paměťová náročnost. Broker může být nastaven, aby fungoval jako brána pro další protokoly například DDS. NanoMQ lze implementovat v cloudovém prostředí. Broker je distribuován pod open-source licencí MIT License. NanoMQ je multiplatformní a podporuje následující operační systémy a platformy: Linux, Windows, MacOS, Raspberry Pi a další. Broker však strádá v nemožnosti použití clusteringu. Praktické využití spočívá v automobilovém průmyslu a robotice [56].

[10]Oficiální stránky brokera NanoMQ: `https://nanomq.io`

6 Implementace navržených protokolů

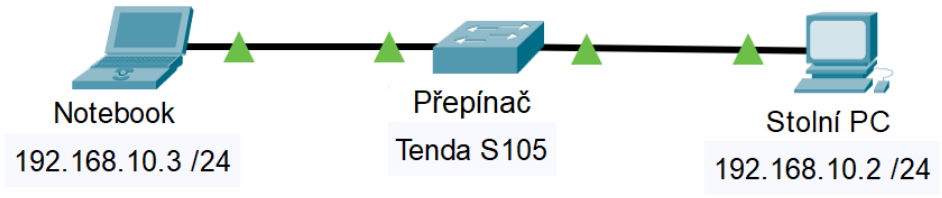
Pro praktickou část v rámci této bakalářské práce byla navržena a implementována aplikace simulující vzdálenou teleoperaci autonomního vozidla. Kapitola je rozdělena na pět podkapitol. V první podkapitole najdeme základní popis aplikace, architekturu zapojení včetně potřebného nastavení pro správné fungování a samotné spuštění aplikace. Následující podkapitoly popisují jednotlivé implementace protokolů simulující komunikaci mezi teloperačním centrem a autonomním vozidlem. V podkapitole 6.2 je představena nezabezpečená implementace za pomocí protokolu MQTT. Komunikace pomocí protokolu MQTT, avšak se zabezpečením pomocí protokolu TLS je realizována v podkapitole 6.3. Podkapitola 6.4 implementuje komunikaci prostřednictvím protokolu CoAP bez přidaného zabezpečení. Poslední podkapitola 6.5 popisuje komunikaci pomocí protokol CoAP doplňuje ji o zabezpečení protokolem DTLS.

6.1 Základní popis aplikace

Aplikace byla napsána v jazyce Java s využitím vývojového prostředí Eclipse IDE na operačním systému Windows 10. Výsledná aplikace se skládá ze dvou samostatných programů, které jsou implementovány na dvou různých zařízeních, a to na stolním počítači a notebooku. Program pro stolní počítač simuluje teleoperační centrum, zatímco program na notebooku představuje autonomní vozidlo. Jedná se o konzolovou aplikaci bez grafického rozhraní. Architektura je typu klient-server. Teleoperační centrum systému slouží k odesílání binárních souborů, které představují příkazy a data. Jeho úkolem je předávat tyto příkazy klientovi čili autonomnímu vozidlu prostřednictvím zvoleného protokolu. Klient tyto soubory následně uloží. Na této programové logice jsou následně implementovány a testovány protokoly pro přenos dat a jejich možnost zabezpečení.

6.1.1 Architektura

Teleoperační centrum představuje stolní počítač (IP adresa: 192.168.10.2/24) jenž je připojen pomocí síťového kabelu typu UTP s konektorem RJ-45 na port 1 přepínače. Z portu 2 přepínače vede další síťový kabel, a ten je připojen k notebooku (IP adresa: 192.168.10.3/24). Ten reprezentuje autonomní vozidlo. Přepínač byl použit pěti-portový s přenosovou rychlostí 10/100 Mb za sekundu a celkovou propustností 1 Gb za sekundu. Použitý přepínač byl model S105 od společnosti Tenda. Celá komunikace tak tvoří lokální síť. Architektura zapojení je zobrazena na obrázku 6.1.



Obr. 6.1: Architektura zapojení

6.1.2 Nastavení sítě a firewall

Aby spojení fungovalo bylo nutné jak na stolním počítači, tak notebooku nakonfigurovat dané IP adresy včetně masky podsítě. Po nakonfigurování IP adres se stav sítě změnil na neznámá síť. Dále bylo potřeba přes místní počítač – zásady; nastavení zabezpečení; zásady správce seznamu sítí; ve vlastnostech neznámé sítě změnit typ z veřejné na privátní síť. Pro správné fungování přenosových protokolů, bylo nutné nastavit na zařízeních příchozí a odchozí pravidla firewallu a povolit vhodné porty.

6.1.3 Import jar souborů

Před samotnou implementací jednotlivých verzí klienta a serveru komunikujících zvoleným protokolem bylo zapotřebí naimportovat potřebné knihovny. Ty jsou popsány v kapitole 5 pro protokol MQTT šlo o knihovnu Eclipse Paho ve verzi MQTT 5, pro protokol CoAP knihovnu Eclipse Californium ve verzi 3.11.0 a její další součásti. Knihovny byly staženy jako soubory jar a přidány do složky external_jar ve složce projektů. Z této složky je potom šlo snadno přidat do vývojového prostředí Eclipse IDE skrze vlastnosti projektu; knihovny; přidat externí knihovnu. Posledním nastavením bylo přidání knihoven do závislostí konfigurace běhu.

6.1.4 Spuštění aplikace

Po spuštění programu, ať už na stolním počítači či notebooku, je uživatel přivítán v menu s několika možnostmi. První možnost zobrazuje základní informace o projektu, včetně jména autora a popisu projektu. Tato sekce umožňuje uživateli získat základní přehled o projektu.

Druhá a třetí možnost umožňují uživateli spustit nezabezpečenou nebo zabezpečenou komunikaci pomocí MQTT. Pokud uživatel vybere možnost druhou dojde k navázání nezabezpečené komunikace klienta publishera nebo subsribera s MQTT brokerem. Volba závisí na tom, jestli je spuštěn program pro teleoperační centrum

nebo autonomní vozidlo. Třetí volba na rozdíl od druhé pracuje se zabezpečeným spojením za použití protokolu TLS.

Čtvrtá a pátá volba menu umožňují uživateli spustit komunikaci pomocí CoAP. Opět zde máme možnost nezabezpečeného a zabezpečeného spojení, kde zabezpečené spojení využívá protokol DTLS. Kód programu spuštěný na stolním počítači představuje CoAP server. Naopak kód programu spuštěný na notebooku simuluje roli klienta, který se připojuje k serveru. Výběrové menu klienta je zobrazeno na obrázku 6.2. Výběrové menu serveru je téměř totožné.

```
Hello I am a autonomous car
 -- Welcome in menu --
 -- Select the desired action --
*****************************************
1 .. Show basic project documentation
2 .. Start an insecure communication between server and client via MQTT protocol
3 .. Start an secure communication between server and client via MQTT with TLS protocol
4 .. Start an insecure communication between server and client via CoAP protocol
5 .. Start an secure communication between server and client via CoAP + DTLS protocol
*****************************************************************************************
```

Obr. 6.2: Výběrové menu klienta

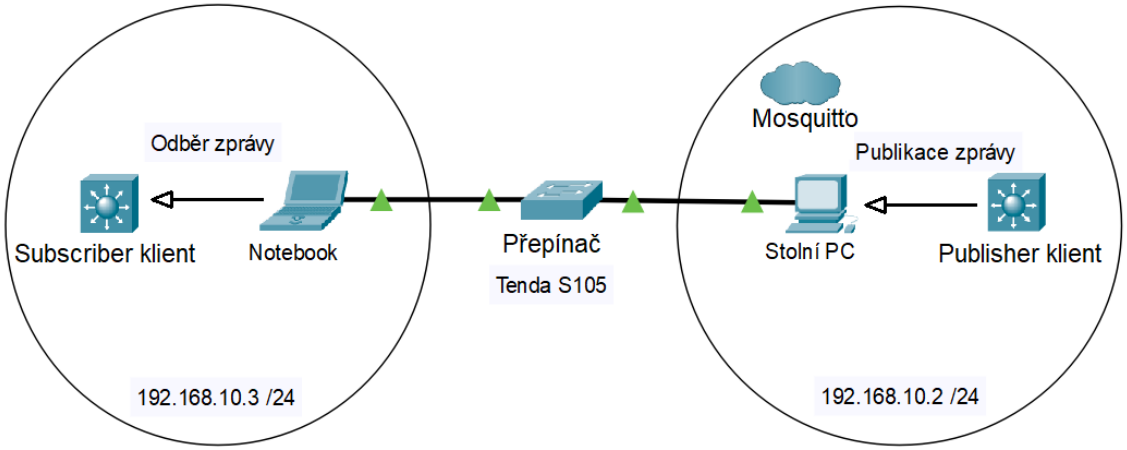
6.2 Implementace protokolu MQTT

Podrobnější postup implementace protokolu v jazyce Java je možné nalézt zde [57]. Pro implementaci byl využit broker Mosquitto a knihovna Eclipse Paho. Protokol MQTT byl realizován ve verzi 5.

6.2.1 Broker

Broker Eclipse Mosquitto je nainstalován na stolním počítači. Pro autentizaci uživatelů broker používá soubor passwd, který obsahuje uživatelská jména a hesla. Z důvodu bezpečnosti bylo v konfiguračním souboru zakázáno anonymní připojení. Soubor passwd byl vytvořen pomocí příkazového řádku ve Windows a programu mosquitto_passwd.exe, který Mosquitto poskytuje. Jediný uživatel, který je v souboru passwd vytvořený a má možnost se připojit k brokerovi je uživatel s uživatelským jménem: **test** a heslem: **test123**. Broker běží na localhostu stolního počítače a poskytuje nezabezpečené spojení na portu 1883. Jelikož bylo Mosquitto ve výchozím nastavení nastavené pouze k přijímání připojení klientů z lokálního zařízení, bylo nutné nastavit parametry připojení. Pro přijímání připojení klientů i z jiného, než lokálního zařízení bylo zapotřebí v konfiguračním soubor **mosquitto.conf** připsat řádek listener s číslem portu, na které broker naslouchá a IP adresu, na které

je broker spuštěný. K brokeru Mosquitto se připojují klienti, kteří jsou popsáni níže. Manuálovou stránku konfiguračního souboru mosquitto.conf najdeme na odkazu [58]. Architektura reálného zapojení klientů a Mosquitto brokera je znázorněna na obrázku 6.3.



Obr. 6.3: Schéma komunikace protokolem MQTT

6.2.2 Klient publisher

Tato část implementace popisuje MQTT klienta v roli publishera, který pracuje s nezabezpečeným spojením. Kód je navržen tak, aby se klient připojil k brokeru Mosquitto přes protokol TCP na IP adresu 192.168.10.2 a port 1883. Dále bylo nastaveno téma „VehicleCommunication“ a úroveň QoS. Nastavení základních parametrů pro připojení k brokerovi znázorňuje výpis 6.1

Výpis 6.1: Nastavení parametrů MQTT spojení klient publisher.

```
public class MqttSubscriberNotSecure {                              1
    public void ConnectToPublisherOverMqtt() {                      2
        String topic = "VehicleCommunication";                      3
        String broker = "tcp://192.168.10.2:1883";                  4
        String id = "PublisherClient";                              5
        int qos = 2; }                                              6
}                                                                   7
```

Qos je nastaveno na úroveň dva, ta zaručuje, že každá zpráva je doručena přesně jednou. Aby se dosáhlo této úrovně spolehlivosti využívá protokol MQTT mechanismus potvrzování doručení a opětovného zasílání zpráv. V případě chyby či ztráty komunikace klient zašle brokerovi zprávu znovu, tato úroveň zaručuje doručení zpráv

bez duplicit. Oproti tomu úroveň jedna zajišťuje doručení zprávy alespoň jednou, ale povoluje vícenásobné doručení stejné zprávy. Poslední možností je úroveň nula, která sice má nižší nároky na síť a nižší latenci než předchozí úrovně, ale v kontextu komunikace mezi teleoperačním centrem a autonomním vozidlem není vhodná. Tato úroveň totiž nezaručuje, že zpráva bude doručena v případě výpadku sítě nebo jiné chyby. Více o QoS v MQTT protokolu najdeme na odkazu [59].

Když uživatel spustí program, je nejprve vyzván k zadání svého uživatelského jména (test) a hesla (test123). Tyto údaje slouží k ověření proti konfiguraci brokeru, kde jsou uloženy. Po zadání těchto informací se klient inicializuje svým ID a využívá paměťového úložiště pro ukládání dat. Následně dochází k připojení k MQTT brokerovi Mosquitto a možnosti publikování zpráv. Ve výpise 6.2 lze vidět základní kód pro publikování zpráv brokerovi pomocí protokolu MQTT.

Výpis 6.2: Publikování zprávy klientem publisherem.

```
public class MqttSubscriberNotSecure {
    public void ConnectToPublisherOverMqtt() {

     MemoryPersistence mp = new MemoryPersistence();
     MqttClient client = new  MqttClient(broker, id, mp);
     MqttConnectionOptions cOpt = new MqttConnectionOptions();
     MqttMessage message = new MqttMessage();

     client.setCleanStart(true);
     client.connect(cOpt);

     message.setPayload(data);
     message.setRetained(true);
     message.setQos(qos);
     client.publish(topic, message); }
}
```

Při úspěšném připojení k brokeru Mosquitto se zaznamená časový záznam o začátku komunikace do logu. Klient následně spustí nekonečnou smyčku, kde uživatel má možnost vybrat a odeslat jeden z předdefinovaných binárních souborů. Tyto soubory jsou v rozsahu od 1 B až po 10 MB. Každý odeslaný soubor je zaznamenán do logu spolu s časem odeslání.

Po úspěšném odeslání uživatel může pokračovat v odesílání dalších souborů nebo ukončit spojení s brokerem zadáním „exit“. Pokud uživatel ukončí spojení, klient se následně odpojí od brokeru a zaznamená čas ukončení spojení do logu.

6.2.3 Klient subscriber

MQTT subscriber se nezabezpečeným spojením připojuje na brokera Mosquitto. IP adresa a port pro navázání komunikace s brokerem jsou stejné jako u klienta publishera. Při spuštění programu je rovněž uživatel vyzván k zadání svého uživatelského jména (test) a hesla (test123), které se porovnají se souborem passwd. Po úspěšné autentizaci dojde k inicializaci klienta, navázání spojení s brokerem a zaznamenaní času zahájení komunikace do logu. Po úspěšném připojení začne klient odebírat zprávy z tématu „VehicleCommunication“. Výpis 6.3 poskytuje kód pro odběr a uložení zpráv od brokera.

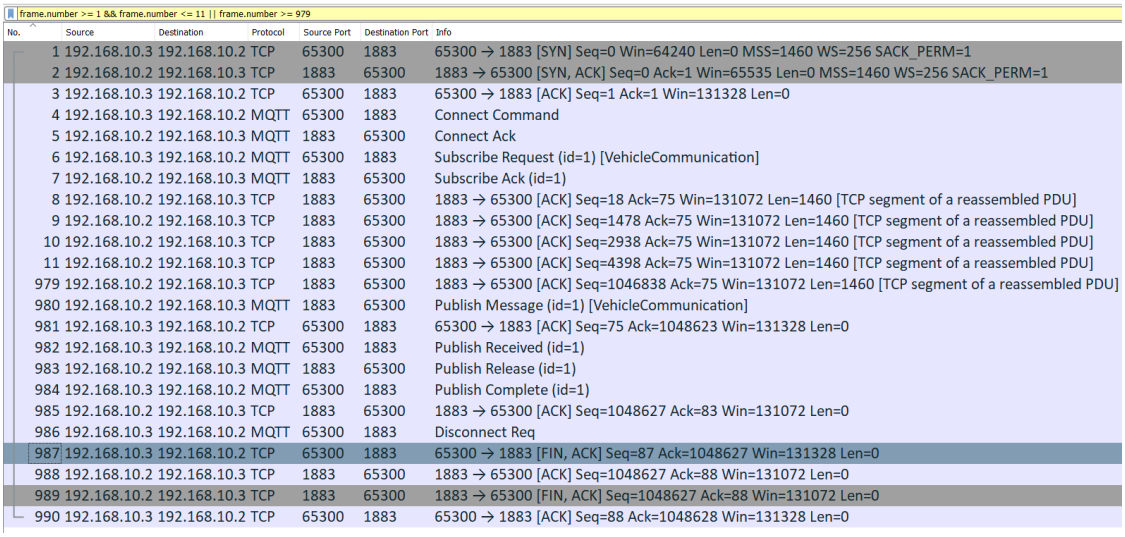
Výpis 6.3: Přijetí a uložení zprávy klient subscriber.

```
public class MqttSubscriberNotSecure {                                  1
    public void ConnectToPublisherOverMqtt() {                          2
                                                                        3
     client.setCallback(new MqttCallback() {                            4
        @Override                                                       5
        public void messageArrived(String topic,                        6
          MqttMessage message) throws Exception {                       7
        FileSaver.saveFile(message.getPayload(), topic);                8
        }                                                               9
                                                                        10
        client.connect(cOpt);                                           11
        client subscribe(topic,2) }                                     12
}                                                                       13
```

Každá přijatá zpráva je uložena. Čas přijetí a uložení souboru je zaznamenán do logu. Uživatel má možnost ukončit spojení s brokerem prostřednictvím zadání „exit“. Pokud tak učiní, klient se odpojí a zaznamená čas ukončení spojení do souboru. Obrázek 6.4 ukazuje průběh nezabezpečené komunikace mezi klienty pomocí protokolu TCP a MQTT v programu Wireshark.

6.3 Implementace protokolu MQTT se zabezpečením TLS

Implementace zabezpečené verze pomocí protokolu MQTT s protokolem TLS vychází z předešlého kódu pro verzi nezabezpečenou. Pro tuto implementaci byl rovněž využit broker Mosquitto a knihovna Eclipse Paho. Pro zabezpečení byla použita verze 1.3 protokolu TLS. Při zabezpečení pomocí protokolu TLS se pro připojení již nevyužívá IP adresy, ale názvu hostitele (hostname). Pro mapování hostname



frame.number >= 1 && frame.number <= 11 || frame.number >= 979

No.	Source	Destination	Protocol	Source Port	Destination Port	Info
1	192.168.10.3	192.168.10.2	TCP	65300	1883	65300 → 1883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	192.168.10.3	192.168.10.2	TCP	65300	1883	65300 → 1883 [ACK] Seq=1 Ack=1 Win=131328 Len=0
4	192.168.10.3	192.168.10.2	MQTT	65300	1883	Connect Command
5	192.168.10.2	192.168.10.3	MQTT	1883	65300	Connect Ack
6	192.168.10.3	192.168.10.2	MQTT	65300	1883	Subscribe Request (id=1) [VehicleCommunication]
7	192.168.10.2	192.168.10.3	MQTT	1883	65300	Subscribe Ack (id=1)
8	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [ACK] Seq=18 Ack=75 Win=131072 Len=1460 [TCP segment of a reassembled PDU]
9	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [ACK] Seq=1478 Ack=75 Win=131072 Len=1460 [TCP segment of a reassembled PDU]
10	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [ACK] Seq=2938 Ack=75 Win=131072 Len=1460 [TCP segment of a reassembled PDU]
11	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [ACK] Seq=4398 Ack=75 Win=131072 Len=1460 [TCP segment of a reassembled PDU]
979	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [ACK] Seq=1046838 Ack=75 Win=131072 Len=1460 [TCP segment of a reassembled PDU]
980	192.168.10.2	192.168.10.3	MQTT	1883	65300	Publish Message (id=1) [VehicleCommunication]
981	192.168.10.3	192.168.10.2	TCP	65300	1883	65300 → 1883 [ACK] Seq=75 Ack=1048623 Win=131328 Len=0
982	192.168.10.3	192.168.10.2	MQTT	65300	1883	Publish Received (id=1)
983	192.168.10.2	192.168.10.3	MQTT	1883	65300	Publish Release (id=1)
984	192.168.10.3	192.168.10.2	MQTT	65300	1883	Publish Complete (id=1)
985	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [ACK] Seq=1048627 Ack=83 Win=131072 Len=0
986	192.168.10.3	192.168.10.2	MQTT	65300	1883	Disconnect Req
987	192.168.10.3	192.168.10.2	TCP	65300	1883	65300 → 1883 [FIN, ACK] Seq=87 Ack=1048627 Win=131328 Len=0
988	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [ACK] Seq=1048627 Ack=88 Win=131072 Len=0
989	192.168.10.2	192.168.10.3	TCP	1883	65300	1883 → 65300 [FIN, ACK] Seq=1048627 Ack=88 Win=131072 Len=0
990	192.168.10.3	192.168.10.2	TCP	65300	1883	65300 → 1883 [ACK] Seq=88 Ack=1048628 Win=131328 Len=0

Obr. 6.4: Průběh komunikace MQTT bez zabezpečení

respektive doménového jména na IP adresu bylo potřeba upravit soubor hosts na obou zařízeních, který je umístěn v operačním systému Windows 10 ve složce „C: Windows/System32/drivers/etc/hosts“. Pro IP adresu 192.168.10.2 je mapováno doménové jméno AVcommunication.com.

6.3.1 Parametry TLS zabezpečení

Protokol TLS pro autentizaci využívá certifikátu a páru soukromého a veřejného klíče. Pro generování certifikátů a klíčů byla využita služba OpenSSL a virtuální počítač s operačním systémem Kali, kde je již služba předinstalována. Výsledné soubory byly potom přesunuty do složky s programy klientů a složky brokera. Postup generování klíčů a certifikátu pomocí služby OpenSSL nalezneme zde [60].

Proces autentizace

Klient se pokusí navázat spojení s brokerem pomocí protokolu TLS. Během navazování spojení se prověří certifikáty procesem nazvaným TLS handshake. Broker ověří certifikát klientů a klienti zase ověří certifikát brokera. Následně se provede ověření certifikátů pomocí certifikační autority (CA), přičemž certifikáty klientů a brokera musí důvěřovat. Po úspěšném dokončení handshake procesu jsou vytvořeny relační klíče, které jsou odvozeny z veřejných klíčů, které byly vyměněny během handshake procesu. Po vytvoření relačních klíčů jsou všechny zprávy, které jsou vyměňovány mezi klientem a brokerem, šifrovány pomocí těchto klíčů a dešifrovány soukromými klíči na opačné straně.

Kryptografické algoritmy certifikátů

Uživatel má možnost si vybrat ze dvou možností kryptografických algoritmů, pomocí kterých byly vytvořeny certifikáty a klíče certifikačních autorit, brokera a klientů. Prvním typem kryptografického algoritmu je „Rivest, Shamir, Adleman“ (RSA) s délkou klíče 2048 b. Pro druhý typ byl zvolen kryptografický algoritmus využívající kombinaci eliptických křivek a algoritmu Digital Signature Algorithm (DSA), a to Elliptic Curve Digital Signature Algorithm (ECDSA). Parametry klíčů vytvořených algoritmem ECDSA byly eliptické křivky prime256v1. Žádosti o certifikát byly navíc doplněny o hashovací algoritmus sha256. Více informací o použitých kryptografických algoritmech najdeme na odkazu [61, 62]. Tabulka 6.1 poskytuje přehled vytvořených certifikátu a klíčů.

Název souboru	**Typ souboru**	**Parametry**	**Platnost**
localhostCA	.crt	RSA 2048 b	3650 dní
CA_ECDSA	.crt	ECDSA prime256v1, sha256	3650 dní
broker	.crt	RSA 2048 b	364 dní
broker_ECDSA	.crt	ECDSA prime256v1, sha256	364 dní
broker	.key	RSA 2048 b	-
broker_ECDSA	.key	ECDSA prime256v1	-
client_pub_RSA	.p12	RSA 2048 b	364 dní
client_pub_ECDSA	.p12	ECDSA prime256v1, sha256	364 dní
client_sub_RSA	.p12	RSA 2048 b	364 dní
client_sub_ECDSA	.p12	ECDSA prime256v1, sha256	364 dní

Tab. 6.1: Přehled využitých certifikátů a klíčů v protokolu TLS

Uživatel si tak může vybrat ze dvou možností CA. Soubor localhostCA.crt je certifikát CA vytvořený algoritmem RSA. Soubor CA_ECDSA představuje certifikát CA vygenerovaný pomocí algoritmu ECDSA.

Uživatel na základě volby CA může použít pro autentizaci brokera Mosquitto ze dvou serverových certifikátů. S využitím algoritmu RSA je to soubor broker.crt pro ECDSA soubor broker_ECDSA.crt. Těmto certifikátům odpovídají páry klíčů, opět ve formátu RSA je to broker.key a ve formátu ECDSA broker_ECDSA.key.

Na základě volby CA a serverového certifikátu brokera jsou k dispozici klientské certifikáty ve formátu PKCS#12. Pro klienta publishera jsou to soubory client_pub_RSA.p12 využívající algoritmus RSA a client_pub_ECDSA.p12 s algoritmem ECDSA. Klient subscriber zase využívá certifikáty uložené v souborech nazývajících se client_sub.p12.

Metoda ustanovení klíčů

Při komunikaci mezi klienty MQTT a brokerem Mosquitto je klíčovou součástí zajištění výměny klíčů. Pro zajištění ustanovení klíčů bylo použito dvou metod. Pro certifikáty vytvořené algoritmem RSA se používá pro ustanovení klíče také RSA. V druhé metodě pro ustanovení klíčů se používá algoritmus Elliptic Curve Diffie-Hellman (ECDH) s eliptickou křivkou prime256v1. Druhá metoda je použita s kombinací certifikátů vytvořených algoritmem ECDSA. Broker Mosquitto tyto metody použije automaticky na základě vytvořených certifikátů.

Generování certifikátů

Průběh vytvoření certifikátu byl následující. Nejprve byl vygenerován soukromý klíč CA. Veřejný klíč byl extrahován automaticky ze soukromého klíče. Následně byl vytvořen certifikát CA ve standardu x509, který byl podepsán sám sebou. Odkaz na soubor CA je potřeba definovat v konfiguračním souboru mosquitto.conf a programech klientů.

Pro autentizaci brokera Mosquitto bylo zapotřebí vygenerovat serverový certifikát a pár soukromého klíče. Soukromý klíč byl použit pro vytvoření nové žádosti o certifikát. Nově vytvořená žádost byla sama podepsána již vytvořenou CA, tím byl vytvořen certifikát brokera. Aby certifikát při TLS spojení fungoval správně musí být hodnota CN (common name) certifikátu stejná jako název hostitele (hostname) pro připojení. Cesty k CA, certifikátu a soukromému klíči brokera jsou definovaný v konfiguračním souboru Mosquitto.

Certifikáty pro klienty byly vytvořené podobným způsobem jako certifikát brokera. Pro práci s certifikáty v programech klientů v jazyce Java je však bylo nutné exportovat z formátu PEM do formátu PKCS#12. Při exportu bylo zapotřebí vytvořit heslo pro certifikáty klientů. Heslo pro oba klienty je **test123**. Cesty k CA a certifikátům s jejich hesly jsou definovány v programech klientů. Výpis 6.4 uvádí příklad vytvoření soukromého klíče a certifikátu klienta algoritmem RSA, jejich podepsání CA a následný export do formátu PKCS#12 pomocí služby OpenSSL.

Výpis 6.4: Vytvoření soukromého klíče a certifikátu klienta pomocí OpenSSL.

```
openssl genrsa -out client.key 2048                                    1
openssl req -new -out client.csr -key client.key                       2
                                                                       3
openssl x509 -req -in client.csr -CA localhostCA.crt -CAkey            4
localhostCA.key -CAcreateserial -out client.crt -days 364              5
                                                                       6
openssl pkcs12 -export -in client.crt -inkey client.key                7
-out client.p12                                                        8
```

Šifrování a integrita

Pro zajištění šifrování a integrity dat je použitá šifra TLS_AES_256_GCM_SHA384, která je výchozí konfigurací protokolu TLS pro šifrování v brokerovi Mosquitto. Použitou šifrou pro zajištění šifrování je bloková symetrická šifra Advanced Encryption Standard (AES) s délkou klíče 256 bitů a režimem GCM, který šifruje data po blocích a zajišťuje šifrování a autentizaci zároveň. Zajištění integrity dat má na starosti kryptografická hashovací funkce Secure Hash Algorithm (SHA) s délkou klíče 384 bitů.

6.3.2 Broker

Broker běží na localhostu stolního počítače a poskytuje zabezpečené šifrované spojení za pomocí protokolu TLS na portu 8883. Pro autentizaci brokera a klientů byla použita metoda výměny certifikátů a klíčů, ty jsou popsány výše. Výpis 6.5 zobrazuje kompletní nastavení konfiguračního souboru pro zabezpečenou komunikaci. Jsou zde nastaveny cesty k certifikátům brokera a certifikační autoritě. Dále je nastaveno poskytnutí certifikátu klienty, verze protokolu TLS, typ šifry a použití CN z certifikátů pro řízení přístupu.

Výpis 6.5: Konfigurační soubor Mosquitto.

```
#MQTT secure connection with TLS + certificates

    listener 8883 AVcommunication.com

    #RSA certificates
    cafile certifikaty\localhostCA.crt
    certfile certifikaty\broker.crt
    keyfile certifikaty\broker.key

    #ECDSA certificates
    #certfile certifikaty\broker_ECDSA.crt
    #cafile certifikaty\CA_ECDSA.crt
    #keyfile certifikaty\broker_ECDSA.key

    tls_version tlsv1.3
    ciphers_tls1.3 TLS_AES_256_GCM_SHA384
    require_certificate true
    use_identity_as_username true
```

6.3.3 Klient publisher

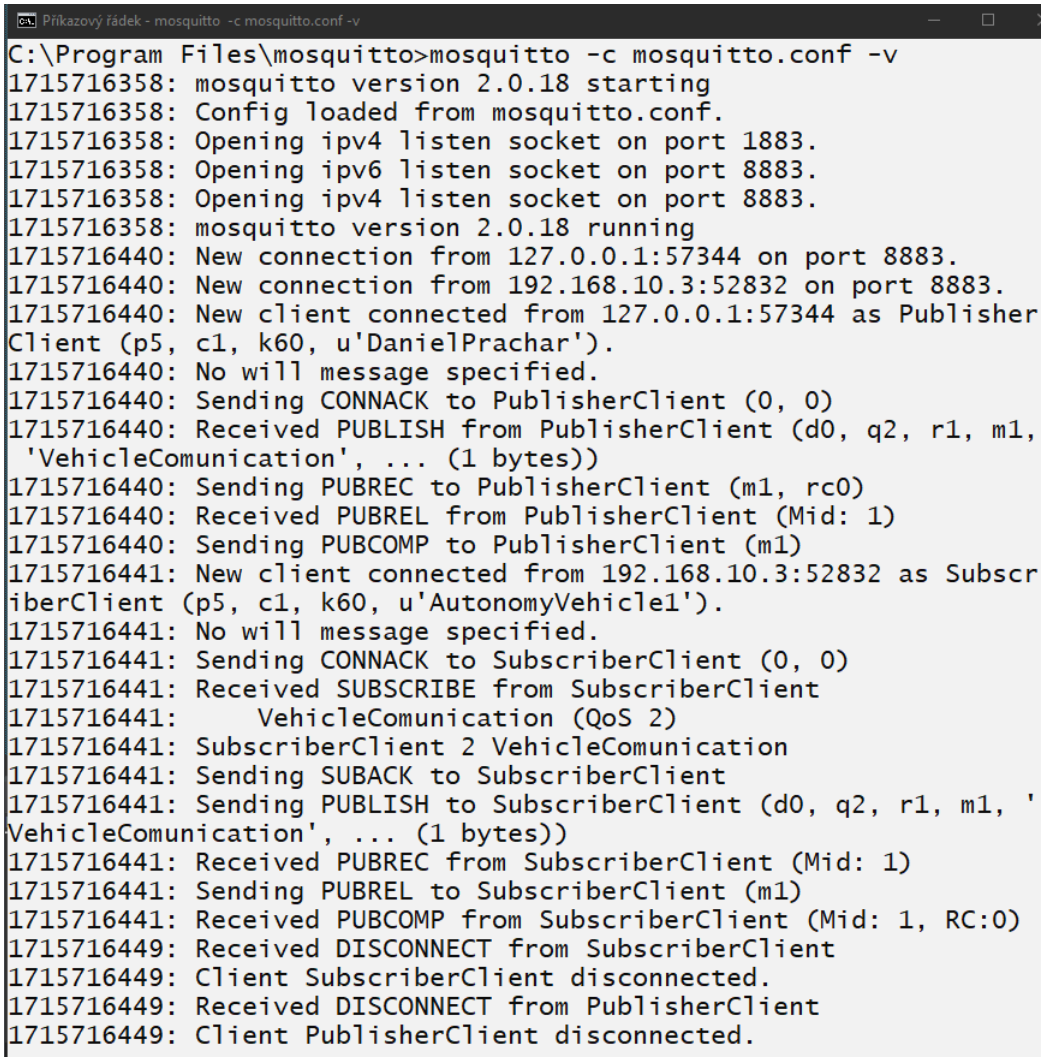
Nastavení klienta publishera v zabezpečené verzi pomocí protokolu TLS je podobné jako u nezabezpečené verze, najdeme zde, ale několik rozdílů. Klient se nyní připojuje k brokerovi Mosquitto pomocí protokolu TLS nikoliv MQTT, a to pomocí doménového jména (AVcommunication.com) a portu 8883. Téma odběru zpráv a úroveň QoS zůstávají neměnné. Výpis 6.6 ukazuje část kódu pro nastavení zabezpečení klienta publishera pomocí protokolu TLS s využitím certifikátu vygenerovaných kryptografickým algoritmem ECDSA.

Výpis 6.6: Kód pro zabezpečení pomocí protokolu TLS.

```
public class MqttPublisherTls {
    public void ConnectToSubscriberOverMqttWithTls(){

    CertificateFactory caFac = CertificateFactory
    .getInstance("X.509");

    FileInputStream caF = new FileInputStream("CA_ECDSA.crt");
    X509Certificate caCert = (X509Certificate)
    caFac.generateCertificate(ca);

    KeyStore ks = KeyStore
    .getInstance(KeyStore.getDefaultType());
    ks.load(null);
    ks.setCertificateEntry("localhost", caCert);

    String password = "test123";
    KeyStore kmKs = KeyStore.getInstance("PKCS12");
    kmKs.load(new FileInputStream("client_pub_ECDSA.p12")
    ,password.toCharArray());

    SSLContext sslContext = SSLContext.getInstance("TLSv1.3");
    sslContext.init(kmf.getKeyManagers()
    ,tmf.getTrustManagers(), null);

    sslContext.getDefaultSSLParameters()
    .setCipherSuites(new String[]{"TLS_AES_256_GCM_SHA384"});

    connOpts.setSocketFactory(sslContext.getSocketFactory());
    }
}
```

Prvkem zabezpečení je využití keystore pro uchování klíčů a certifikátů, které zajišťují autentizaci. Při inicializaci spojení s brokerem je klient vybaven certifikátem od CA, který je použit k ověření serveru. Tento certifikát je načten z CA a následně uložen do keystore. Pro autentizaci klienta je použit certifikát a soukromý klíč obojí uložené v PKCS#12 formátu. Tyto klíče a certifikáty jsou načteny a následně nakonfigurovány pomocí instancí KeyManagerFactory a TrustManagerFactory. Po načtení a konfiguraci všech potřebných certifikátů je vytvořen SSL kontext s verzí TLS 1.3.

Po úspěšném navázání komunikace je klient schopen posílat nyní již šifrované binární soubory. Funkce logování či ukončení spojení po zadání „exit“ zůstaly zachovány. Níže lze vidět průběh komunikace v brokerovi Mosquitto obrázek 6.5.



Obr. 6.5: Ověření komunikace v Mosquitto

6.3.4 Klient subscriber

Pro klienta v roli subscriber je nastavení připojení, téma odběru zpráv, úroveň QoS a postup autentizace naprosto totožný jako u klienta v roli publishera. Nejdříve se ověří server pomocí certifikátu CA, poté dojde k načtení a uložení certifikátu do keystore. Pro autentizaci klienta je využit certifikát a soukromý klíč.

Po navázání šifrovaného spojení s brokerem přijímá klient šifrované zprávy, které dešifruje a uloží. Veškerá komunikace se opět zaznamenává do logu a je možné ji ukončit jako u nezabezpečené verze řešení. Obrázek 6.6 ukazuje průběh komunikace mezi klienty v programu Wireshark pomocí protokolu TLS.

frame.number >= 1 && frame.number <= 22 || frame.number >= 882

No.	Source	Destination	Protocol	Source Port	Destination Port	Info
1	192.168.10.3	192.168.10.2	TCP	52938	8883	52938 → 8883 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	192.168.10.2	192.168.10.3	TCP	8883	52938	8883 → 52938 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	192.168.10.3	192.168.10.2	TCP	52938	8883	52938 → 8883 [ACK] Seq=1 Ack=1 Win=131328 Len=0
4	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Client Hello
5	192.168.10.2	192.168.10.3	TLSv1.3	8883	52938	Server Hello, Change Cipher Spec, Application Data, Application Data
6	192.168.10.2	192.168.10.3	TLSv1.3	8883	52938	Application Data, Application Data, Application Data
7	192.168.10.3	192.168.10.2	TCP	52938	8883	52938 → 8883 [ACK] Seq=481 Ack=2471 Win=131328 Len=0
8	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Change Cipher Spec
9	192.168.10.2	192.168.10.3	TCP	8883	52938	8883 → 52938 [ACK] Seq=2471 Ack=487 Win=130816 Len=0
10	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Application Data
11	192.168.10.2	192.168.10.3	TCP	8883	52938	8883 → 52938 [ACK] Seq=2471 Ack=1414 Win=129792 Len=0
12	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Application Data, Application Data
13	192.168.10.2	192.168.10.3	TLSv1.3	8883	52938	Application Data
14	192.168.10.3	192.168.10.2	TCP	52938	8883	52938 → 8883 [ACK] Seq=1806 Ack=3622 Win=130048 Len=0
15	192.168.10.2	192.168.10.3	TLSv1.3	8883	52938	Application Data
16	192.168.10.3	192.168.10.2	TCP	52938	8883	52938 → 8883 [ACK] Seq=1806 Ack=4773 Win=131328 Len=0
17	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Application Data
18	192.168.10.2	192.168.10.3	TLSv1.3	8883	52938	Application Data
19	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Application Data
20	192.168.10.2	192.168.10.3	TLSv1.3	8883	52938	Application Data
21	192.168.10.2	192.168.10.3	TCP	8883	52938	8883 → 52938 [ACK] Seq=4834 Ack=1940 Win=131072 Len=1460 [TCP segment of a reassembled PDU]
22	192.168.10.2	192.168.10.3	TCP	8883	52938	8883 → 52938 [ACK] Seq=6294 Ack=1940 Win=131072 Len=1460 [TCP segment of a reassembled PDU]
882	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Application Data
883	192.168.10.2	192.168.10.3	TCP	8883	52938	8883 → 52938 [ACK] Seq=1054894 Ack=2024 Win=131072 Len=0
884	192.168.10.3	192.168.10.2	TLSv1.3	52938	8883	Application Data
885	192.168.10.2	192.168.10.3	TLSv1.3	8883	52938	Application Data
886	192.168.10.2	192.168.10.3	TCP	8883	52938	8883 → 52938 [FIN, ACK] Seq=1054918 Ack=2066 Win=131072 Len=0

Obr. 6.6: Průběh komunikace MQTT se zabezpečením TLS

6.4 Implementace protokolu CoAP

Jako druhý protokol, který byl implementován a testován v kontextu komunikace mezi vzdáleným teleoperačním serverem a autonomním vozidlem byl protokol CoAP. Stolní počítač (IP adresa 192.168.10.2) představuje CoAP server, zatímco notebook (IP adresa 192.168.10.3) CoAP klienta. Implementace využívá knihovnu Eclipse Californium. Článek popisující protokol CoAP a jeho implementaci lze nalézt na odkazu [63].

6.4.1 Serverová část

Pro simulaci teleoperačního centra byl vytvořen kód jednoduchého CoAP serveru, ten naslouchá na portu 5683. Před spuštěním serveru se program dotáže uživatele na zadání uživatelské jména (test) a hesla (test123), které se následně porovná s přihlašovacími údaji v souboru passwd.txt. Ten se nachází ve složce s programem. Po spuštění serveru se registrují konfigurační parametry protokolu CoAP a UDP. Následně se vytvoří instance CoapServer a dojde k zaznamenání času spuštění serveru do logu. Spuštění serveru na daném portu představuje výpis 6.7.

Výpis 6.7: Vytvoření CoAP serveru na portu 5683.

```
public class ServerCoapNotSecure {
    public void ConnectionToClientOverCoap(){

        CoapConfig.register();
        UdpConfig.register();

        CoapServer server = new CoapServer(5683);
        server.start(); }
}
```

Server vytvoří zdroj komunikace s názvem „VehicleCommunication“, ten slouží pro odeslání binárních souborů klientovi. Zdroj obsahuje metodu handleGET, která reaguje na GET požadavek od klienta. Když je klient připojen, server odpoví odesláním binárního souboru. Mohou být odeslány binární soubory o velikosti od 1 B do 1 kB. Odesílané binární soubory jsou doplněny o kód odpovědi CONTENT, který je používán při úspěšném zpracování GET požadavku klienta na získání informací o zdroji, což znamená, že odpověď obsahuje požadovaný obsah. Výpis 6.8 popisuje kód, který umožňuje ze serveru odeslat binární soubor jakmile se připojí klient.

Výpis 6.8: Odeslání binárního souboru CoAP klientovi.

```
public class ServerCoapNotSecure {
  public void ConnectionToClientOverCoap(){
    server.add(new CoapResource("VehicleCommunication") {
      @Override
        public void handleGET(CoapExchange exchangefile) {

          exchangefile.respond(ResponseCode.CONTENT, data);
          }
        }); }
}
```

Server také zaznamenává čas odeslání souboru do logu. Server umožňuje ukončit spojení s klientem zadáním „exit“ do konzole, což způsobí ukončení běhu serveru a zápis do logu o ukončení komunikace. V případě neukončení spojení zadáním „exit“ zůstane port serveru blokovaný a program CoAP serveru nepůjde znovu spustit.

6.4.2 Klientská část

Ta je určena k přijímání zpráv od serveru prostřednictvím protokolu CoAP. Klient inicializuje komunikaci se serverem umístěným na IP adrese 192.168.10.2 a portu 5683. Nejprve se registruje konfigurace pro protokoly CoAP a UDP. Uživatel je následně požádán o zadání uživatelského jména (test) a hesla (test123) pro ověření přístupu k serveru, to se jako v případě serveru porovná se souborem passwd.txt, který je uložený ve složce s programem. Poté program vytvoří instanci CoapClient pro komunikaci se serverem, nastaví serverový socket, název zdroje pro komunikaci a zaznamená čas o zahájení komunikace. Po úspěšném nastavení klient naváže komunikaci se serverem a odesílá GET požadavek na server pro přijetí binárního souboru. Server GET požadavek zpracuje a odešle soubor. Pokud je výměna souboru úspěšná, klient uloží přijatý soubor a zaznamená čas uložení souboru. Základní nastavení CoAP klienta a přijetí binárního souboru zobrazuje výpis 6.9.

Výpis 6.9: Přijetí binárního souboru CoAP klientem.

```
public class ServerCoapNotSecure {
    public void ConnectionToClientOverCoap(){
        CoapConfig.register();
        UdpConfig.register();

        String serverSocket = "coap://192.168.10.2:5683/";
        String source = "VehicleCommunication";

        CoapClient client = new CoapClient();
        client.setURI(serverSocket+source);

        CoapResponse responseFile = client.get();

    if (responseFile != null && responseFile.isSuccess()){
        FileSaver.SaveFileCoap(responseFile.getPayload());
    }else {
        System.err.println("Failed to exchange file.");
        return; }
    }
}
```

V případě úspěšného získání a uložení souboru, klient nabídne možnost ukončení spojení s CoAP serverem prostřednictvím zadání „exit“. Pokud uživatel zadá tento požadavek, klient se odpojí od serveru a zaznamená čas ukončení komunikace. Na obrázku 6.7 můžeme vidět průběh komunikace v programu Wireshark mezi CoAP serverem a CoAP klientem.

coap

No.	Source	Destination	Protocol	Source Port	Destination Port	Data	Info
1	192.168.10.3	192.168.10.2	CoAP	65034	5683		CON, MID:17943, GET, TKN:80 40 e3 fe f0 66 1c 5e, /VehicleCommunication
2	192.168.10.2	192.168.10.3	CoAP	5683	65034		ACK, MID:17943, 2.05 Content (text/plain), TKN:80 40 e3 fe f0 66 1c 5e, /VehicleCommunication
3	192.168.10.3	192.168.10.2	CoAP	65034	5683		CON, MID:17944, GET, TKN:f0 3d 03 bf 60 04 46 a6, /SendBinaryFile
4	192.168.10.2	192.168.10.3	CoAP	5683	65034	✓	ACK, MID:17944, 2.05 Content, TKN:f0 3d 03 bf 60 04 46 a6, /SendBinaryFile

Obr. 6.7: Průběh komunikace CoAP bez zabezpečení

6.5 Implementace protokolu CoAP se zabezpečením DTLS

V implementaci zabezpečení pomocí DTLS se využívá knihovna Scandium a modul Element-connector. Pro autentizaci byly opět zvoleny X509 certifikáty a dvojice klíčů. Pro vygenerování certifikátů a klíčů byla znovu využita služba OpenSSL a virtuální systém s operačním systémem Kali. Vytvořené soubory najdeme ve složce příslušných programů. Aby komunikace fungovala muselo se při vytvoření certifikátu v poli CN definovat stejné doménové jméno jaké se používá při komunikaci. Doménové jméno (AVcommunication.com) je mapováno v souboru hosts na IP adresu serveru 198.168.10.2.

6.5.1 Parametry DTLS zabezpečení

Proces autentizace

Autentizace pomocí protokolu DTLS probíhá podobně jako u protokolu TLS s rozdílem, že je využit transportní protokol UDP namísto TCP. Během inicializace spojení každá strana prezentuje svůj certifikát druhé straně, která ho ověřuje. Autentizace se provádí pomocí certifikátů a páru veřejných a soukromých klíčů. Po úspěšné autentizaci strany vytvoří společný symetrický klíč, který je použit k šifrování a dešifrování dat během komunikace.

Kryptografické algoritmy certifikátů

Pro autentizaci mezi CoAP serverem a klientem, má uživatel na výběr ze dvou kryptografických algoritmů využitých při tvorbě certifikátů a klíčů. Kryptografické

algoritmy a jejich parametry jsou stejné jako v předchozí implementaci pomocí protokolu MQTT s TLS a jsou popsány v kapitole 6.3.1 liší se pouze v názvech souborů. Tabulka 6.2 uvádí přehled vytvořených certifikátu a klíčů pro zabezpečení pomocí DTLS.

Název souboru	**Formát souboru**	**Parametry**	**Platnost**
localhostCA	.crt	RSA 2048 b	3650 dní
CA_ECDSA	.crt	ECDSA prime256v1, sha256	3650 dní
CoAP_server_RSA	.p12	RSA 2048 b	364 dní
CoAP_server_ECDSA	.p12	ECDSA prime256v1, sha256	364 dní
CoAP_client_RSA	.p12	RSA 2048 b	364 dní
CoAP_client_ECDSA	.p12	ECDSA prime256v1, sha256	364 dní

Tab. 6.2: Přehled využitých certifikátů a klíčů v protokolu DTLS

Jako certifikační autoritu lze využít již vytvořené CA, a to localhostCA.crt s algoritmem RSA nebo CA_ECDSA využívající eliptických křivek a algoritmu DSA. Certifikáty klienta a serveru byly exportovány do formátu PKCS#12. Ten je podporovaný knihovnou Scandium. Při exportu certifikátu jak pro server, tak pro klienta, bylo nutné definovat heslo a alias. Heslo je **test123**, to je ve všech případech stejné pro všechny certifikáty pro CoAP komunikaci. Alias pro certifikáty serveru je „ServerCoAP“. Alias pro certifikáty klienta je „ClientCoAP“.

Metoda ustanovení klíče

Algoritmus pro zajištění ustanovení klíčů byl použit ECDH, a to jak pro certifikáty vytvořené algoritmem ECDSA, tak i pro certifikáty vytvořené algoritmem RSA. Knihovna Scandium nepodporuje metodu ustanovení klíče algoritmem RSA.

Šifrování a integrita

Pro zajištění šifrování a integrity dat je použita stejná šifra jako u implementace protokolu MQTT. Konkrétně šifra AES s délkou klíče 256 bitů a módem GCM. Integritu zajišťuje kryptografická hashovací funkce SHA s délkou klíče 384 bitů.

6.5.2 Serverová část

Kód serverové části je navržen k vytvoření zabezpečeného serveru, který komunikuje s klientem pomocí protokolu DTLS. Nejprve náš server registruje potřebné konfigurační parametry pro protokoly CoAP a DTLS. Následně dojde k načtení certifikátu

a klíče pro server z odpovídajícího souboru dle volby kryptografického algoritmu a certifikátu CA. Po nastavení konfigurace a načtení certifikátu je vytvořen a zároveň inicializován DTLS builder, ten umožňuje nastavit různé parametry spojení jako port nebo povolené šifry. Metodou SingleCertificateProvider vytvoříme poskytovatele certifikátu, který obsahuje identitu serveru. Tento certifikát bude použit k ověření serveru v rámci DTLS spojení. Pomocí NewAdvancedCertificateVerifier se vytvoří ověřovatel certifikátu v našem případě CA, ta bude použita k ověření certifikátů klienta. Nakonec se vytvoří instance DTLS konektoru na základě zkonfigurovaného builderu. V případě využití protokolu DTLS musel být, na straně serveru vytvořený endpoint. Výpis 6.10 ukazuje způsob načtení certifikátu serveru a CA, konfiguraci DTLS spojení a vytvoření Endpointu pro komunikaci.

Výpis 6.10: Načtení certifikátu a konfigurace DTLS zabezpečení.

```
public class ClientCoapDtls {                                                    1
    public void ConnectionToServerOverCoAPwithDTLS(){                            2
                                                                                 3
    String cert = "CoAP_client_RSA.p12";                                         4
    String alias = "ClientCoAP";                                                 5
    char[] keystorePassword = "test123".toCharArray();                           6
    char[] keyPassword = "test123".toCharArray();                                7
    String truststore = "localhostCA.crt";                                       8
                                                                                 9
    Credentials serverCredentials = SslContextUtil                               10
    .loadCredentials(cert,alias,keystorePassword,keyPassword);                   11
                                                                                 12
    Certificate[] caCredentials = SslContextUtil                                 13
    .loadTrustedCertificates(truststore);                                        14
                                                                                 15
    DtlsConnectorConfig.Builder builder = DtlsConnectorConfig                    16
    .builder(conf);                                                              17
    builder.setAddress(new InetSocketAddress(5684));                             18
                                                                                 19
    builder.setAsList(DtlsConfig.DTLS_CIPHER_SUITES,                             20
    CipherSuite.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384);                          21
                                                                                 22
    CoapEndpoint endpoint = new CoapEndpoint.Builder()                           23
    .setConnector(connector).build();                                            24
                                                                                 25
    CoapServer server = new CoAPServer();                                        26
    server.addEndpoint(endpoint);                                                27
    server.start(); }}                                                           28
```

Coap server s DTLS konektorem je následně spuštěn. Server naslouchá na portu 5684 a čeká na připojení klienta. Po připojení klienta, server vytvoří zdroj pro komunikaci a na základě metody HandleGET odešle soubor klientovi jako v případě nezabezpečené verze implementace protokolu CoAP. Logování časů a možnost ukončení komunikace funguje stejně jako u nezabezpečené verze.

6.5.3 Klientská část

Program CoAP klienta komunikuje s CoAP serverem skrze protokol DTLS s využitím certifikátu a soukromého klíče. Klient využívá certifikát k ověření své identity vůči serveru a k zahájení bezpečného spojení. Kód pro práci s certifikáty je stejný jako pro serverovou část. Klient nejprve inicializuje a registruje konfigurační parametry pro UDP, CoAP a DTLS protokoly. Načte certifikát, soukromý klíč, alias a heslo z PKCS12 souboru. Dál se nastaví důvěryhodný certifikát CA pro ověření serveru. Vytvoří se a inicializuje DTLSConnector s konfigurací z předchozího kroku.

Pro zahájení komunikace klient vytvoří Endpoint a instanci CoapClient, kde přidá vytvořený endpoint, nastaví URI adresu serveru a zvolený zdroj. Endpoint je síťový uzel, který může odesílat nebo přijímat CoAP zprávy. Bez vytvoření endpointu komunikace pomocí DTLS se sice navázala, ale nešlo odesílat a přijímat GET požadavky. URI adresa serveru je nyní doménové jméno AVcommunication.com a port 5684.

Dále je implementovaný stejný kód klienta jako v případě nezabezpečené verze. Zabezpečený klient tedy dokáže přijímat binární soubory a zaznamenávat časový průběh komunikace do logu. Rovněž dokáže ukončit spojení se serverem zadáním „exit“. Obrázek 6.8 ukazuje průběh navazování komunikace protokolem DTLS mezi klientem a serverem v programu Wireshark.

No.	Source	Destination	Protocol	Source Port	Destination Port	Data	Info
1	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Client Hello
2	192.168.10.2	192.168.10.3	DTLSv1.2	5684	49915		Hello Verify Request
3	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Client Hello
4	192.168.10.2	192.168.10.3	DTLSv1.2	5684	49915		Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
5	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
6	192.168.10.2	192.168.10.3	DTLSv1.2	5684	49915		Change Cipher Spec, Encrypted Handshake Message
7	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Application Data
8	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Application Data
9	192.168.10.2	192.168.10.3	DTLSv1.2	5684	49915		Application Data
10	192.168.10.2	192.168.10.3	DTLSv1.2	5684	49915		Application Data
11	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Application Data
12	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Application Data
17	192.168.10.3	192.168.10.2	DTLSv1.2	49915	5684		Application Data
18	192.168.10.2	192.168.10.3	DTLSv1.2	5684	49915		Application Data

Obr. 6.8: Průběh komunikace CoAP se zabezpečením DTLS

7 Měření zpoždění přenosu a zhodnocení

V této kapitole bylo provedeno výkonnostní zhodnocení navržených řešení na základě několika měření času v závislosti na velikosti přenášených binárních souborů, typu protokolů pro přenos a jeho možnosti zabezpečení.

7.1 Příprava měření

Získání času

Pro měření bylo využito funkce LocalDateTime pro získání času, ta je implementována v obou programech serveru a klienta. Funkce získává systémový čas z operačního systému Windows, na kterém je aplikace spuštěna. Ten byl před zahájením měření na obou zařízeních synchronizovaný ručně přes ovládací panely a nastaven na stejnou hodnotu. Automatické nastavení času a jeho synchronizace s externími časovými servery nebyla přesná a lišila se na obou zařízeních.

Vlastnosti hardwaru

Před zahájením samotného měření je nutné specifikovat hardware obou použitých zařízení. V textu níže jsou uvedeny hardwarové specifikace a vlastnosti obou zařízení, které mohou ovlivnit výsledky měření.

Stolní počítač: osmi jádrový procesor Intel(R) Core(TM) i7-4820K s kmitočtem 3,70 GHz; nainstalovaná paměť RAM 16 GB; 64 bitový operační systém Windows 10 Pro; síťová karta Interl(R) 82579V Gigabit Network Connection s rychlostí spojení 100/100 Mb za sekundu.

Notebook: čtyř jádrový procesor Intel(R) Core(TM) i3-10110U s kmitočtem 2,10 GHz; nainstalovaná paměť RAM 8GB; 64 bitový operační systém Windows 10 Pro Education; síťová karta Realtek PCIe GbE Family Controller s rychlostí spojení 100/100 Mb za sekundu.

Přepínač: model S105 od společnosti Tenda; celková propustnost 1Gb; přenosová rychlost 10/100 Mb za sekundu.

Části měření

Měření probíhalo ve dvou částech. První část je rozdíl časů mezi zahájením komunikace a odesláním binárního souboru. V této části byla analýza zpoždění zaměřena na navázání komunikace při použití nezabezpečené i zabezpečené implementace. U zabezpečené implementace byla zkoumána i volba kryptografického algoritmu a jeho parametry při vytvoření certifikátu.

Druhá část zkoumá míru velikosti latence mezi samotným odesláním binárního souboru ze serveru a jeho přijetím klientem s následným uložením. Latence se zde zkoumá v závislosti na velikosti binárního souboru a rozdílů v použití protokolu pro přenos, včetně možnosti využití nezabezpečené i zabezpečené verze řešení a jejich nastavení kryptografických parametrů.

Úprava kódu

Pro přesnější měření bylo nutné lehce upravit kód serveru, aby bylo možné změřit přesné časové údaje. Server při úpravě pro měření zpoždění nyní nečeká na zadaní názvu souboru pro odeslání při spuštění programu, ale rovnou je tento název potřeba uvést v kódu. Rovněž je zde upravena nekonečná smyčka, tak že se program vždy uživatele zeptá zda-li chce odejít. V případě zadání „exit“ dojde k zaznamenání času ukončení spojení. Tyto úpravy jsou ve výchozím použití programu zakomentovány pod názvem „//test use“. Klientská část programu zůstává beze změny.

Vzorečky a výpočty

U každé implementace je vypočítán rozdíl časů mezi navázáním komunikace a odesláním binárního souboru. V prvním případě byl proveden výpočet rozdílu časů na základě vzorce (7.1).

$$\Delta t = |t_2 - t_1| \tag{7.1}$$

Následně byla vypočítána průměrná doba na základě rozdílu času mezi navázáním komunikace a odesláním souboru. Pro tento výpočet byl použit vzorec pro aritmetický průměr (7.2).

$$\bar{t} = \frac{t_1 + t_2 + t_3 + \cdots + t_n}{n} \tag{7.2}$$

Stěžejní částí měření je velikost latence mezi odesláním souboru ze serveru a jeho přijetím klientem, kde se následně uloží v závislosti na velikosti přenášených souborů. Pro výpočet velikosti latence byl použit vzorec (7.3).

$$\Delta L = |t_p - t_o| \tag{7.3}$$

Posledním výpočtem je průměrná doba latence mezi odeslanými binárními soubory ze serveru a jejich přijetí klientem. Pro tento výpočet sloužil vzorec (7.4).

$$\overline{L} = \frac{1}{n} \sum_{i=1}^{n} L_i \tag{7.4}$$

7.2 Výsledky měření protokol MQTT

Na základě logů byl vypočítán rozdíl časů mezi spuštěním klienta publishera, který navazuje komunikace s brokerem Mosquitto a dobou před samotnou publikací zprávy.

Jako vstup pro výpočet rozdílů času bylo použito 8 časových údajů. Na základě těchto údajů byla vypočítána průměrná hodnota doby navázání komunikace pro nezabezpečenou verzi i zabezpečené verze protokolu MQTT. Průměrnou dobu navázání komunikace ukazuje tabulka 7.1 níže.

Parametry zabezpečeného spojení TLS s RSA: autentizace RSA, ustanovení klíče RSA, šifrování AES 256 GCM, integrita SHA 384.

Parametry zabezpečeného spojení TLS s ECDSA: autentizace ECDSA, ustanovení klíče ECDH, šifrování AES 256 GCM, integrita SHA 384.

Bez zabezpečení	**MQTT + TLS s RSA**	**MQTT + TLS s ECDSA**
$\bar{t} = 0{,}442003950$ s	$\bar{t} = 0{,}790503450$ s	$\bar{t} = 0{,}774738800$ s

Tab. 7.1: Průměrná doba navázání komunikace protokolu MQTT

7.2.1 Protokol MQTT bez zabezpečení

V této části bylo měření zaměřeno na velikost latence při komunikaci mezi klientem publisherem a klientem subscriberem prostřednictvím protokolu MQTT a brokera Mosquitto. Latence je v tomto případě doba zpoždění mezi publikováním zprávy klientem a jejím přijetím klientem na straně druhé, kde se následně uloží. Měření probíhalo v závislosti na velikosti přenášených zpráv. Tabulka 7.2 udává naměřené hodnoty latence protokolu MQTT bez zabezpečení.

Zpráva v [B]	**Odeslání zprávy**	**Přijetí zprávy**	**Latence [s]**
1	0,677866200	0,846031800	0,168165600
10	0,362705400	0,482191700	0,119486300
100	0,419447000	0,560872000	0,141425000
1024	0,406555600	0,535578100	0,129022500
10240	0,605116500	0,697414200	0,092297700
102400	0,562869200	0,690401000	0,127531800
1048576	0,562291300	0,802094300	0,239803000
10485760	0,954925700	2,406233400	1,451307700

Tab. 7.2: Latence protokolu MQTT bez zabezpečení

7.2.2 Protokol MQTT se zabezpečením TLS

Při využití zabezpečení protokolem TLS v kombinaci s MQTT protokolem a brokerem Mosquitto se analýza míry latence stává klíčovou. Pro zabezpečení komunikace

byla použita metoda certifikátů, přičemž v prvním případě byl certifikát vytvořen pomocí algoritmu RSA a v druhém případě pomocí algoritmu ECDSA. Oběma klientům byl přidělený daný certifikát na základě volby algoritmu a bylo vytvořené TLS spojení. V následujících tabulkách jsou prezentovány naměřené hodnoty latence v závislosti na velikosti přenášených zpráv a použití kryptografického algoritmu. Tabulka 7.3 zobrazuje naměřené hodnoty pro zabezpečené spojení kombinací protokolu MQTT s protokolem TLS a kryptografickým algoritmem RSA. Naměřené hodnoty latence s využitím kryptografického algoritmu ECDSA popisuje tabulka 7.4.

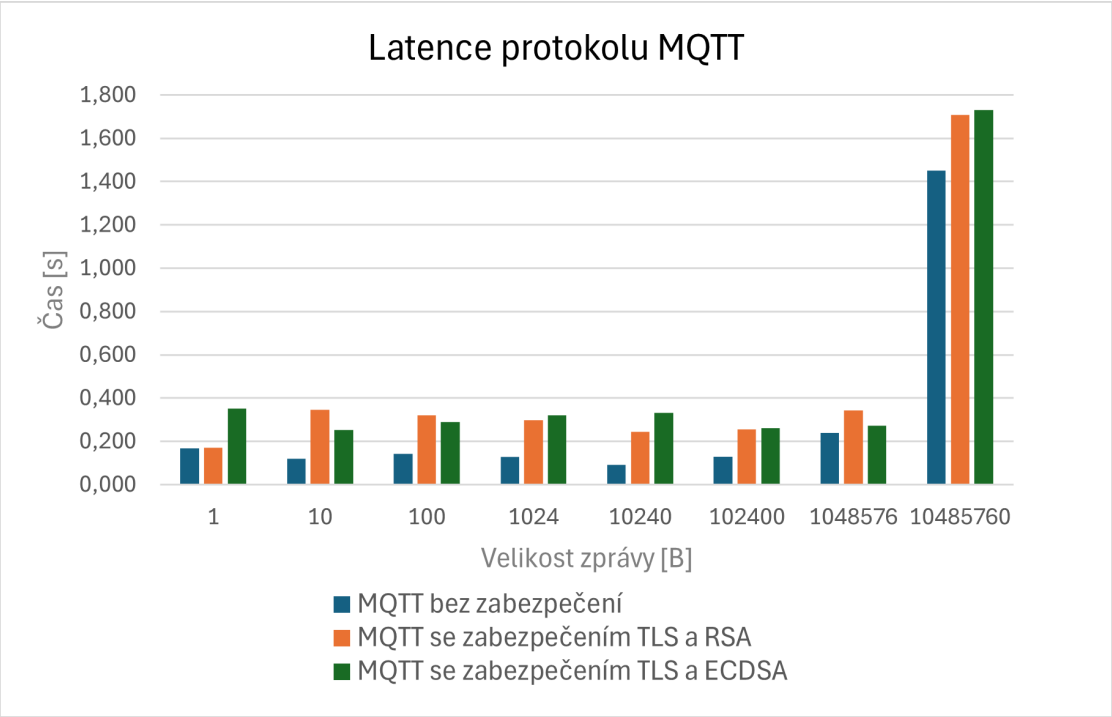
Zpráva v [B]	**Odeslání zprávy**	**Přijetí zprávy**	**Latence [s]**
1	0,769109100	0,940899800	0,171790700
10	0,294503600	0,641222700	0,346719100
100	0,476874300	0,798233000	0,321358700
1024	0,034059800	0,332734100	0,298674300
10240	0,033912800	0,276671800	0,242759000
102400	0,450551400	0,705017800	0,254466400
1048576	0,821076900	1,163978500	0,342901600
10485760	0,447979800	2,156588900	1,708609100

Tab. 7.3: Latence protokolu MQTT se zabezpečením algoritmus RSA

Zpráva v [B]	**Odeslání zprávy**	**Přijetí zprávy**	**Latence [s]**
1	0,822429400	1,172659900	0,350230500
10	0,268503200	0,522084000	0,253580800
100	0,626645800	0,914877000	0,288231200
1024	0,962850300	1,284033800	0,321183500
10240	0,084086300	0,414982600	0,330896300
102400	0,859870900	1,121418300	0,261547400
1048576	0,720651400	0,993909600	0,273258200
10485760	0,024361400	1,753884000	1,729522600

Tab. 7.4: Latence protokolu MQTT se zabezpečením algoritmus ECDSA

Hodnoty z tabulek 7.2, 7.3, 7.4 byly následně vneseny do grafického znázornění, které zobrazuje obrázek 7.1.



Obr. 7.1: Grafické znázornění latence protokolu MQTT

7.3 Výsledky měření protokol CoAP

Pro protokol CoAP byl rovněž změřen časový rozdíl mezi navázáním komunikace CoAP serveru s CoAP klientem. Opět bylo použito měření skládající se z 8 časových údajů. Průměrnou dobu navázání komunikace skrze protokol CoAP v nezabezpečené verzi a ve dvou zabezpečených verzích pomocí DTLS ukazuje tabulka 7.5.

Parametry zabezpečeného spojení DTLS s RSA: autentizace RSA, ustanovení klíče ECDH, šifrování AES 256 GCM, integrita SHA 384.

Parametry zabezpečeného spojení DTLS s ECDSA: autentizace ECDSA, ustanovení klíče ECDH, šifrování AES 256 GCM, integrita SHA 384.

Bez zabezpečení	**CoAP + DTLS s RSA**	**CoAP + DTLS s ECDSA**
$\bar{t} = 0{,}489516975$ s	$\bar{t} = 0{,}883374875$ s	$\bar{t} = 0{,}801493075$ s

Tab. 7.5: Průměrná doba navázání komunikace protokolu CoAP

7.3.1 Protokol CoAP bez zabezpečení

V této kapitole byla prozkoumána a změřena velikost latence mezi odesláním zprávy z CoAP serveru a jejím přijetím CoAP klientem. Byl prozkoumaný vliv velikosti přenášených zpráv na míru latence. Z naměřených hodnot byla vytvořena tabulka 7.6.

Zpráva v [B]	**Odeslání zprávy**	**Přijetí zprávy**	**Latence [s]**
1	0,710811000	0,760318400	0,049507400
10	0,270429000	0,311410100	0,040981100
100	0,270162000	0,306679200	0,036517200
1024	0,687573300	0,725012500	0,037439200

Tab. 7.6: Latence protokolu CoAP bez zabezpečení

7.3.2 Protokol CoAP se zabezpečením DTLS

Implementace protokolu CoAP byla doplněna o zabezpečení DTLS. V této kapitole byla analyzována a změřena velikost latence mezi odesláním zprávy z CoAP serveru a jejím přijetím CoAP klientem prostřednictvím zabezpečeného kanálu protokolem DTLS. Míra latence byla měřena v závislosti na velikosti přenášených zpráv. Na základě naměřených hodnot byla vytvořena tabulka 7.7, která popisuje míru latence s využitím protokolu DTLS a kryptografického algoritmu RSA. Naopak tabulka 7.8 podrobně popisuje míru latence při použití protokolu DTLS v kombinaci s kryptografickým algoritmem ECDSA.

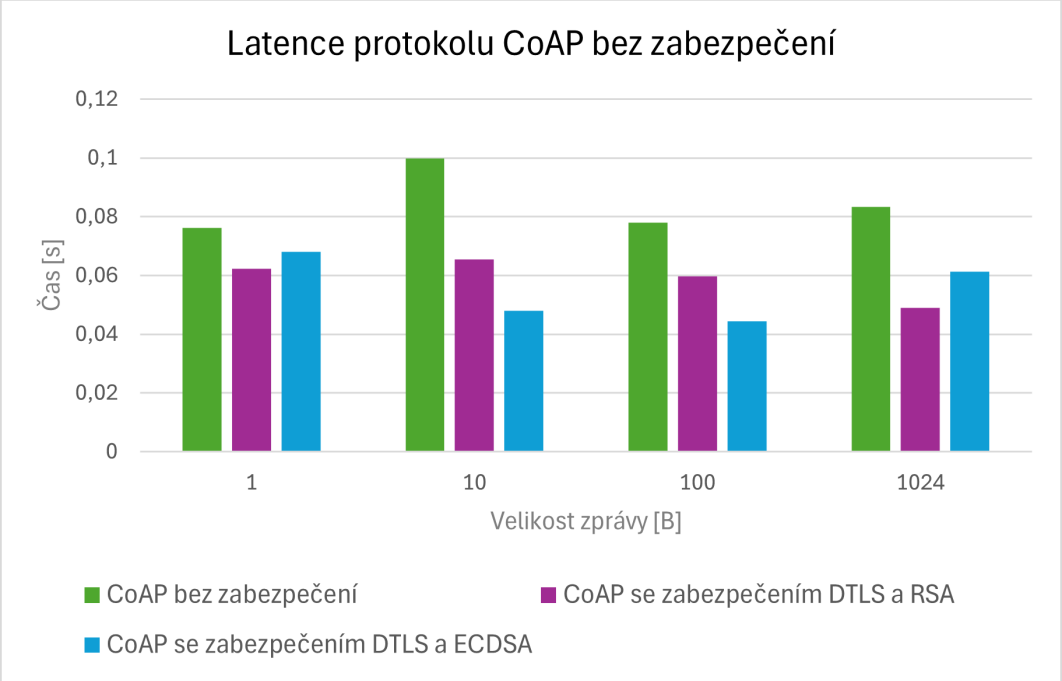
Zpráva v [B]	**Odeslání zprávy**	**Přijetí zprávy**	**Latence [s]**
1	0,675195200	0,737530300	0,062335100
10	0,625300900	0,690810600	0,065509700
100	0,956281300	1,015987700	0,059706400
1024	0,056680100	0,105664400	0,048984300

Tab. 7.7: Latence protokolu CoAP se zabezpečením algoritmus RSA

Zpráva v [B]	Odeslání zprávy	Přijetí zprávy	Latence [s]
1	0,059663300	0,127781300	0,068118000
10	0,688677900	0,736657800	0,047979900
100	0,522935300	0,567427900	0,044492600
1024	0,040621500	0,101860100	0,061238600

Tab. 7.8: Latence protokolu CoAP se zabezpečením algoritmus ECDSA

Zjištěné hodnoty z tabulek 7.6, 7.7, 7.8 byly následně využity k vytvoření grafu, jenž je zobrazen na obrázku 7.2. Tento graf poskytuje vizuální interpretaci naměřených výsledků.



Obr. 7.2: Grafické znázornění latence protokolu CoAP

7.4 Zhodnocení výsledků

Tato kapitola se zaměřuje na zhodnocení výsledků měření navržených protokolů. Protokoly, které byly testovány, jsou MQTT a CoAP. Níže je provedeno zhodnocení jejich nezabezpečených i zabezpečených verzí. Pro zajištění konzistentních a srovnatelných výsledků byly oba protokoly testovány v kontrolovaném prostředí s následujícími parametry měření:

- Průměrná doba navázání spojení: Byl měřen čas potřebný k navázání spojení mezi klientem a serverem;
- Míra latence: Byla měřena míra latence při výměně zpráv mezi klientem a serverem;
- Maximální velikost přenášených zpráv: Byla testována maximální možná velikost zprávy, kterou lze úspěšně přenést zvoleným protokolem.

Průměrná doba navázání spojení

Pro nezabezpečenou verzi MQTT byla naměřena průměrná doba navázání spojení mírně pod půl vteřiny. U protokolu MQTT byla doba navázání spojení kratší než u protokolu CoAP, a to o $\bar{t} \doteq 9{,}7$ %. S přidáním šifrování pomocí RSA nebo ECDSA u protokolu MQTT došlo k prodloužení této doby na téměř dvojnásobnou hodnotu. Kdy průměrná doba navazování spojení u algoritmu ECDSA je o $\bar{t} \doteq 2$ % rychlejší než u algoritmu RSA. V případě zabezpečení verze protokolu prostřednictvím DTLS nad protokolem CoAP se doba navázání spojení rovněž zvýšila skoro o polovinu času, než v případě nezabezpečené verze protokolu CoAP. Podobně jako u protokolu MQTT byla verze s algoritmem ECDSA rychlejší než RSA, a to o $\bar{t} \doteq 9{,}3$ %.

Velikost zpráv

Pomocí protokolu MQTT můžeme posílat zprávy o velikosti až 256 MB, zatímco protokolem CoAP ve výchozím nastavení pouze zprávy o maximální velikosti 1152 B. Maximální velikost přenášených zpráv, tak může ovlivnit možnosti přenosu dat mezi teleoperačním centrem a autonomním vozidlem.

Velikost latence

Pro výpočet průměrné velikosti latence protokolů CoAP a MQTT byly brány pouze hodnoty odpovídající velikosti zpráv 1, 10, 100 a 1024 B.

Na základě těchto hodnot bylo vypočítáno, že nezabezpečená verze protokolu CoAP vykazuje nižší průměrnou latenci než protokol MQTT, a to o $\overline{L} \doteq 40$ %. Průměrné naměřené hodnoty velikosti latence protokolu MQTT a CoAP bez zabezpečení zobrazuje tabulka 7.9.

MQTT bez zabezpečení	**CoAP bez zabezpečení**
$\overline{L} = 0{,}139524850$ s	$\overline{L} = 0{,}084341200$ s

Tab. 7.9: Průměrné míra latence nezabezpečených verzí protokolů

Zabezpečená verze protokolu MQTT pomocí TLS vykazuje vyšší průměrnou latenci než verze bez přidaného zabezpečení, konkrétně o $\overline{L} \doteq 51$ % v případě použití algoritmu RSA a o $\overline{L} \doteq 54$ % v případě ECDSA. Průměrné hodnoty latence zabezpečené verze protokolu MQTT zobrazuje tabulka 7.10. Přidané zabezpečení pomocí algoritmu ECDSA bylo v tomto případě mírně pomalejší než zabezpečení algoritmem RSA přesněji o $\overline{L} \doteq 6$ %. I když ECDSA a ECDH obecně poskytují vyšší úroveň bezpečnosti a efektivity z hlediska velikosti klíčů, konkrétní implementace vedla k tomu, že RSA vykazuje nižší průměrnou míru latence.

MQTT + TLS s RSA	**MQTT + TLS s ECDSA**
$\overline{L} = 0{,}284635700$ s	$\overline{L} = 0{,}303306500$ s

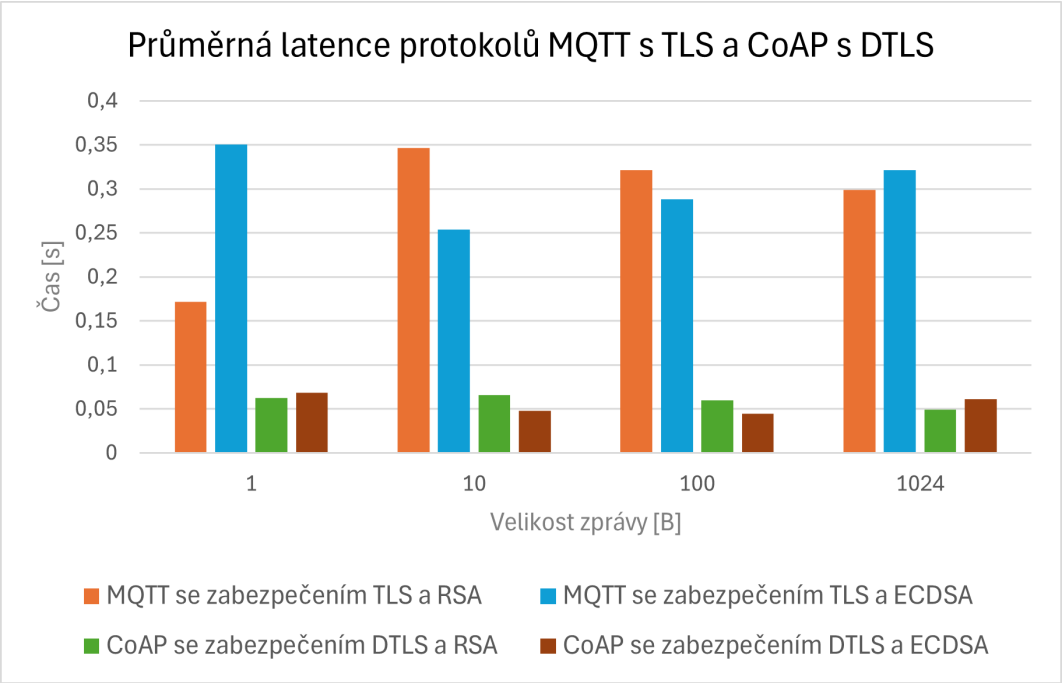
Tab. 7.10: Průměrné míra latence MQTT se zabezpečením

Zabezpečená verze protokolu CoAP pomocí DTLS překvapivě vykázala nižší průměrnou latenci než nezabezpečená verze. Průměrné hodnoty latence zabezpečené verze protokolu CoAP zobrazuje tabulka 7.11. Přidané zabezpečení pomocí algoritmu ECDSA bylo v tomto případě srovnatelné se zabezpečení algoritmem RSA. Nicméně algoritmus ECDSA poskytoval mírně menší zpoždění přesněji o $\overline{L} \doteq 6$ %.

CoAP + DTLS s RSA	**CoAP + DTLS s ECDSA**
$\overline{L} = 0{,}059133875$ s	$\overline{L} = 0{,}055457275$ s

Tab. 7.11: Průměrné míra latence CoAP se zabezpečením

Shrnutí naměřených výsledků zabezpečených verzí protokolů MQTT a CoAP a jejich zhodnocení zobrazuje obrázek 7.3. Z grafu je patrné, že míra zpoždění při posílání binárních souborů protokolem CoAP se zabezpečením DTLS je výrazně menší než při použití protokolu MQTT se zabezpečením TLS.



Obr. 7.3: Grafické znázornění průměrné latence

7.5 Doporučení pro praxi

Na základě analýzy a následného zhodnocení průměrné doby navázání spojení, velikosti zpráv a míry latence pro zabezpečené verze protokolu MQTT a CoAP, lze formulovat následující doporučení pro praxi.

Pro komunikaci, kde je zásadní nízká latence a bezpečnost, je doporučeno použít CoAP s DTLS a ECDSA. Tento protokol poskytuje nejnižší latenci při zabezpečení a je vhodný pro kritické operace autonomního vozidla. Pro komunikaci vyžadující přenos větších objemů dat (více než 1152 B), kde mírně vyšší latence není problémem je doporučeno použít MQTT se zabezpečením TLS. Tento protokol umožňuje přenos velmi velkých zpráv oproti CoAP s DTLS a poskytuje srovnatelnou bezpečnost.

Výběr protokolu pro komunikaci mezi teleoperačním centrem a autonomním vozidlem by měl být založen na specifických požadavcích aplikace na latenci, velikosti přenášených dat a úrovní zabezpečení. Výše uvedená doporučení poskytují vodítko pro optimalizaci těchto faktorů a zajištění efektivní a bezpečné komunikace v reálných podmínkách.

Závěr

Autonomní vozidla představují inovativní a perspektivní směr v oblasti dopravy a technologií v dnešním moderním světě. Cílem bakalářské práce bylo analyzovat bezpečnostní hrozby, útoky a protiopatření v oblasti autonomní dopravy a navrhnout řešení umožňující bezpečné vzdálené řízení těchto vozidel s jeho výkonnostním zhodnocením.

Teoretická část nám poskytla pevný základ pro pochopení problematiky autonomních vozidel, od úrovní autonomie, přes společnosti zabývající se vývojem a provozem těchto vozidel. Dále analýzu komunikace v autonomních vozidlech a její kybernetickou bezpečnost v podobě identifikace zranitelností, druhu kybernetických útoků a možnosti protiopatření.

V praktické části byly rozebrány aspekty vzdáleného řízení. Byl představen příklad komunikačního modelu pro vzdálené řízení včetně provedení analýzy rizik. Rovněž byla provedena analýza, zhodnocení a výběr vhodných kryptografických protokolů pro bezpečné vzdálené řízení. Na základě této analýzy byly vybrány protokoly Message Queuing Telemetry Transport a Constrained Application Protocol, které byly následně implementovány jako aplikace simulující komunikaci mezi teleoperačním centrem a autonomním vozidlem. Tyto protokoly byly rozšířeny o možnost zabezpečení pomocí protokolů Transport Layer Security či Datagram Transport Layer Security. Toto zabezpečení doplňuje komunikaci o autentizaci, důvěrnost, integritu, šifrování a další bezpečnostní aspekty. Navržené protokoly včetně jejich doplněného zabezpečení byly výkonnostně testovány v několika rovinách komunikace. Zároveň zde bylo poukázáno na to, jaký vliv má velikost zpoždění na čas potřebný pro navázání komunikace a odeslání binárních zpráv. V testování bylo zohledněno i různé použití kryptografických algoritmů využitých při zabezpečení komunikace.

Na základě výkonnostního zhodnocení byla představena optimální volba protokolu a jeho kryptografických algoritmů pro bezpečnou komunikaci. Protokol CoAP s implementovaným DTLS a algoritmem ECDSA byl identifikován jako optimální volba pro situace, kde je klíčová nízká latence a zároveň bezpečná komunikace. Naopak protokol MQTT se zabezpečením TLS se ukázal jako vhodný pro situace, které vyžadují větší přenos dat než 1152 B a menší než 256 MB.

Tato práce poskytuje základní analýzu a srovnání protokolů MQTT a CoAP s jejich možnostmi zabezpečení pro komunikaci s autonomními vozidly. Pro další rozšíření práce je možné zvážit testování protokolů v reálném nasazení, použití jiných kryptografických algoritmů pro poskytnutí co možná nejnižší míry zpoždění komunikace.

Literatura

[1] SYNOPSYS. *The 6 Levels of Vehicle Autonomy Explained*. Online. ©2023. Dostupné z: https://www.synopsys.com/automotive/autonomous-driving-levels.html. [cit. 2023-10-21].

[2] SAE MOBILUS. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Online. 2021. Dostupné z: https://doi.org/10.4271/J3016_202104. [cit. 2023-10-21].

[3] JANSKÝ, Martin. *Autonomní řízení: Pět kroků k autům bez řidiče*. Online. In: GARAZ.CZ. 2022. Dostupné z: https://www.garaz.cz/clanek/reportaze-autonomni-rizeni-pet-kroku-k-autum-bez-ridice-21007584. [cit. 2023-10-22].

[4] AUTONOMNE.CZ. *Kdo všechno autonomní vozidla vyvíjí?* Online. ©2023. Dostupné z: https://www.autonomne.cz/front/homepage/index-one?index_id=10. [cit. 2023-11-07].

[5] AUTONOMNE.CZ. *Příběh průkopníka*. Online. 2020. Dostupné z: https://www.autonomne.cz/front/homepage/article-detail?article_id=77. [cit. 2023-11-07].

[6] AUTONOMNE.CZ. *Žebříček vývojářů autonomních nákladních vozidel*. Online. 2022 Dostupné z: https://www.autonomne.cz/front/homepage/article-detail?article_id=114. [cit. 2023-11-07].

[7] LENAGHAN, Amanda. *Partnership is essential to powering Cruise for Good: Behind the scenes of our meal delivery program*. Online. In: CRUISE LLC. 2022. Dostupné z: https://getcruise.com/news/blog/2022/partnership-is-essential-to-powering-cruise-for-good/. [cit. 2023-11-07].

[8] AUTONOMNE.CZ. *Cruise pozastavuje veškerou činnost svých autonomních vozů v USA*. Online. 2023. Dostupné z: https://www.autonomne.cz/front/homepage/news-detail?news_id=598. [cit. 2023-11-07].

[9] AUTONOMNE.CZ. *Zoox nasazuje vozidla bez bezpečnostního řidiče*. Online. 2023 Dostupné z: https://www.autonomne.cz/front/homepage/news-detail?news_id=556. [cit. 2023-11-10].

[10] AUTONOMNE.CZ. *Aurora připravuje vlastní robotaxi*. Online. 2022 Dostupné z: https://www.autonomne.cz/front/homepage/news-detail?news_id=357. [cit. 2023-11-10].

[11] KOON, John. *How Many Sensors For Autonomous Driving?* Online. In: SEMICONDUCTOR ENGINEERING. 2023. Dostupné z: https://semiengineering.com/how-many-sensors-for-autonomous-driving/. [cit. 2023-11-10].

[12] IGNATIOUS, Henry Alexander; SAYED, Hesham-El; KHAN, Manzoor. *An overview of sensors in Autonomous Vehicles*. Online. In: SCIENCEDIRECT. 2022. Dostupné z: https://doi.org/10.1016/j.procs.2021.12.315. [cit. 2023-11-10].

[13] CADENCE SYSTEM ANALYSIS. *The Use of RADAR Technology in Autonomous Vehicles*. Online. In: CADENCE DESIGN SYSTEMS. ©2022. Dostupné z: https://resources.system-analysis.cadence.com/blog/msa2022-the-use-of-radar-technology-in-autonomous-vehicles. [cit. 2023-11-10].

[14] WANG, Zhangjing; WU, Yu; NIU, Qingqing. *Multi-Sensor Fusion in Automated Driving: A Survey*. Online. In: IEEE. 2020. Dostupné z: https://doi.org/10.1109/ACCESS.2019.2962554. [cit. 2023-11-10].

[15] VAZQUEZ, Marta Martinez. *Radar For Automotive: Why Do We Need Radar?* Online. In: SEMICONDUCTOR ENGINEERING. 2022. Dostupné z: https://semiengineering.com/radar-for-automotive-why-do-we-need-radar/. [cit. 2023-11-10].

[16] SYNOPSYS. *What is LiDAR?* Online. ©2023. Dostupné z: https://www.synopsys.com/glossary/what-is-lidar.html. [cit. 2023-10-21].

[17] FEDERAL AVIATION ADMINISTRATION [FAA]. *Satellite Navigation - GPS - How It Works*. Online. 2022. Dostupné z: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks. [cit. 2023-10-21].

[18] GARMIN. *Co je to GPS?* Online. ©1996-2023. Dostupné z: https://www.garmin.com/cs-CZ/aboutgps/. [cit. 2023-10-21].

[19] ULLAH, Hanif; NAIR, G. Nithya; MOORE, Adrian; NUGENT, Chris; MUSCHAMP, Paul. et al. *5G Communication: An Overview of Vehicle-to-Everything, Drones, and Healthcare Use-Cases*. Online. In: IEEE. 2019. Dostupné z: https://doi.org/10.1109/ACCESS.2019.2905347. [cit. 2023-11-12].

[20] TUOHY, Shane; GLAVIN, Martin; HUGHES, Ciarán; JONES, Edward; TRIVEDI, Mohan. et al. *Intra-Vehicle Networks: A Review*. Online. In: IEEE. 2014. Dostupné z: https://doi.org/10.1109/TITS.2014.2320605. [cit. 2023-11-10].

[21] CHEN, Huimin; LIU, JiaJia; WANG, Jiadai; XUN Yijie. *Towards secure intra-vehicle communications in 5G advanced and beyond: Vulnerabilities, attacks and countermeasures*. Online. In: SCIENCEDIRECT. 2023. Dostupné z: https://doi.org/10.1016/j.vehcom.2022.100548. [cit. 2023-11-10].

[22] DEMBA, Albert; MÖLLER, Deitmar. *Vehicle-to-Vehicle Communication Technology*. Online. In: IEEE. 2018. Dostupné z: https://doi.org/10.1109/EIT.2018.8500189. [cit. 2023-11-11].

[23] ZEADALLY, Sherali; GUERRERO, Juan; CONTRERAS, Juan. *A tutorial survey on vehicle-to-vehicle communications*. Online. In: SPRINGER LINK. 2020. Dostupné z: https://doi.org/10.1007/s11235-019-00639-8. [cit. 2023-11-11].

[24] ARENA, Fabio; PAU, Giovanni. *An Overview of Vehicular Communications*. Online. In: MDPI. 2019. Dostupné z: https://doi.org/10.3390/fi11020027. [cit. 2023-11-12].

[25] KANTHAVEL, Dhaya; SANGEETHA, S.K.B.; KEERTHANA, K.P. *An empirical study of vehicle to infrastructure communications - An intense learning of smart infrastructure for safety and mobility*. Online. In: SCIENCEDIRECT. 2021. Dostupné z: https://doi.org/10.1016/j.ijin.2021.06.003. [cit. 2023-11-12].

[26] SEWALKAR, Parag; SEITZ Jochen. *Vehicle-to-Pedestrian Communication for Vulnerable Road Users: Survey, Design Considerations, and Challenges*. Online. In: MDPI. 2019. Dostupné z: https://doi.org/10.3390/s19020358. [cit. 2023-11-12].

[27] KABIL, Ahmad; RABIEH, Khaled; KALEEM, Faisal; AZER, A. Marianne. *Vehicle to Pedestrian Systems: Survey, Challenges and Recent Trends*. Online. In: IEEE. 2022. Dostupné z: https://doi.org/10.1109/ACCESS.2022.3224772. [cit. 2023-11-12].

[28] TEAM EVERYTHINK RF. *What is V2N?* Online. In: EVERYTHINKRF. 2022. Dostupné z: https://www.everythingrf.com/community/what-is-v2n. [cit. 2023-11-13].

[29] GIORDANI, Marco; ZANELLA, Andrea; HIGUCHI, Takamasa; ALTINTAS, Onur; ZORZI, Michele. *Performance study of LTE and mmWave in vehicle-to-network communications*. Online. In: IEEE. 2018. Dostupné z: https://doi.org/10.23919/MedHocNet.2018.8407093. [cit. 2023-11-13].

[30] VODIČKA, Milan; DVOŘÁK, František. *Hackeři unesli jeep i s řidičem, 1,4 milionu aut proto jede do servisu*. Online. In: IDNES.CZ. 2015. Dostupné z: https://www.idnes.cz/auto/zpravodajstvi/hackeri-unesli-jeep-dalkove-ovladani-auta.A150723_135910_automoto_fdv. [cit. 2023-11-18].

[31] EL-REWINI, Zeinab; SADATSHARAN, Karthikeyan; SELVARAJ, Flora Daisy; PLATHOTTAM, Siby Jose; RANGANATHAN, Prakash. *Cybersecurity challenges in vehicular communications*. Online. In: SCIENCEDIRECT. 2020. Dostupné z: https://doi.org/10.1016/j.vehcom.2019.100214. [cit. 2023-11-24].

[32] LIU, Jiajia; ZHANG Shubin; SUN Wen; SHI Yongpeng. *In-Vehicle Network Attacks and Countermeasures: Challenges and Future Directions*. Online. In: IEEE. 2017. Dostupné z: https://doi.org/10.1109/MNET.2017.1600257. [cit. 2023-11-19].

[33] ALIWA, Emad; RANA Omer; PERERA Charith; BURNAP Peter. *Cyberattacks and Countermeasures for In-Vehicle Networks*. Online. In: ACM DIGITAL LIBRARY. 2021. Dostupné z: https://doi.org/10.1145/3431233. [cit. 2023-11-20].

[34] KHATRI, Narayan; SHRESTHA Rakesh; NAM Seung Yeob. *Security Issues with In-Vehicle Networks, and Enhanced Countermeasures Based on Blockchain*. Online. In: MDPI. 2021. Dostupné z: https://doi.org/10.3390/electronics10080893. [cit. 2023-11-23].

[35] LANG, Martin. *Secure Automotive Ethernet: Balancing Security and Safety in Time Sensitive Systems*. Online, disertační práce. Karlskrona, Sweden: Blekinge Institute of Technology, Faculty of Computing, Department of Computer Science, 2019. Dostupné z: https://www.diva-portal.org/smash/get/diva2:1332281/FULLTEXT01.pdf. [cit. 2023-11-28].

[36] ZORZ, Zejlka. *Researchers hack BMW cars, discover 14 vulnerabilities*. Online. In: HELP NET SECURITY. 2018. Dostupné z: https://doi.org/10.48550/arXiv.1807.09338. [cit. 2024-05-21].

[37] LAURENDEAU, Christine; BARBEAU, Michel. *Threats to Security in DSRC/WAVE*. Online. In: SPRINGERLINK. 2006. Dostupné z: https://link.springer.com/chapter/10.1007/11814764_22. [cit. 2023-12-09].

[38] MAROJEVIC, Vuk. *C-V2X Security Requirements and Procedures: Survey and Research Directions*. Online. In: ARXIV. 2018. Dostupné z: https://doi.org/10.48550/arXiv.1807.09338. [cit. 2023-12-09].

[39] NEUMEIER, Stefan; GAY Nicolas; DANNHEIM, Clemens; FACCHI, Christian. *On the Way to Autonomous Vehicles Teleoperated Driving*. Online. In: IEEE. 2018. Dostupné z: https://ieeexplore.ieee.org/document/8402813/authors#authors. [cit. 2023-12-10].

[40] TENER, Felix; LANIR, Joel. *Driving from a Distance: Challenges and Guidelines for Autonomous Vehicle Teleoperation Interfaces*. Online. In: ACM DIGITAL LIBRARY. 2022. Dostupné z: https://doi.org/10.1145/3491102.3501827. [cit. 2023-12-10].

[41] GNATZIG, Sebastian; CHUCHOLOWSKI, E. Frederic; TANG, Tito; LIENKAMP, Markus. *A System Design for Teleoperated Road Vehicles*. Online. In: RESEARCHGATE. 2013. Dostupné z: https://www.researchgate.net/publication/256946185_A_System_Design_for_Teleoperated_Road_Vehicles. [cit. 2024-03-14].

[42] GORMAN, Ben. *TCP vs UDP: What's the Difference and Which Protocol Is Better?* Online. IN: AVAST. 2023. Dostupné z: https://www.avast.com/c-tcp-vs-udp-difference. [cit. 2024-03-14]

[43] YASAR, Kinza. *Transmission Control Protocol (TCP)*. Online. In: TECHTARGET. 2023. Dostupné z: https://www.techtarget.com/searchnetworking/definition/TCP. [cit. 2024-03-14].

[44] ASHTARI, Hossein. *What Is User Datagram Protocol (UDP)? Definition, Working, Applications, and Best Practices for 2022*. Online. In: SPICEWORKS. 2022. Dostupné z: https://www.spiceworks.com/tech/networking/articles/user-datagram-protocol-udp/. [cit. 2024-03-14].

[45] Craggs, Ian. *MQTT Vs CoAP for IoT*. Online. In: HiveMQ. 2024 Dostupné z: https://www.hivemq.com/blog/mqtt-vs-coap-for-iot/. [cit. 2024-04-03].

[46] MAAYAN, Gilad David. *Applying MQTT for the Internet of Vehicles*. Online. In: EMBEDDED. 2023. Dostupné z: https://www.embedded.com/applying-mqtt-for-the-internet-of-vehicles/. [cit. 2024-04-03].

[47] EMQ TECHNOLOGIES INC. *Driving the Future of Connected Cars with MQTT*. Online. ©2023. Dostupné z: https://www.autocar.co.uk/sites/autocar.co.uk/files/wp1_driving-the-future-of-connected-cars-with-mqtt.pdf/. [cit. 2024-04-03].

[48] REAL-TIME INNOVATIONS. *DDS Standard*. Online. ©2024. Dostupné z: https://www.rti.com/products/dds-standard. [cit. 2024-04-04].

[49] REAL-TIME INNOVATIONS. *DDS in Autonomous Car Design*. PDF. Online. ©2016. Dostupné z: https://info.rti.com/hubfs/whitepapers/DDS_in_Autonomous_Car_Design.pdf. [cit. 2024-04-04].

[50] EMQ TECHNOLOGIES INC. *CoAP Protocol: Key Features, Use Cases, and Pros/Cons*. Online. 2023. Dostupné z: https://www.emqx.com/en/blog/coap-protocol. [cit. 2024-04-04].

[51] INTERNET SOCIETY. *TLS Basics*. Online. ©2023. Dostupné z: https://www.internetsociety.org/deploy360/tls/basics/. [cit. 2024-04-14].

[52] ZELLE, Daniel; KRAUß, Christoph; STRAUß, Hubert; SCHMIDT, Karsten. *On Using TLS to Secure In-Vehicle Networks*. Online. In: ACM DIGITAL LIBRARY. 2017. Dostupné z: https://dl.acm.org/doi/10.1145/3098954.3105824. [cit. 2024-04-14].

[53] HACKCONTROL. *What is DTLS and how is it used?* Online. ©2024. Dostupné z: https://hackcontrol.org/blog/what-is-dtls-and-how-is-it-used/. [cit. 2024-04-15].

[54] CDNETWORKS INC. *What is QUIC? How Does It Boost HTTP/3?*. Online. ©2024. Dostupné z: https://www.cdnetworks.com/media-delivery-blog/what-is-quic/. [cit. 2024-04-15].

[55] WANG, Fan. *What Is the QUIC Protocol?* Online. In: EMQ TECHNOLOGIES INC. 2023. Dostupné z: https://www.emqx.com/en/blog/quic-protocol-the-features-use-cases-and-impact-for-iot-iov#mqtt-over-quic-use-cases-in-iov. [cit. 2024-04-15].

[56] JOYE. *Top 3 Open Source MQTT Brokers for Industrial IoT in 2023*. Online. In: EMQ TECHNOLOGIES INC. 2023. Dostupné z: https://www.emqx.com/en/blog/top-3-open-source-mqtt-brokers-for-industrial-iot-in-2023#1-emqx. [cit. 2024-04-17].

[57] YIN, Chongyuan. *MQTT with Java: A Beginner's Guide with Examples & FAQs*. Online. In: EMQ TECHNOLOGIES INC. 2024. Dostupné z: https://www.emqx.com/en/blog/how-to-use-mqtt-in-java. [cit. 2024-04-24].

[58] ECLIPSE MOSQUITTO. *mosquitto.conf man page*. Online. Dostupné z: https://mosquitto.org/man/mosquitto-conf-5.html. [cit. 2024-04-24].

[59] HIVEMQ *What is MQTT Quality of Service (QoS) 0,1, & 2? – MQTT Essentials: Part 6*. Online. ©2024. Dostupné z: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/. [cit. 2024-05-14].

[60] CDP TECHNOLOGIES. *TLS Certificate Setup*. Online. ©2024. Dostupné z: https://cdpstudio.com/manual/cdp/mqttio/mqttio-certificate-setup.html. [cit. 2024-04-24].

[61] WICKRAMASINGHE, Shanika.*RSA Algorithm in Cryptography: Rivest Shamir Adleman Explained*. Online. In: SPLUNK INC. 2023. Dostupné z: https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html. [cit. 2024-05-14].

[62] ENCRYPTION CONSULTING LLC. *What is ECDSA Encryption? How does it work?*. Online. ©2024. Dostupné z: https://www.encryptionconsulting.com/education-center/what-is-ecdsa/. [cit. 2024-05-14].

[63] BRUNO, J. Eric. *Introducing the Constrained Application Protocol (CoAP) for Java*. Online. In: ORACLE. 2024. Dostupné z: https://blogs.oracle.com/javamagazine/post/java-coap-constrained-application-protocol-introduction. [cit. 2024-04-24].

Seznam symbolů a zkratek

5G	Bezdrátové sítě páté generace
ABS	Anti-lock Braking System
AES	Advanced Encryption Standard
ARS	Advanced Restraint System
BTS	Base transceiver station
CA	Certifikační autorita
CAN	Controller Area Network
CD	Compact Disk
CN	Common name
CoAP	Constrained Application Protocol
DDS	Data Distribution Service
DNS	Domain Name System
DoS	Denial of Service
DSA	Digital Signature Algorithm
DSRC	Dedicated Short Range Communication
DTLS	Datagram Transport Layer Security
ECDH	Elliptic-curve Diffie–Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECU	Electronic Control Unit
EPL	Eclipse Public License
FM	Frequency Modulation
GCM	Galois/Counter Mode
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol

HTTPS	Hypertext Transfer Protocol Secure
ID	IDentification
IDS	Intrusion Detection System
IMAP	Internet Message Access Protocol
IoT	Internet of Things
IP	Internet Protocol
ITS	Intelligent Transportation System
IVI	In-Vehicle Infotainment
IZS	Integrovaný Záchranný Systém
JKS	Java KeyStore
LiDAR	Light Detection and Ranging
M2M	Machine-to-Machine
MAC	Message Authentication Code
MITM	Man in the Midle
MOST	Media Oriented Systems Transport
MQTT	Message Queuing Telemetry Transport
NHTSA	National Highway Traffic Safety Administration
OBD	On-Board Diagnostics
OBU	On-Board Unit
PEM	Privacy-enhanced Electronic Mail
PKCS	Public-Key Cryptography Standards
PSK	Pre-Shared Keye
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RSA	Rivest, Shamir, Adleman

RSU	Roadside Unit
SAE	Society of Automotive Engineers
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protoco
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
USA	United States of America
USB	Universal Serial Bus
UTP	Unshielded twisted pair
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Networ
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
VRUs	Vulnerable Road Users
WiFi	Wireless Fidelity

Seznam příloh

A Program teleoperačního centra

Obsahem první části elektronické přílohy je struktura vytvořeného programu v jazyce Java, který simuluje teleoperační centrum.

Složka *lib* obsahuje jar soubory knihoven, které implementují jednotlivé protokoly včetně jejich zabezpečení a jsou potřebné pro správné fungování programu teleopreačního centra. Složka *src* sdružuje třídy, ve kterých jsou provedeny jednotlivé implementace protokolů. Dále třídu pro vytvoření logování a autentizaci uživatele. Tyto třídy spouští spustitelná třída Main.java. Program dále obsahuje binární soubory různých velikostí, které lze posílat klientovi, potřebné certifikáty a klíče pro autentizaci v TLS a DLTS spojení.

```
TeleoperateServer ..... Program teleoperační centrum
├── lib ..... Použité knihovny
│   ├── californium-core-3.11.0.jar
│   ├── californium-legal-3.11.0.jar
│   ├── eddsa-0.3.0.jar
│   ├── element-connector-3.11.0.jar
│   ├── org.eclipse.paho.mqttv5.client-1.2.5.jar
│   ├── scandium-3.11.0.jar
│   └── slf4j-api-2.0.12.jar
├── src ..... Zdrojový kód programu
│   ├── Logger.java
│   ├── Main.java
│   ├── MqttPublisherNotSecure.java
│   ├── MqttPublisherTls.java
│   ├── ServerCoapNotSecure.java
│   ├── ServerCoapSecureDtls.java
│   └── UserAuthenticator.java
├── CA_ECDSA.crt
├── Californium3.properties
├── CoAP_server_ECDSA.p12
├── CoAP_server_RSA.p12
├── binary_100B.bin
├── binary_100kB.bin
├── binary_10B.bin
├── binary_10MB.bin
├── binary_10kB.bin
├── binary_1B.bin
├── binary_1MB.bin
├── binary_1kB.bin
├── client_pub_ECDSA.p12
├── client_pub_RSA.p12
├── localhostCA.crt
└── passwd.txt
```

B Program autonomního vozidla

Druhá část elektronické přílohy poskytuje strukturu vytvořeného programu v jazyce Java, který naopak simuluje autonomní vozidlo.

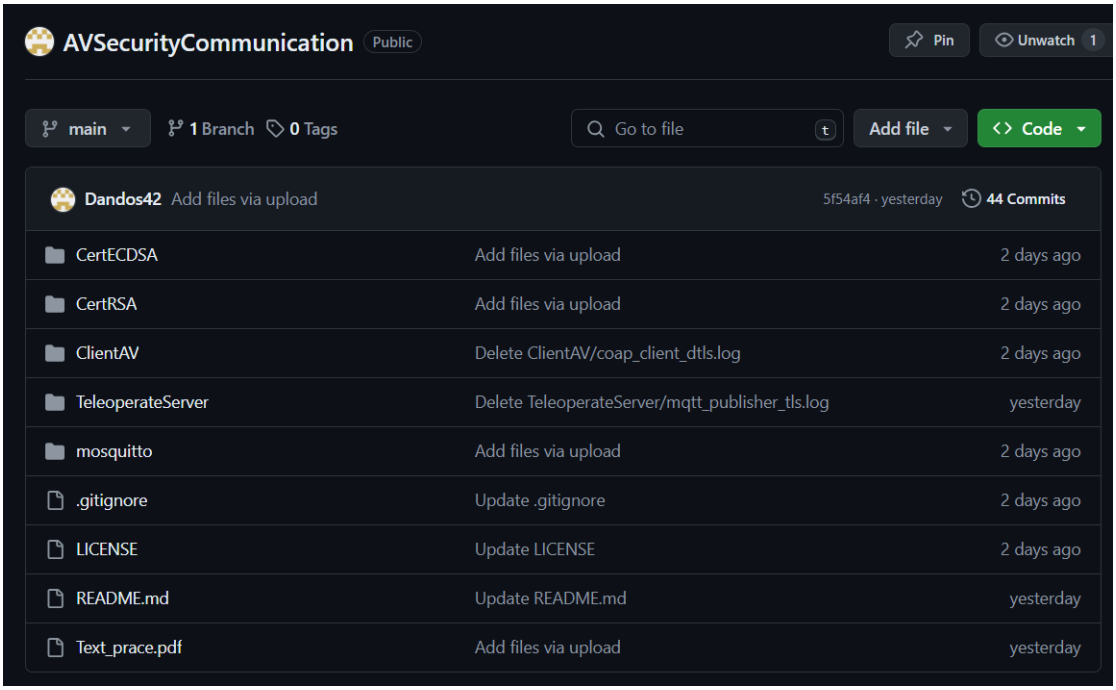
Složka *lib* opět obsahuje jar soubory knihoven, jež implementují jednotlivé protokoly. Složka *src* zahrnuje třídy, ve kterých jsou provedeny jednotlivé implementace protokolů. Rovněž třídu pro vytvoření logování a autentizaci uživatele. Navíc obsahuje třídu FileSaver pro uložení přijatých binárních souborů a rozhraní Override. Tyto třídy jsou spouštěny třídou Main.java. V programu najdeme také, potřebné certifikáty a klíče pro autentizaci v TLS a DLTS spojení. Poslední položkou je soubor passwd.txt, ten obsahuje přihlašovací údaje pro autentizaci uživatele.

```
ClientAV.....................................Program atonomního vozidla
├── lib.....................................Použité knihovny
│   ├── californium-core-3.11.0.jar
│   ├── californium-legal-3.11.0.jar
│   ├── eddsa-0.3.0.jar
│   ├── element-connector-3.11.0.jar
│   ├── org.eclipse.paho.mqttv5.client-1.2.5.jar
│   ├── scandium-3.11.0.jar
│   └── slf4j-api-2.0.12.jar
├── src.....................................Zdrojový kód programu
│   ├── ClientCoapDtls.java
│   ├── ClientCoapNotSecure.java
│   ├── FileSaver.java
│   ├── Logger.java
│   ├── Main.java
│   ├── MqttSubscriberNotSecure.java
│   ├── MqttSubscriberTls.java
│   ├── Override.java
│   └── UserAuthenticator.java
├── CA_ECDSA.crt
├── Californium3.properties
├── CoAP_client_ECDSA.p12
├── CoAP_client_RSA.p12
├── client_sub_ECDSA.p12
├── client_sub_RSA.p12
├── localhostCA.crt
└── passwd.txt
```

C Repozitář GitHub

Kvůli omezení velikosti příloh nebylo možné aplikaci odevzdat obvyklým způsobem. Proto byl vytvořen repozitář, který je dostupný na následující adrese: **`https://github.com/Dandos42/AVSecurityCommunication`**.

Repozitář obsahuje program teleoperačního centra, autonomního vozidla a jejich potřebné části pro spuštění aplikace, certifikáty a klíče certifikačních autorit, serveru, brokera a klientů. Je zde uložen i konfigurační soubor brokera Mosquitto včetně jeho certifikátů. V repozitáři také najdeme licenci, odkazy na použité knihovny a soubor README.MD, který obsahuje veškeré potřebné informace ohledně aplikace jejího použití a instalace. Obrázek C.1 zobrazuje výstřižek repozitáře GitHub, kde je k dispozici výsledná aplikace.



Obr. C.1: Prostředí GitHub s výslednou aplikací