

**Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky**

**Technologie WebRTC a její podpora v open-source IP
telefonních řešeních**

**WebRTC Technology and Its Support in Open-source IP
Telephony Solutions**

2019

Bc. Jan Dvořáček

Zadání diplomové práce

Student: **Bc. Jan Dvořáček**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1801T064 Informační a komunikační bezpečnost

Téma: **Technologie WebRTC a její podpora v open-source IP telefonních řešeních.**
WebRTC Technology and Its Support in Open-source IP Telephony Solutions.

Jazyk vypracování: čeština

Zásady pro vypracování:

Technologie WebRTC se v současnosti stala trendem v IP telefonním světě. Komunikace pomocí WebSocketů je již otestovaná a plně funkční v komerčních aplikacích, avšak open-source produkty ještě vyžadují často optimalizaci konfigurace a přídavné moduly pro správnou výměnu komunikace. Cílem diplomové práce je navrhnout, nakonfigurovat, otestovat a zdokumentovat možnosti komunikace s využitím WebRTC technologie na dostupných otevřených IP telefonních řešeních a to především z pohledu zabezpečeného provozu.

Body zadání:

1. Detailně nastudujte technologii WebRTC a komunikaci s využitím WebSocketů a zabezpečených WebSocketů.
2. Navrhněte vhodné open-source IP telefonní řešení, na kterých bude probíhat konfigurace a testování WebRTC provozu.
3. Realizujte instalaci a konfiguraci WebRTC provozu na vybraných IP telefonních řešeních se zaměřením na zabezpečení.
4. Otestujte funkčnost řešení a analyzujte zachycený provoz komunikace.
5. Vytvořte podrobnou dokumentaci ke konfiguraci a postupu při realizaci na jednotlivých IP telefonních řešeních.

Seznam doporučené odborné literatury:

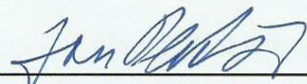
- [1] Getting Started with WebRTC, Rob Manson , 2013, ISBN-10: 1782166300.
- [2] Handbook of SDP for Multimedia Session Negotiations: SIP and WebRTC IP Telephony, Radhika Ranjan Roy , 2018, ISBN-10: 9781138484498.
- [3] Real-Time Communication with WebRTC: Peer-to-Peer in the Browser, Salvatore Loreto and Simon Pietro Romano, 2014, ISBN-10: 1449371876.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

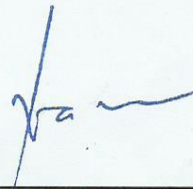
Vedoucí diplomové práce: **Ing. Filip Řezáč, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry



prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: 29. dubna 2019



.....
podpis studenta

Poděkování

Rád bych poděkoval Ing. Filipu Řezáčovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Tato diplomová práce se zabývá technologií WebRTC a její podporou v open-source IP telefonních řešeních. WebRTC je v současnosti trendem v IP telefonním světě. Tato technologie je již plně funkční v komerčních aplikacích, avšak open-source produkty vyžadují často optimalizaci. Výstupem této práce je podrobná dokumentace konfigurace technologie WebRTC na open-source produktech a následná analýza spojení pomocí paketového analyzátoru Wireshark. Účelem této práce je zrealizovat a zdokumentovat konfiguraci WebRTC v nejvíce rozšířených open-source IP telefonních řešeních.

Klíčová slova

WebRTC; Asterisk; FreeSWITCH; PBX; Websocket; Security; Sipml5

Abstract

This thesis deals with WebRTC technology and its support in open-source IP telephony solutions. WebRTC is currently a trend in the IP telephony world. This technology is already fully operational in commercial applications, but open-source products often require optimization. The output of this work is detailed documentation of WebRTC configuration on open-source products and subsequent analysis of the connection using Wireshark packet analyzer. The purpose of this work is to implement and document WebRTC configuration in the most widespread open-source IP telephony solutions

Key words

WebRTC; Asterisk; FreeSWITCH; PBX; WebSocket; Security; Sipml5

Seznam použitých zkratek

Zkratka	Význam
API	Application Programming Interface
DH	Diffie-Hellman
DTLS	Datagram Transport Layer Security
ECDH	Elliptic-Curve Diffie – Hellman
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICE	Interactive Connectivity Establishment
IP	Internet Protocol
ISO/OSI	International Standards Organization Open Systems Interconnection
LTS	Long Term Support
NAT	Network Address Translation
P2P	Peer to Peer
PBX	Private branch exchange
PC	Personal Computer
RFC	Request for Comments
RTP	Real-Time Protocol
SCTP	Stream Control Transmission Protocol
SIP	Session Initiation Protocol
SRTP	Secure Real-Time Transport Protocol
SSL	Secure Sockets Layer
STUN	Session Traversal Utilities for Nat
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TURN	Traversal Using Relay NAT
UA	User Agent
UDP	User Datagram Protocol
URL	Uniform Resource Locator

VoIP	Voice over Internet Protocol
WebRTC	Web Real-Time Communications
WS	WebSocket
WSS	Secure WebSocket
XHR	XML Http Request
XML	Extensible Markup Language

Seznam obrázků

Obrázek 1.1:	MediaStream API.....	- 18 -
Obrázek 1.2:	WebRTC protokoly a služby 1.....	- 19 -
Obrázek 1.3:	NAT - ukázkové schéma.....	- 21 -
Obrázek 1.4:	Schéma komunikace STUN serveru.....	- 21 -
Obrázek 1.5:	Schéma komunikace TURN serveru.....	- 22 -
Obrázek 1.6:	DTLS-SRTP handshake.....	- 24 -
Obrázek 1.7:	Klientská DTLS-SRTP relace.....	- 25 -
Obrázek 1.8:	Identifikátor využití mikrofonu.....	- 26 -
Obrázek 1.9:	Signalizace v ideálních podmínkách.....	- 27 -
Obrázek 1.10:	Schéma navazování spojení 1.....	- 27 -
Obrázek 1.11:	Signalizace v ideálních podmínkách.....	- 28 -
Obrázek 1.12:	Schéma navazování spojení 2.....	- 28 -
Obrázek 1.13:	Signalizace v reálných podmínkách STUN.....	- 29 -
Obrázek 1.14:	Signalizace v reálných podmínkách TURN.....	- 29 -
Obrázek 1.15:	Příklad INVITE zprávy.....	- 31 -
Obrázek 1.16:	Příklad odpovědi typu 200 - OK.....	- 32 -
Obrázek 1.17:	Struktura WebSocket rámce.....	- 35 -
Obrázek 1.18:	Registrace uživatele využitím WebSocketu.....	- 38 -
Obrázek 4.1:	Logická topologie.....	- 42 -
Obrázek 4.3:	Informace o registraci klienta.....	- 47 -
Obrázek 4.4:	Průběh hovoru v PBX Asterisk.....	- 48 -
Obrázek 4.5:	Šifrovaná SIP komunikace.....	- 49 -
Obrázek 4.6:	Tělo TCP packetu.....	- 49 -
Obrázek 4.7:	Import klíče pro dešifrování komunikace.....	- 50 -
Obrázek 4.8:	Vložení klíče.....	- 50 -
Obrázek 4.9:	Dešifrovaná komunikace.....	- 51 -
Obrázek 4.10:	Zpráva HTTP.....	- 51 -
Obrázek 4.11:	Výpis registrovaných uživatelů.....	- 53 -
Obrázek 4.12:	Nastavení parametrů hovoru.....	- 54 -
Obrázek 4.13:	SIP flow.....	- 55 -

Seznam tabulek

<i>Tabulka 1.1:</i>	<i>Podporované prohlížeče.....</i>	<i>- 16 -</i>
<i>Tabulka 1.2:</i>	<i>Podporované platformy.....</i>	<i>- 16 -</i>
<i>Tabulka 2.1:</i>	<i>FreeSWITCH a podpora WebRTC.....</i>	<i>- 40 -</i>
<i>Tabulka 3.1:</i>	<i>Asterisk a podpora WebRTC.....</i>	<i>- 41 -</i>

Obsah

Seznam obrázků.....	- 10 -
Seznam tabulek.....	- 11 -
Úvod.....	- 12 -
1 WebRTC.....	- 16 -
1.1 Architektura WebRTC API	- 17 -
1.1.1 MediaStream API.....	- 17 -
1.1.2 RTCPeerConnection API.....	- 18 -
1.1.3 RTCDataChannel	- 18 -
1.2 WebRTC - protokoly a mechanismy.....	- 19 -
1.2.1 UDP (User Datagram Protocol)	- 19 -
1.2.2 SDP (Session Description Protocol).....	- 19 -
1.2.3 SRTP (Secure Real-time Transport Protocol).....	- 20 -
1.2.4 SCTP (Stream Control Transmission Protocol)	- 20 -
1.2.5 DTLS (Datagram Transport Layer Security)	- 20 -
1.2.6 NAT (Network Address Translation)	- 20 -
1.2.7 STUN (Session Traversal Utilities for NAT).....	- 21 -
1.2.8 TURN	- 22 -
1.2.9 ICE (Interactive Connectivity Establishment)	- 22 -
1.3 WebRTC - zabezpečení.....	- 22 -
1.3.1 Ustanovení zabezpečeného spojení	- 22 -
1.3.2 Vyjednání klíčů pro SRTP	- 23 -
1.3.3 TLS.....	- 23 -
1.3.4 DTLS	- 23 -
1.3.5 DTLS-SRTP.....	- 24 -
1.3.6 SOP (Same Origin Policy)	- 25 -
1.3.7 Přístup k mikrofonu a kameře	- 26 -
1.4 WebRTC - komunikace mezi klienty.....	- 26 -
1.4.1 Komunikace v ideálním prostředí	- 26 -
1.4.2 Komunikace v reálných podmínkách	- 28 -
1.5 Možnosti přenosu signalizace ve WebRTC.....	- 30 -

1.5.1	SIP	- 30 -
1.5.2	Obsah hlavičky	- 32 -
1.5.3	Adresy	- 32 -
1.5.4	SIP - komponenty	- 32 -
1.6	Comet/XHR/SSE	- 33 -
1.7	WebSocket	- 33 -
1.7.1	Navázání spojení	- 33 -
1.7.2	WebSocket zprávy	- 35 -
1.7.3	WebSocket Secure	- 37 -
1.8	SIP over WebSocket	- 37 -
1.9	XMPP/Jingle	- 38 -
2	FreeSWITCH	- 40 -
2.1	FreeSWITCH a WebRTC	- 40 -
3	Asterisk	- 41 -
3.1	Asterisk a WebRTC	- 41 -
4	Praktická realizace	- 42 -
4.1	Konfigurace PBX Asterisk	- 42 -
4.1.1	Konfigurace vestavěného HTTP serveru	- 42 -
4.1.2	Konfigurace uživatelských účtů	- 44 -
4.1.3	Konfigurace dialplánu	- 46 -
4.1.4	Předpřípravení webového prohlížeče	- 46 -
4.1.5	Povolení výjimky pro wss transport se serverem	- 46 -
4.1.6	Analýza testovacího hovoru	- 47 -
4.1.7	Analýza komunikace	- 48 -
4.2	Konfigurace PBX FreeSWITCH	- 52 -
4.2.1	Konfigurace modulu Sofia	- 52 -
4.2.2	Konfigurace uživatelských účtů	- 52 -
4.2.3	Konfigurace základního kontextu	- 52 -
4.2.4	Testovací hovor	- 53 -
4.2.5	Analýza komunikace	- 54 -
4.3	Zhodnocení	- 55 -

Závěr	- 56 -
Seznam použitých zdrojů	- 57 -
Seznam příloh.....	- 59 -

Úvod

Komunikační služby v prostředí Internetu jsou dnes standardem, bez kterého by si většina z nás již nedokázala život představit. Téměř každý z nás používá chytré zařízení, na kterém je aplikace umožňující online komunikaci. Tato komunikace již dávno není omezená pouze na textovou formu, jak tomu bylo dříve. Dnešní aplikace zprostředkovávající online komunikaci, jako jsou např. FaceTime, WhatsApp, nebo Facebook Messenger umožňují rovněž komunikaci hlasovou, či v současné době velice populární videohovory a videokonference. Podmínkou pro využívání takovýchto služeb byla až doposud nutnost instalace klientského softwaru a pokud služba disponuje svou online verzí, bez nutnosti instalace rozšiřujících plug-inů, které jsou zapotřebí pro přenos audia z mikrofону a videa z kamery, umožňovala pouze základní textovou formu komunikace. Instalace rozšiřujících plug-inů třetích stran, představuje podstatné bezpečnostní riziko. Jelikož tyto plug-iny obvykle mývají bezpečnostní mezery a mohou způsobovat problémy se stabilitou webového prohlížeče, či operačního systému, z pohledu dnešních stále se zvyšujících požadavků na bezpečnost nepředstavují vhodné a především bezpečné řešení. Tento problém řeší projekt společnosti Google s názvem WebRTC, který nabízí vývojářům webových aplikací takové nástroje, aby byli schopni vytvořit aplikace, které dokáží odchytnit audio nebo video pouze za pomoci nástrojů, které jsou integrovány v samotném webovém prohlížeči, tedy nevyužívají rozšiřující plug-iny, nebo další rozšíření. Díky tomu má projekt WebRTC velikou výhodu, jelikož může v budoucnu nahradit veškeré aplikace, které sice umožňují online komunikaci, ale využívají mnoha rozšíření, které nejsou vždy žádoucí. Zavedením WebRTC umožníme přístup k online komunikaci všem zařízením s přístupem na internet a s nainstalovaným webovým prohlížečem, bez nutnosti cokoliv stahovat, nebo instalovat.

První kapitola této práce popisuje projekt WebRTC a příslušné protokoly a mechanismy, které WebRTC používá, také jsou zde uvedeny principy zabezpečení, na které se ve WebRTC klade veliký důraz. Ve druhé kapitole práce cílí na popis pobočkové ústředny Asterisk a její podporu WebRTC. Třetí kapitola popisuje popis pobočkové ústředny FreeSWITCH a podporu WebRTC na této ústředně. Ve čtvrté kapitole se práce zabývá praktickou realizací WebRTC na open-source IP telefonních řešeních a analýzou síťové komunikace z pohledu jejího zabezpečení.

WebRTC má velký potenciál a má celou řadu výhod. Díky tomu můžeme očekávat, že se projekt WebRTC bude v budoucnu rozšiřovat a bude stále populárnější. Proto přibudou další webové aplikace, které na něm budou založené.

Víra v budoucnost a potenciál projektu WebRTC společně s osobním zájmem mě vedly k výběru práce na toto téma.

1 WebRTC

WebRTC neboli Web Real-Time Communications je projekt společnosti Google, která v roce 2011 zveřejnila jeho zdrojové kódy. Podstatou WebRTC je umožnění multimediální komunikace v reálném čase v prostředí webových prohlížečů. Multimediální komunikaci může představovat přenos videa, dat, nebo audia. WebRTC není prvním projektem, který se touto možností komunikace zabývá, rozdíl mezi WebRTC a ostatními projekty spočívá v tom, že jako první umožňuje komunikaci v reálném čase bez nutnosti instalace rozšiřujících plug-inů, či použití softwaru třetích stran. Pro tuto funkcionalitu využívá WebRTC své vlastní API (Application Programming Interface) rozhraní, které přistupuje k jazyku HTML5 a Javascript kódu. Využitím tohoto API se řeší problémy kompatibility, jelikož koncoví uživatelé se již nemusí starat o kompatibilitu se softwarem třetích stran, ani o instalaci rozšiřujících balíčků. Vše co koncový uživatel potřebuje, je mít nainstalovaný jeden z podporovaných prohlížečů.

Projekt samotný se již od začátku těšil velkému zájmu a stal se vizí toho, jak by měla komunikace v reálném čase v prostředí webových prohlížečů vypadat.

V počátcích vývoje byl projekt vyvíjen přednostně pro prostředí prohlížečů Mozilla Firefox a Google Chrome, postupným rozvojem implementace projektu se podpora ostatních platforem a prohlížečů rozšířila a dnes můžeme prohlásit, že je tento projekt multiplatformní [13]. Aktuální seznam podporovaných prohlížečů je uveden v tabulce 1.1 a seznam podporovaných platforem v tabulce 1.2.

Tabulka 1.1: *Podporované prohlížeče*

Podporované prohlížeče	
Název Prohlížeče	Podporovaná verze
Microsoft Edge	12+
Google Chrome	28+
Mozilla Firefox	22+
Opera	18+
Vivaldi	1,9+
Safari	11+

Tabulka 1.2: *Podporované platformy*

Podporované platformy				
Android	iOS 11	Chrome OS	Firefox OS	Blackberry 10

1.1 Architektura WebRTC API

Poskytnutí vysoce kvalitních aplikací, jako jsou například audio a video konference, nebo peer-to-peer komunikace, vyžaduje ve webovém prohlížeči mnoho nových funkcí jako např. zpracování audia a videa, nebo podporu nových síťových protokolů. Pro tyto potřeby webový prohlížeč přistupuje k již dříve zmíněnému WebRTC API, které implementuje tři základní API.

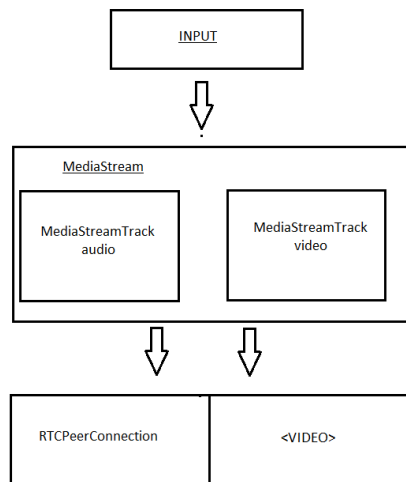
- MediaStream API umožňuje přistupovat k lokálním vstupním zařízením jako je např. kamera, nebo mikrofón a podporuje práci s nimi. Každý objekt MediaStream může obsahovat mnoho rozdílných MediaStream Track objektů, kdy každý objekt může reprezentovat rozdílné video, či zvukovou stopu
- RTCPeerConnectionAPI je základem peer-to-peer komunikace mezi každým webovým prohlížečem podporujícím WebRTC. Toto API se stará o přenos audia a videa mezi účastníky komunikace.
- RTCDataChannelAPI toto API slouží k přenosu libovolného toku dat.

1.1.1 MediaStream API

Již řadu let bylo nutné se při pořizování videa a zvuku z počítače spoléhat na plug-iny třetích stran jako jsou např. Flash, nebo Silverlight. Vše se změnilo s nástupem éry HTML5, která přinesla přímý přístup k hardwaru a hardwarovým zařízením. MediaStream API je jedno z Javascriptových API, které se používá k přístupu k datovým tokům z hardwaru jako je mikrofón, nebo kamera [12].

Základním objektem MediaStream API je objekt MediaStream, který představuje synchronizovaný tok mediálních dat. Každý tento datový tok, se může skládat z několika instancí typu MediaStreamTrack, kdy každá instance tohoto typu obsahuje vzájemně synchronizované zvukové a video stopy (viz obrázek č.1.1).

Vstupem objektu MediaStream, může být lokální zařízení, jako je např. kamera, nebo příchozí datový tok jiného účastníka komunikace. Výstupem objektu pak může být lokální zařízení schopné přehrát zvukovou stopu, video, nebo dalším typem výstupu může být objekt MediaStream spojení s jiným uživatelem.



Obrázek 1.1: *MediaStream API*

1.1.2 RTCPeerConnection API

RTCPeerConnection rozhraní se stará o koordinaci výměn zásadních metadat mezi webovými prohlížeči, tato metadata určují, jaká je veřejná IP adresa a číslo portu prohlížeče. Tyto informace jsou zásadní proto, aby byla možná výměna multimediálních dat v reálném čase. U dvou koncových bodů WebRTC, které navazují spojení, musí být předány tři druhy informací. Informace o řízení relací určují, kdy inicializovat, ukončit a změnit komunikační relace. Síťová data přenášejí IP adresu a číslo portu každého koncového bodu tak, aby volající mohl nalézt ostatní účastníky hovoru. Media data se týkají typu kodeků a typu médiu, které mají volající společné. Aby mohlo být spojení úspěšné, musí RTCPeerConnection získat metadata pro místní mediální podmínky (rozlišení, použité kodeky) a získat síťové adresy pro ostatní uživatele aplikace. Mechanismus signalizace, který předává tato data z jednoho prohlížeče k druhému, však není v rámci RTCPeerConnection zahrnut. Signalizace ve službě WebRTC je jeden z povinných aspektů, ale neexistují žádné definované signalizační standardy, které by vyžadovaly všechny WebRTC aplikace. Existují ovšem řešení, které jsou již dostatečně vyzkoušena a doporučena [12].

1.1.3 RTCDataChannel

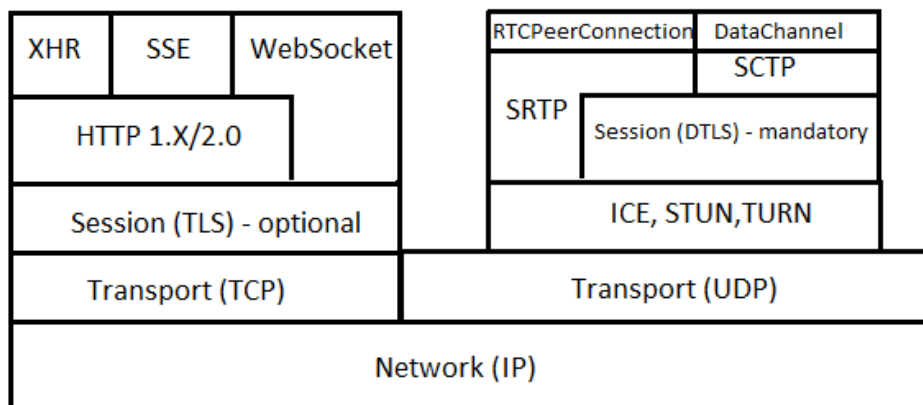
RTCDataChannel představuje hlavní obousměrný komunikační kanál, kterým dochází k výměně libovolných aplikačních dat mezi oběma účastníky spojení. Jinými slovy, slouží k přenosu dat přímo z jednoho účastníka spojení k druhému. Ačkoliv existuje řada alternativních možností pro komunikační kanály (WebSocket, Server Sent Events), tyto alternativy byly založeny na typu spojení klient-server zatímco RTCDataChannel používá spojení typu peer-to-peer [12].

1.2 WebRTC - protokoly a mechanismy

Komunikace v reálném čase je citlivá vůči negativním vlivům přenosového prostředí, převážně na zpoždění a ztrátovost. Výsledkem je, že aplikace pro streamování audia a videa jsou navrženy tak, aby tolerovaly přerušovanou ztrátu paketů. Podobně musí aplikace implementovat vlastní logiku, aby se zotavila ze ztracených nebo zpožděných paketů nesoucích jiné typy aplikačních dat. Včasnost a nízká latence mohou být důležitější než spolehlivost.

Požadavek na včasnost nad spolehlivostí je hlavním důvodem, proč je protokol UDP preferovaným transportním protokolem pro doručování dat v reálném čase. Avšak kromě transportního protokolu jsou také zapotřebí mechanismy potřebné k překonání mnoha vrstev NAT a firewallů, vyjednávání parametrů pro každý datový tok, poskytování šifrování uživatelských dat, implementace přetížení a řízení toku atd.

Pro splnění všech požadavků potřebuje prohlížeč velké podpurné obsazení protokolů a služeb nad ním (viz obrázek č.1.2) [12][1].



Obrázek 1.2: *WebRTC protokoly a služby*

1.2.1 UDP (User Datagram Protocol)

Protokol UDP je transportním protokolem 4. vrstvy referenčního modelu ISO/OSI. Jelikož protokol UDP nezaručuje úspěšné doručení zpráv, nezasílá potvrzení o přijetí, nezavádí číslování datagramů, nesleduje stav připojení a nestará se o opakované vysílání, je označován jako nespolehlivý způsob pro doručení dat k cíli. Díky své jednoduchosti je tento protokol vhodný pro použití v aplikacích, kde se počítá se ztrátovostí datagramů a kde není potřebné a vhodné ztrácet čas s opětovným odesláním.

1.2.2 SDP (Session Description Protocol)

SDP je internetový protokol určený k popisu vlastností relace multimediálního přenosu dat. Nepřenáší se pomocí něj vlastní data, ale slouží pro vyjednávání parametrů, jako je typ média (audio, video atd.), transportní protokol (RTP/UDP/IP, atd.), typ kodeku, přenosová rychlost.

1.2.3 SRTP (Secure Real-time Transport Protocol)

Základní protokol RTP nemá žádné zabudované bezpečnostní mechanismy, a proto neposkytuje ochranu přenášených dat. Externí mechanismy se místo toho spoléhají na šifrování. Ve skutečnosti je použití nešifrovaného protokolu RTP výslovně zakázáno specifikací WebRTC. Pro šifrování multimediálních toků používá WebRTC protokol SRTP, nikoliv DTLS. Důvodem je to, že SRTP představuje menší zátěž než DTLS. Výměna klíčů SRTP se však zpočátku provádí pomocí protokolu DTLS-SRTP, což umožňuje detekci jakýchkoli útoků typu MiTM.

1.2.4 SCTP (Stream Control Transmission Protocol)

Stejně jako UDP nebo TCP se SCTP nachází na transportní vrstvě referenčního modelu ISO/OSI. Od stávajících transportních protokolů se SCTP liší schopností přenášet několik navzájem nezávislých kanálů paralelně. Po navázání spojení, kterému se říká asociace lze přenášet několik navzájem nezávislých streamů. V rámci každého z nich dokáže SCTP garantovat doručení všech dat ve správném pořadí. Případný výpadek (a pozdější opakování, čili zdržení) v některém z proudů se však nijak netýká proudů ostatních. Jejich komunikace pokračuje bez přerušení.

1.2.5 DTLS (Datagram Transport Layer Security)

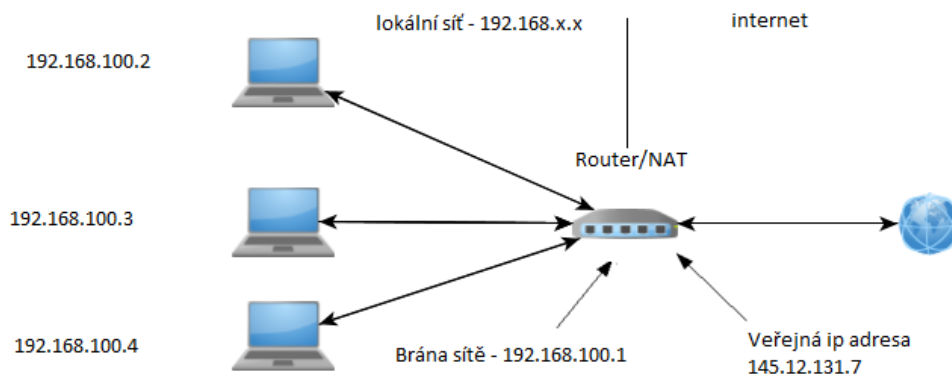
DTLS je standardizovaný protokol, který je integrován do všech prohlížečů, které podporují technologii WebRTC a je jedním z protokolů, který se důsledně používá ve webových prohlížečích, e-mailu a VoIP platformách pro šifrování informací. Tato integrace také znamená, že před použitím není požadováno žádné předchozí nastavení. Stejně jako u jiných protokolů šifrování je DTLS navržen tak, aby zabránil odposlechu a manipulaci s informacemi. Samotný DTLS je modelován na streamově orientovaném TLS, což je protokol, který nabízí plné šifrování metodami asymetrické kryptografie, ověření dat a autentizace zpráv. TLS je de facto standardem pro šifrování webu, který se používá pro účely protokolů, jako je protokol HTTPS. TLS je určen pro spolehlivý mechanismus přenosu TCP, ale aplikace pro technologie VoIP (hry apod.) typicky využívají nespolehlivé typy přenosů datagramů, jako je UDP.

Vzhledem k tomu, že služba DTLS je odvozená od SSL, je známo, že všechna data jsou stejně bezpečná jako při použití jakéhokoli standardního připojení založeného na protokolu SSL.

1.2.6 NAT (Network Address Translation)

V prostředí internetu, je každé zařízení reprezentováno unikátní IP adresou. Takových adres ale existuje pouze omezené množství. To bylo jedním z důvodů vzniku systému NAT, díky kterému mohou sítě s velkým počtem počítačů vystupovat pod jedinou unikátní veřejnou IP adresou. Při komunikaci směrem do internetu, zařízení s NAT přeloží skupiny vnitřních neveřejných adres, které se nachází v lokální síti, na adresu vnější a do internetu tak zařízení vstupují s veřejnou unikátní IP adresou. NAT tak v podstatě představuje hranici mezi internetem a LAN sítí [6].

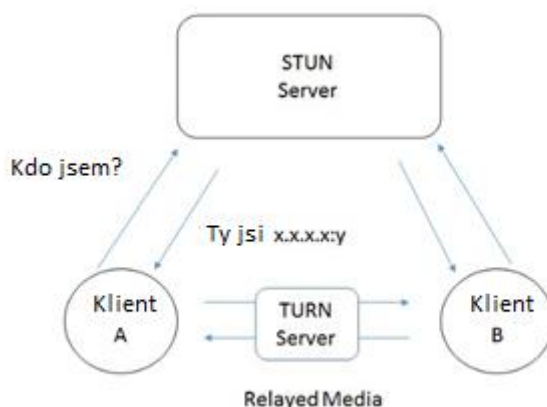
Na obrázku č.1.3 vidíme tři počítače v lokální síti 192.168.100.0/24. Každý z PC má svojí unikátní lokální IP adresu. Pokud však tyto počítače chtějí komunikovat se zařízením, které leží mimo jejich lokální síť, pošlou požadavek na svou výchozí bránu, což je router s funkcí NAT, který má adresu 192.168.1.1. Tento router přepisuje jejich lokální IP adresy na veřejnou adresu 145.12.131.7.



Obrázek 1.3: NAT - ukázkové schéma

1.2.7 STUN (Session Traversal Utilities for NAT)

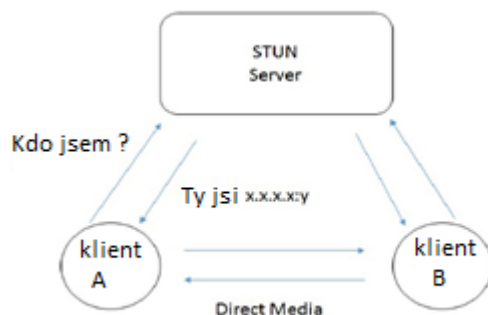
Aby mohla být úspěšně vykonána komunikace typu P2P, obě strany nutně vyžadují alespoň znalost IP adresy svého peeru a přiděleného UDP portu. V důsledku toho je nutné, aby byla před navázáním WebRTC komunikace, nezbytná určitá výměna informací. Server STUN je používán každým peerem k určení jeho vlastní veřejné IP adresy, veřejného portu přiřazeného k překladači NAT s určitým lokálním portem a typu překladu NAT.



Obrázek 1.4: Schéma komunikace STUN serveru

1.2.8 TURN

TURN server je jakýmsi rozšířením STUN serveru, který umožňuje průchod médií skrze symetrický NAT. Na rozdíl od STUN serveru, TURN server zůstává mezi klienty i po sestavení spojení. V komunikaci funguje jako media proxy server.



Obrázek 1.5: Schéma komunikace TURN serveru

1.2.9 ICE (Interactive Connectivity Establishment)

ICE je framework používaný pro vytváření spojení mezi uživateli přes Internet. Jeho funkce spočívá v nalezení nejvhodnějších STUN a TURN serverů. Existuje mnoho důvodů, proč přímé P2P spojení mezi uživateli jednoduše nebude fungovat. Je třeba obejít brány firewall a NAT servery, k tomu slouží již výše zmiňované STUN a TURN servery. K jejich vyhledání slouží právě ICE. Můžeme tedy říct, že ICE je technika, která slouží WebRTC uživatelům překonat reálné síťové prostředí.

1.3 WebRTC - zabezpečení

Mezi hlavní problémy real-time komunikace patří možnost odposlouchávání komunikace třetí stranou. V případě nešifrované komunikace mezi webovými prohlížeči a serverem, se přenášená data mohou stát cílem odposlouchávání. V případě, že jsou data šifrovaná, stávají se pro třetí stranu nečitelná, pokud ovšem zmíněná třetí nevlastní šifrovací klíč, který slouží k jejich dešifrování. Ve WebRTC je zabezpečené spojení a šifrování povinným aspektem, aplikace se serverem vždy musí komunikovat přes zabezpečené spojení a média. Data mohou být také šifrována, díky tomu můžeme WebRTC spojení považovat za bezpečné. WebRTC používá pro šifrování médií a vyjednání výměny klíčů protokoly DTLS a SRTP. DTLS se používá pro šifrování datových toků odesílaných přes RTCDataChannel a SRTP pro šifrování audio a video toků [1].

1.3.1 Ustanovení zabezpečeného spojení

Mezi dvěma komunikujícími stranami jsou prvně vyměněna signalizační data (viz podkapitola 1.5), je provedena kontrola ICE kandidátů a následně obě strany začnou

ustanovovat zabezpečený kanál (nebo i více kanálů). Na všech kanálech ustanovených přes ICE je prvně proveden DTLS handshake. Protokol DTLS slouží k výměně klíčů pro protokol SRTP, který se poté používá pro šifrování RTP relace [1].

1.3.2 Vyjednání klíčů pro SRTP

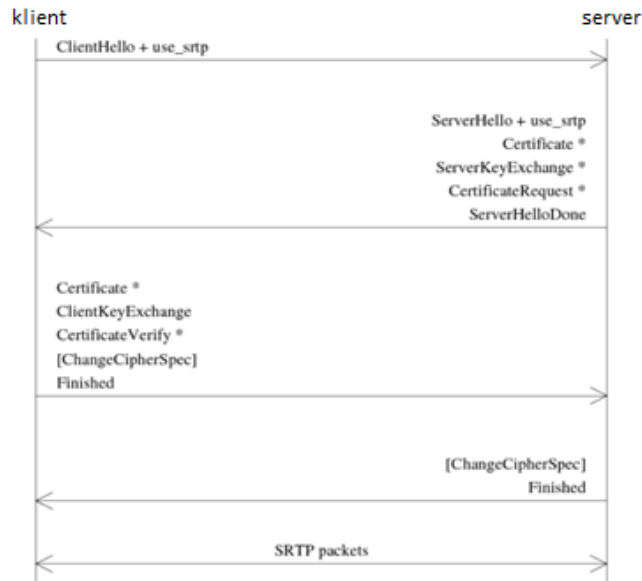
Jednou z metod pro vyjednání klíčů pro SRTP je Security Descriptions for Media Streams (SDS), který k přenosu klíčů a parametrů zabezpečení využívá protokolu SDP. Materiál zabezpečení je tedy v tomto případě přenášén v rámci signalizace. Dalším podstatným faktem je, že z využití SDP také vyplývá, že jsou šifrovací klíče pro média vyjednány v textové podobě a je tedy potřeba využít protokol pro zašifrování signalizačního kanálu. Hlavním problémem SDS však zůstává fakt, že jsou signalizace a média šifrovány zvlášť. To může potenciálně vést k situaci, ve které mají signalizace a média různého uživatele. Jelikož však SDS neposkytuje žádné prostředky pro zajištění stejného uživatele pro signalizaci i média, byla tato metoda pro vyjednání klíčů nahrazena protokolem DTLS-SRTP. WebRTC specifikace nařizuje podporu DTLS-SRTP a stejně tak jeho volby jako hlavního schéma pro výměnu klíčů. Odpadá tak nebezpečí jejich odhalení (což byl problém výměny klíčů ve formě SDP).

1.3.3 TLS

Protokol Transport Layer Security (TLS) poskytuje šifrování, kontrolu a integritu dat mezi dvěma komunikujícími aplikacemi. Je navržen tak, aby byl používán přes některý ze spolehlivých transportních protokolů jako je protokol TCP. TLS se skládá ze dvou vrstev: TLS Record Protocol a Handshake TLS Protocol. Na nejnižší úrovni je protokol TLS Record, který zajišťuje soukromí a spolehlivost spojení. Zabezpečuje šifrování dat a kontrolu integrity zpráv. Protokol Handshake TLS umožňuje serveru a klientovi vzájemné ověření. Používá se k vyjednávání šifrovacího algoritmu a šifrovacích klíčů, které jsou potřebné pro šifrování dat ve vrstvě protokolu TLS Record Protocol. TLS Handshake Protocol poskytuje autentizaci identity uživatelů spojení pomocí asymetrické, nebo symetrické kryptografie. TLS je nezávislé na aplikačním protokolu, který běží nad ním. Když je TLS handshake kompletní, aplikační protokol může bezpečně přenášet data přes TLS [4].

1.3.4 DTLS

Datagram Transport Layer Security (DTLS) protokol umožňuje aplikacím klient-server komunikovat způsobem, který je navržen tak, aby zabránil odposlechu, manipulaci, nebo podvržení zpráv, stejně jako TLS. DTLS-SRTP je založen na protokolu DTLS 1.0, který zase vychází z TLS 1.1. Na rozdíl od TLS poskytuje DTLS komunikaci přes protokol UDP. DTLS tak zachovává možnost nespolehlivého doručování dat na aplikační vrstvě [10]. Handshake DTLS pro relaci DTLS-SRTP je uveden na obrázku č.1.6.



Obrázek 1.6: *DTLS-SRTP handshake*

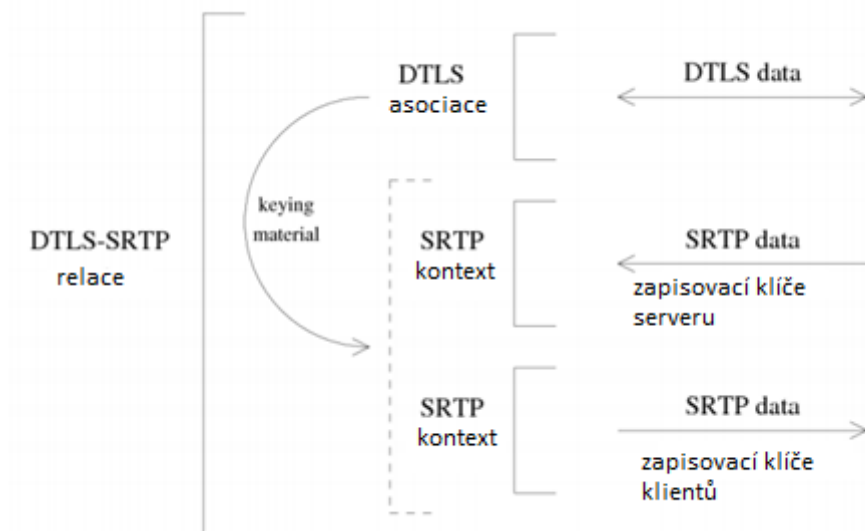
Handshake začíná zprávou ClientHello, která obsahuje seznam šifrovacích sad, které chce klient použít pro šifrování dat. Server vybírá z tohoto seznamu jednu podporovanou šifrovací sadu, a zahrne jej do zprávy ServerHello. Tato šifrovací sada je dále používána pro šifrování DTLS dat. V relacích DTLS-SRTP obsahují zprávy ClientHello a ServerHello příponu use_srtp, která se používá k vyjednávání profilu ochrany pro šifrovací algoritmus SRTP. Při vyjednávání DTLS spojení používají účastníci komunikace certifikáty X.509 k výměně šifrovacích klíčů. Během sestavení spojení odesílají server i klient certifikáty druhé straně. Kromě výměny certifikátů odesílají účastníci komunikace i tzv. Fingerprints. FingerPrints jsou hashem certifikátu a spojují výměnu klíčů DTLS na mediální a signalizační rovině.

1.3.5 DTLS-SRTP

DTLS-SRTP je rozšíření protokolu DTLS, jehož úkolem je vyjednání klíčů pro protokol SRTP, je definován v RFC 5746 [13]. Každá DTLS-SRTP relace chrání jeden pár zdrojových a cílových portů, obecně jeden RTP, nebo RTCP tok. Každá relace obsahuje jednu asociaci DTLS, tedy spojení chráněné protokolem DTLS a jeden, nebo dva kontexty SRTP v jednom směru na daném zdrojovém a cílovém portu, takže pro obousměrnou relaci DTLS-SRTP jsou potřebné dva kontexty SRTP. K vytvoření DTLS asociace uživatelé spojení provedou DTLS handshake na dvojici zdrojového a cílového portu. Tato asociace může být použita k vytvoření klíčového materiálu pro vytvoření klíče v SRTP. Tento klíčový materiál je složen z master klíče a soli pro klienta i server [10].

Klíčový materiál je dále poslán do tzv. SRTP odvozovacího/derivačního mechanismu klíčů, který produkuje SRTP a SRTCP klientské a serverové klíče. Klientské klíče využívá

klient k šifrování a autentizaci odchozích paketů a server je využívá k dešifrování a ověření autenticity příchozích paketů. Obdobně server využívá serverového klíče k šifrování a autentizaci odchozích paketů a klient je využívá k dešifrování a ověření autenticity příchozích paketů.



Obrázek 1.7: *Klientská DTLS-SRTP relace*

Obrázek č.1.7 obsahuje obousměrnou klientskou DTLS-SRTP relaci. Tato relace obsahuje asociaci DTLS a jeden kontext SRTP pro příchozí data a další kontext SRTP pro odchozí data. Oba kontexty získají klíčový materiál pro SRTP z asociace DTLS.

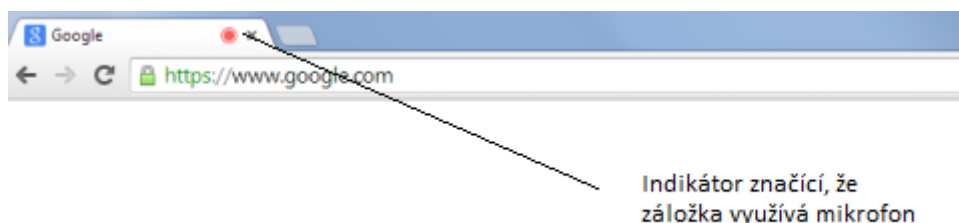
Jelikož se jedná o stranu klienta, příchozí data používají šifrovací klíč serveru a odchozí data používají šifrovací klíč klienta. DTLS pakety jsou doručeny oběma způsoby prostřednictvím jediné asociace DTLS. Obvykle je přenos RTCP odeslán na jiný port než RTP. Použití různých portů vyžaduje dvě relace DTLS-SRTP, jednu pro RTP a jednu pro RTCP.

1.3.6 SOP (Same Origin Policy)

SOP je jedna z vlastností webových prohlížečů, která vyžaduje, aby všechny součásti webových stránek byly získány ze stejného zdroje. Příkladem může být načtení webové stránky. SOP nedovoluje skriptu zaslání požadavku na získání zdrojů na libovolný server, vždy musí jít o server, ze kterého skript pochází. Skripty jsou spouštěny v rámci webových prohlížečů v tzv. sandboxech. Sandboxy umožňují komunikaci skriptů ze stejného zdroje a zabraňují výměně informací mezi skripty s různým původem.

1.3.7 Přístup k mikrofonu a kameře

WebRTC pracuje s kamerou a mikrofonem, čehož může útočník využít a narušit tím soukromí uživatele. WebRTC řeší tento problém vynucením povolení pro přístup ke kameře a mikrofonu. WebRTC aplikace tedy nemá možnost získat přístup k těmto zařízením bez vědomí uživatele. Webový prohlížeč je zároveň povinen zobrazit skutečnost o využití mikrofonu, případně kamery (viz obrázek č.1.8).



Obrázek 1.8: *Identifikátor využití mikrofonu*

1.4 WebRTC - komunikace mezi klienty

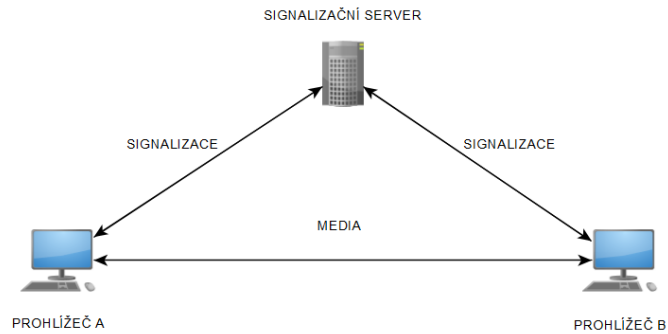
Pro vytvoření spojení mezi prohlížeči využívá WebRTC již výše zmiňované RTCPeerConnection API, toto API má při prvotním vytváření spojení několik důležitých funkcí, mezi které patří zjištění komunikačních proporcí obou klientů, jako jsou např. podporované kodeky, schopnost rozlišení a podporované formáty. Tyto informace slouží k popisu SDP relace a slouží také k sestavení zprávy typu Answer.

V rámci popisu komunikace můžeme sestavení spojení popsat v ideální prostředí tzn. bez využití síťových prvků a funkcí typu firewall a NAT a v reálném prostředí.

1.4.1 Komunikace v ideálním prostředí

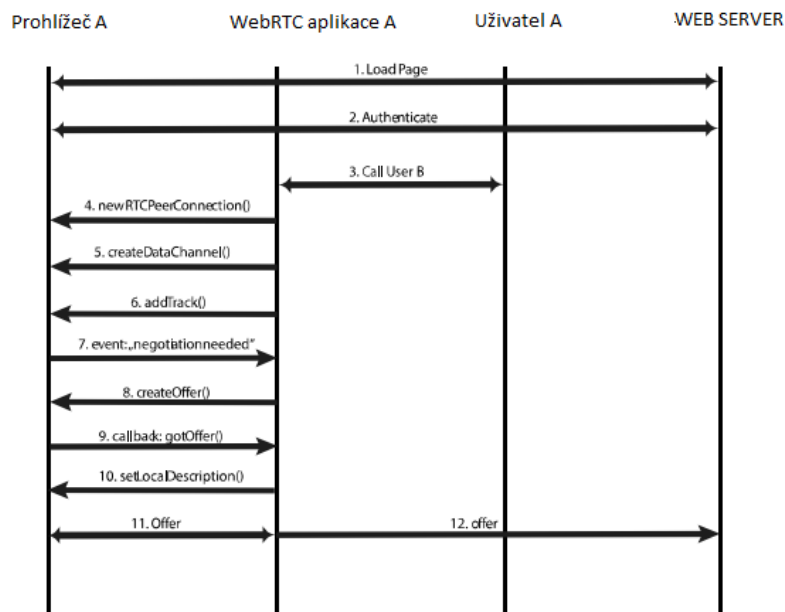
Jelikož se P2P komunikace neobejde bez prvotního kontaktování protistrany, neobejde se komunikace WebRTC bez signalizačních serverů. Signalizační servery se využívají v počátečním navázání spojení a jejich prostřednictvím může WebRTC klient říci, na jaké adrese je možno ho kontaktovat.

V ideálním prostředí, počítejme, že každý WebRTC klient v síti má svou unikátní IP adresu a port, přes které může komunikovat s ostatními klienty a z toho důvodu, nejsou zapotřebí mechanismy pro překonání síťových překladáčů NAT (viz obrázek č.1.9).



Obrázek 1.9: *Signalizace v ideálních podmínkách*

Na začátku sestavení spojení pošle WebRTC klient A požadavek na vytvoření spojení s WebRTC klientem B tento požadavek je reprezentován zprávou offer (viz obrázek č.1.10). Každý klient využívající službu WebRTC je reprezentován registračním záznamem na registračním serveru, na základě registračního záznamu ověří signalizační server existenci uživatele. Klient B následně reaguje na přeposlanou zprávu offer zprávou typu answer, kterou přepoše zpět klientovi A skrz signalizační server. Po domluvení společných parametrů, které se následně použijí pro sestavení P2P spojení je RTCPeerConnection API připraveno navázat spojení a může začít s přenosem mediálních dat.

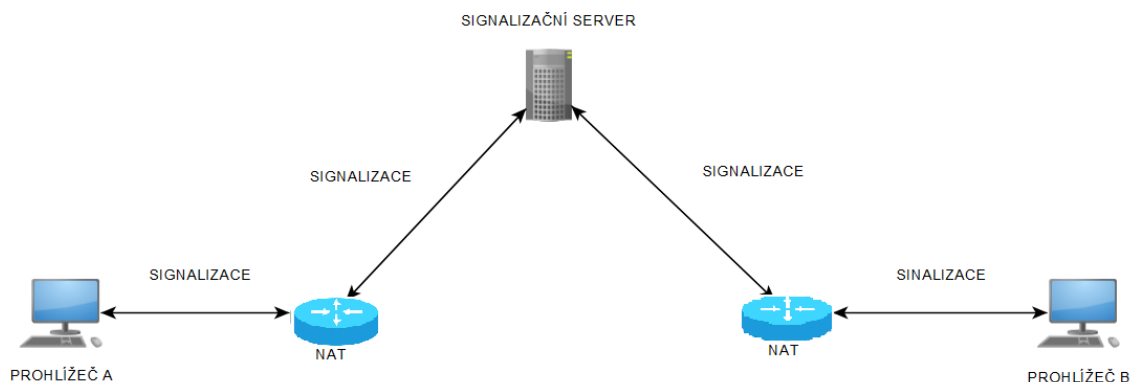


Obrázek 1.10: *Schéma navazování spojení 1*

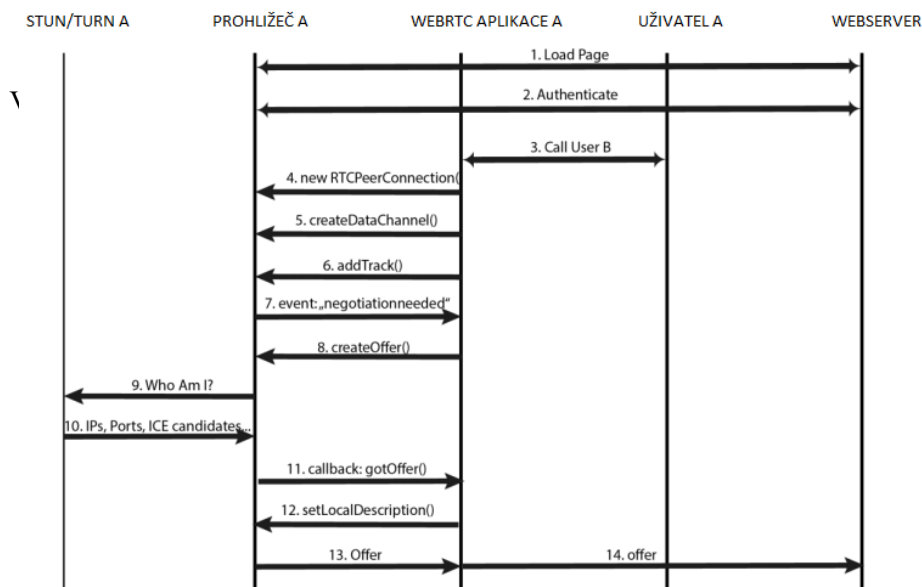
1.4.2 Komunikace v reálných podmínkách

V reálných podmínkách, jak můžeme vidět na obrázku č.1.11 se z důvodu bezpečnosti, většina zařízení nachází za firewally, které filtrují síťový provoz a blokují nepovolené služby, porty a protokoly. Také jsou zařízení umístěná za síťovým překladačem NAT. Z tohoto důvodu nelze navázat spojení pouze pomocí signalizačního serveru.

V reálných podmínkách, kdy jsou zařízení umístěná za síťovým překladačem NAT, se pro identifikaci zařízení ve veřejné síti využije STUN severu. Jak lze vidět na obrázku č.1.12, RTCPeerConnection klienta A kontaktuje STUN server na základě výsledků NAT mapování provedenou metodou ICE a tímto získá potřebné informace k jednoznačné identifikaci ve veřejné síti.



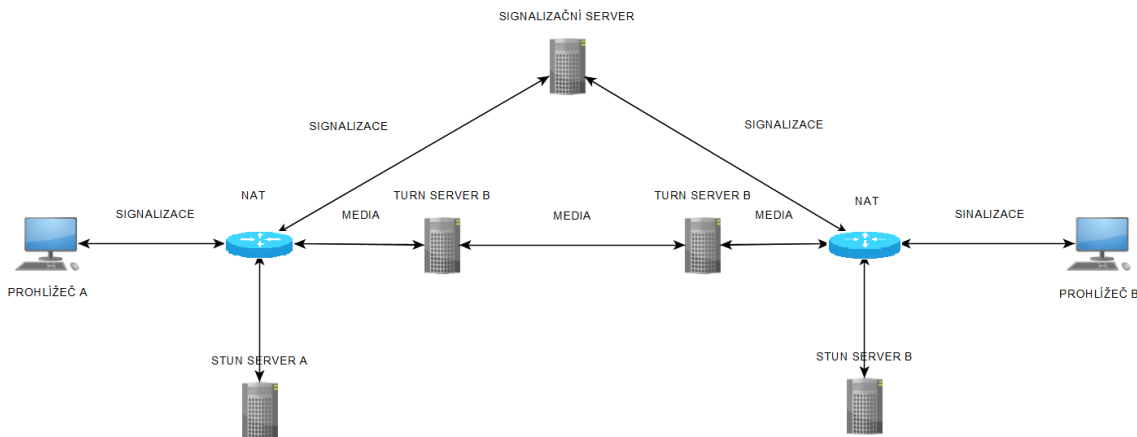
Obrázek 1.11: *Signalizace v reálných podmínkách*



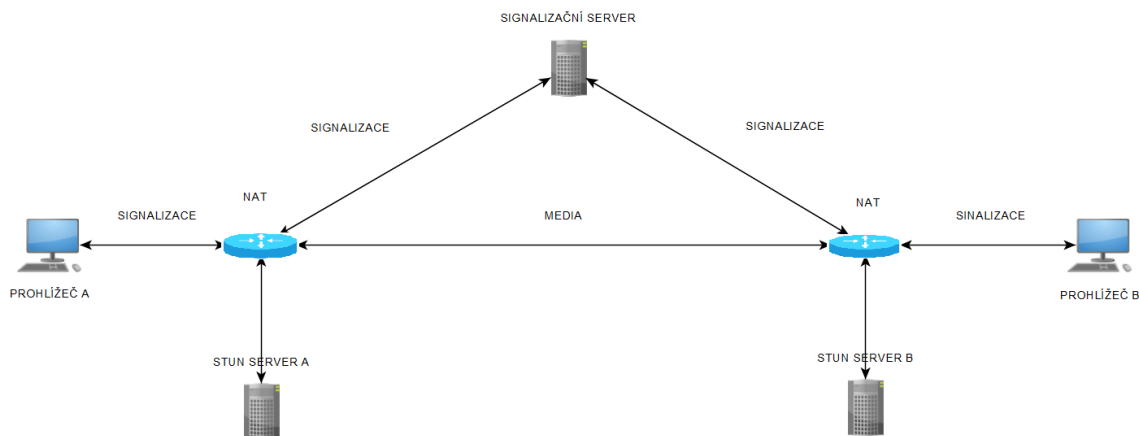
Obrázek 1.12: *Schéma navazování spojení 2*

Ve většině případů je STUN server používán pouze během sestavování spojení. Po navázání této relace je média stream sestaven přímo mezi klienty.

V případě, že se STUN serveru nepodaří zjistit veřejnou IP adresu, porty a typ překladač NAT klienta A, využije ICE TURN severu, jehož funkcionality je popsána výše. Ve většině případů TURN server plní funkci media proxy serveru a zůstává mezi klienty i po úspěšném sestavení spojení (viz obrázek č.1.14).



Obrázek 1.13: *Signalizace v reálných podmínkách STUN*



Obrázek 1.14: *Signalizace v reálných podmínkách TURN*

1.5 Možnosti přenosu signalizace ve WebRTC

Jak již bylo řečeno, signalizační protokoly a metody nejsou ve WebRTC standardu specifikovány a to z důvodu kompatibility se zavedenými technologiemi. V praxi se ovšem používá několik řešení, které jsou již dostatečně odzkoušeny a doporučeny komunitou. Jedná se jmenovitě o tato řešení [11]:

- SIP
- Comet/XHR/SSE
- WebSocket
- SIP over WebSocket
- XMPP/Jingle
- Data Channel

1.5.1 SIP

SIP neboli Session Initiation Protocol je signalizační protokol zajišťující počáteční inicializaci, změnu a ukončení multimediální relace. Jelikož je tento protokol textový, může být přirovnáván k protokolu HTTP a protokolu SMTP. V rámci relace, která je protokolem zajištěna, se může jednat o audio, nebo audio-video hovor a to jak mezi dvěma účastníky tak i mezi více účastníky. Jelikož se ale jedná o signalizační protokol, popisuje SIP jen samotnou relaci ovšem ne data, které jsou v rámci relace vyměňována. V referenčním modelu TCP/IP nalezneme protokol SIP v aplikační vrstvě. Pro přenos SIP zpráv můžeme využít jak transportní protokol UDP tak TCP. V současné době je protokol dále vyvíjen a stále se k němu přidávají nové možnosti a funkce [9].

1.5.1.1 Činnosti protokolu

Pro vytvoření a řízení multimediální relace musí SIP zajistit následující činnosti:

- Lokalizace účastníka - nalezení spojení s koncovou stanicí
- Zjištění stavu účastníka - zjištění, jestli je účastník schopen relaci navázat
- Zjištění možnosti účastníka - zjištění, jaké jsou možnosti účastníka (typ kodeku, maximální přenosová rychlost)
- Navázání spojení
- Řízení probíhajícího spojení - případné změny v průběhu relace a činnosti s jejím ukončováním

1.5.1.2 SIP signalizace

K signalizaci v SIP dochází díky výměně zpráv, tyto jsou dvojího druhu - žádosti, říká se jim též metody a odpovědi [8].

Mezi metody protokolu patří:

- REGISTER – registrace účastníka na SIP Proxy serveru
- INVITE – zahájení komunikace

- ACK – potvrzení zahájení relace
- CANCEL – přerušení zahajování relace ještě před jejím navázáním
- BYE – ukončení probíhající relace
- OPTIONS – požádá o informace o možnostech volajícího, aniž by se sestavilo volání

```

Session Initiation Protocol
Request-Line: INVITE sip:107@10.172.0.2 SIP/2.0
Message Header
Via: SIP/2.0/UDP 10.172.0.101:5060;branch=z9hG4bK59fab8a8a649810a
From: "101" <sip:101@10.172.0.2>;tag=0374a1343263be14
To: <sip:107@10.172.0.2>
Contact: <sip:101@10.172.0.101:5060>
Supported: replaces, timer
Call-ID: d61d626db1c1d19de@10.172.0.101
CSeq: 1660 INVITE
User-Agent: Grandstream GXP2000 1.1.0.14
Max-Forwards: 70
Allow: INVITE,ACK,CANCEL,BYE,NOTIFY,REFER,OPTIONS,INFO,SUBSCRIBE,UPDATE,PRACK
Content-Type: application/sdp
Content-Length: 307
Message body
Session Description Protocol
Session Description Protocol Version (v): 0
Owner/Creator, Session Id (o): 101 8000 8000 IN IP4 10.172.0.101
Session Name (s): SIP Call
Connection Information (c): IN IP4 10.172.0.101
Time Description, active time (t): 0 0

```

Obrázek 1.15: *Příklad INVITE zprávy*

Odpovědi protokolu používají „stovkové“ rozdělení odpovědí. Vedle číselného označení mají jednotlivé odpovědi také textovou verzi např. 200 – OK, 100 – Trying, 180 – Ringing atd [8].

Odpovědi jsou rozděleny do těchto kategorií:

- 1xx – průběh – krok probíhá bez problémů, ale ještě není ukončen
- 2xx – úspěch – krok byl ukončen bez problémů
- 3xx – přesměrování – krok probíhá, ale ještě se v souvislosti s ním něco očekává
- 4xx – chyba klienta – požadavek je chybný a nemůže být serverem zpracován
- 5xx – chyba serveru – požadavek je zřejmě v pořádku, ale chyba je na straně serveru
- 6xx – fatální chyba – fatální chyba, kterou nelze zpracovat

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 161.66.3.161:5061;branch=z9hG4bK0791962b005f2d1d18cade166;X-
DptMsg=135
Call-ID: bda483bac40353eb3a414e8bc877f8d2@10.18.5.64
From: <sip:161.66.3.161:5060>;tag=eafd7848
To: <sip:161.66.3.161>;tag=77ae3429
CSeq: 1 OPTIONS
Allow:
INVITE,ACK,CANCEL,OPTIONS,BYE,REGISTER,PRACK,INFO,UPDATE,SUBSCRIBE,
NOTIFY,MESSAGE,REFER,PUBLISH
Content-Length: 0
```

Obrázek 1.16: *Příklad odpovědi typu 200 - OK*

1.5.2 Obsah hlavičky

Hlavička SIP paketu obsahuje důležité informace pro jeho průchod sítí. Obsahuje například adresu odesílatele, příjemce nebo pořadové číslo probíhajícího hovoru[7]

- Call-ID – identické číslo hovoru, generované klientem
- Contact – zde je uložena SIP adresa, pomocí které je možno kontaktovat druhou stranu bez nutnosti kontaktovat redirect server.
- CSeq – pořadové číslo žádosti v rámci jednoho hovoru. Při opakování žádosti je číslo stejné. Číslo se zvyšuje po zaslání každého požadavku INVITE
- From – obsahuje adresu odesílatele
- To – obsahuje adresu příjemce
- Via – do této hlavičky každý proxy server vkládá svou adresu. Při odesílání opačným směrem ji odebírá a zároveň kontroluje, zdali adresa, na kterou odesílá, není v této hlavičce obsažena. Zabraňuje se tím vzniku smyček

1.5.3 Adresy

Adresy užívané SIP protokolem velice připomínají emailovou adresu. Jsou označovány jako SIP URI a od e-mailových adres se liší prakticky jen návěštím „sip, jak naznačují následující příklady:

- sip: Honza@192.168.150.134
- sip: 12345@vsb.cz
- sip: Honza@vsb.cz

1.5.4 SIP - komponenty

SIP v zásadě používá tyto síťové komponenty:

- Uživatelský agent - UA (User agent) je souhrnný název používaný pro koncová zařízení obsahující klientského i serverového UA.
- Redirect server - Redirect sever se uplatní v případě, kdy klient nezná IP adresu serveru, u kterého se nachází volaný uživatel. V tomto případě zjistí redirect server žádanou adresu prostřednictvím lokalizační služby a tuto adresu předá zpět klientskému UA.

- Proxy server - Proxy server se chová podobným způsobem jako redirect server. Ovšem po obdržení adres(y) z lokalizačního serveru sám naváže spojení se serverem volaného uživatele a potvrdí navázání spojení volajícímu klientovi.
- Lokalizační server - Lokalizační server, nebo také adresářový server slouží jako zdroj informace o umístění čísla/adresy klienta pro proxy a redirect servery.

1.6 Comet/XHR/SSE

Comet, někdy nazývaný také jako XHR, nebo SSE je model webové komunikace, který využívají webové aplikace pro zajištění komunikace mezi klientem a serverem prostřednictvím protokolu HTTP. Tato komunikace využívá dlouho trvajících spojení, které je zajištěno cíleně a to pomocí zprávy „http request“. Tato zpráva umožňuje serveru zasílat data do webových prohlížečů [11].

1.7 WebSocket

WebSocket je komunikační protokol, který umožňuje oboustrannou komunikaci mezi klientem a serverem za pomoci protokolu HTTP. Protokol WebSocket stejně jako protokol HTTP je umístěn na sedmé vrstvě referenčního modelu ISO/OSI a jako takový závisí na protokolu TCP, který je umístěn ve čtvrté vrstvě tohoto modelu. I když jsou protokoly WebSocket a HTTP různé, RFC 6455 uvádí, že je WebSocket navržen tak, aby fungoval přes HTTP porty 80 a 443[15]. Pro dosažení kompatibility těchto protokolů využívá handshake WebSocketu hlavičku Upgrade HTTP, která změní protokol HTTP na protokol WebSocket. Jak již bylo zmíněno WebSocket spojení využívá transportní protokol TCP, toto spojení je definováno dvojicí IP adres, použitým portem a protokolem pro komunikaci. Použití WebSocketu umožňuje výměnu informací v reálném čase a to i v režimu full-duplex. Výhodou spojení prostřednictvím WebSocketu je také eliminace problémů s průchodem skrze firewall, nebo NAT.

1.7.1 Navázání spojení

Před každým začátkem WebSocket komunikace předchází automatizovaný proces vyjednávání, který má za úkol dynamicky nastavit parametry komunikačního kanálu zřízeného mezi dvěma entitami. Tento proces se nazývá handshake.

Účelem tohoto procesu je dosažení kompatibility mezi serverem a entitami podporující HTTP. Jeden port pak může být použit pro HTTP klienty, kteří provozují komunikaci se serverem a zároveň pro WebSocket klienty komunikující se stejným serverem [15]. Jak již bylo zmíněno WebSocket spojení používá porty 80 a 443 takže tato komunikace nepředstavuje problém.

Každé WebSocket spojení začíná žádostí klienta:

```
GET ws://echo.example.com/ HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGh1IHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Request-URI metody GET, které je uvedeno na prvním řádku výše uvedené metody, je využito k identifikaci koncového bodu spojení WebSocket, pro kterého je žádost určena. Na druhém řádku klientovi žádosti se nachází jméno hostitele (hostname), které využívá klient i server k zjištění, který hostitel je zrovna používán. Další žádost se používá k výběru jedné z možností, kterou protokol WebSocket nabízí. Pole "Sec-WebSocket-Protocol" se používá při zahájení WebSocket komunikace a je určené k potvrzení enkapsulovaného protokolu který budou klient a server v rámci WebSocket využívat [15]. Toto pole je přenášeno od klienta na server a zpět ze serveru ke klientovi. Informace o využitém protokolu umožňuje skriptům jak výběr samotného enkapsulovaného protokolu, tak jistotu, že server souhlasí s jeho použitím. Enkapsulovaný protokol představuje protokol aplikační vrstvy, práce s tímto polem je klíčová jelikož představuje možnost, jak lze přenášet například SIP zprávy pomocí WebSocket zpráv.

Pole Upgrade poskytuje mechanismus přepnutí protokolu na protokol, který není kompatibilní s protokolem HTTP. Klient se tímto táže serveru, jestli je možno použít protokol uvedený v poli Upgrade místo protokolu HTTP. Pokud server tento protokol podporuje, přepne protokol na protokol, který je v poli Upgrade uveden.

Pro prokázání přijetí klientovi žádosti potřebuje server dva druhy informací [15]. Na základě jejich kombinací vytváří odpověď. První druh informace pochází z pole "Sec-WebSocketKey". Z tohoto pole si server vezme hodnotu a spojí ji s globálně jedinečným identifikátorem "258EAF5-E914-47DA-95CA-C5AB0DC85B11". Je vysoce nepravděpodobné, že řetězec vzniklý tímto spojením je používán koncovými body, které nerozumí WebSocket komunikaci. Na vzniklý řetězec je dále použit hash typu SHA-1 a vzniklý hash je dále zakódován na bázi base64. Výsledek je vrácen jako odpověď serveru na přijatou žádost [15]. Tato výsledná hodnota objeví ve vráceném poli "Sec-WebSocketAccept".

Odpověď serveru:

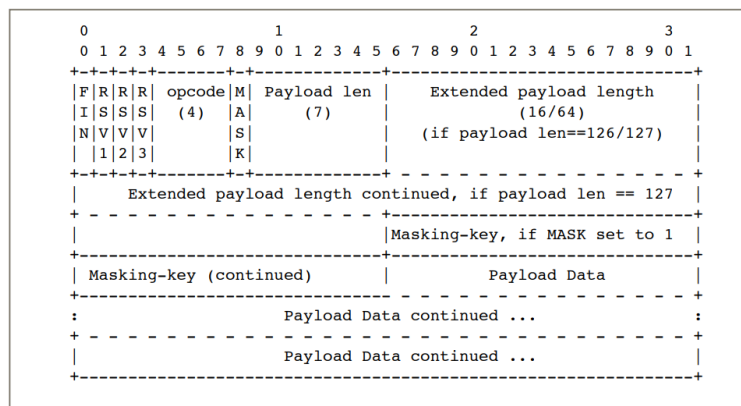
```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

První řádek HTTP response zprávy obsahuje HTTP status line s návratovým status kódem 101: HTTP/1.1 101 Switching Protocols. Všechny zprávy, které obsahují jiný status kód, než je 101 znamenají, že proces vyjednávání WebSocket spojení není kompletní a stále platí sématika HTTP. Pole "Upgrade" a "Connection" dokončují HTTP Upgrade. Pole "Sec-WebSocketAccept" dává klientovi na vědomí, zda je server ochotný přijmout spojení. Toto pole musí obsahovat hash klientovy hodnoty zaslané v poli "Sec-WebSocket-Key" spolu s přednastaveným globálně jedinečným identifikátorem. Žádná jiná hodnota nesmí být akceptována jako přijetí zprávy Serverem.

Pole HTTP response zprávy jsou následně kontrolovány WebSocket klientem. Pokud obsažená hodnota pole "Sec-WebSocket-Accept" neodpovídá očekávané hodnotě, nebo pokud chybí, nebo dokonce HTTP návratový kód není 101, spojení nebude sestaveno a WebSocket rámce nebudou odeslány.

1.7.2 WebSocket zprávy

Po úspěšném vyjednávání spojení lze posílat data. To lze až do doby, než je poslán řídicí rámec "Close". Data jsou přenášena pomocí sekvence rámců, tyto rámce jsou z důvodu bezpečnosti maskovány respektive šifrovány a to ze směru od klienta k serveru. Maskování je prováděno bez ohledu na to, zdali se využívá spojení přes TLS či nikoliv. Server nesmí povolit spojení, kde jsou rámce posílány nemaskované, takové spojení musí uzavřít. Server na rozdíl od klienta své rámce nemaskuje a klient na rozdíl od serveru musí uzavřít spojení, když obdrží maskovaný rámec od serveru [15]. Struktura rámce je popsána na obrázku č.1.17.



Obrázek 1.17: Struktura WebSocket rámce

FIN: 1 bit

Znamená, že se jedná o poslední fragment ve zprávě.

RSV1, RSV2, RSV3: každý 1 bit

Souvisí s možným rozšířením. Tyto bity musí být rovny 0. Pokud je přijata nenulová hodnota a žádné z vyjednaných rozšíření nedefinuje význam takové nenulové hodnoty, koncový bod, který takový rámec obdrží, to pak považuje za selhání WebSocket spojení.

Opcode: 4 bity

Definuje význam přenášených dat – "Payload data". Pokud je na jeho místě obdržena neznámá kombinace těchto 4 bitů, přijímající koncový bod tuto situaci musí považovat za selhání WebSocket spojení.

Jsou definovány následující hodnoty:

- %x0 označuje pokračování rámce
- %x1 označuje textový rámec
- %x2 označuje binární rámec
- %x3-7 jsou rezervovány pro další neřídící rámce
- %x8 označuje uzavření spojení
- %x9 označuje ping

Mask: 1 bit

Říká, zda přenášená data (Payload data) jsou maskována. Pokud je nastaven na 1, je přítomen maskovací klíč v poli „masking-key“, a ten je používán k odmaskování užitečných dat. Všechny rámce poslané od klienta k serveru mají tento bit nastaven na 1.

Payload length: 7 bitů, 7+16 bitů, nebo 7+64 bitů

Délka přenášených dat v bytech: pokud je hodnota v rozmezí 0-125, pak se jedná o délku přenášených dat. Jestliže je 126, následující 2 byty interpretované jako 16-bitové celé číslo bez znaménka představují délku užitečné zátěže. Jestliže je 127, následujících 8 bytů je interpretováno jako 64-bitové celé číslo bez znaménka (nejvýznamnější bit musí být 0) a představují délku užitečné zátěže. Přenášené data představují součet dat rozšíření (anglický termín je Extension data) a dat aplikačních (Application data). Délka dat rozšíření může být nulová – nemusí se ve zprávě objevit, v takovém případě je délka přenášených dat rovna délce dat aplikačních.

Masking-key: 0 nebo 4 byty

Všechny rámce poslané od klienta k serveru jsou maskovány pomocí 32-bitové hodnoty, která je obsažena v rámu. Toto pole je přítomné, jestliže mask bit je nastaven na jedna. Chybí v případě, že mask bit je nastaven na 0.

Payload data: (x+y) bytů

Přenášená data jsou definovány jako data rozšíření spojené s daty aplikačními (jejich součet).

Application data: y bytů

Libovolná aplikační data nastupují do zbytku rámce po jakýkoliv datech rozšíření. Délka aplikačních dat je rovna délce přenášených dat minus data rozšíření.

1.7.3 WebSocket Secure

Technologie WebSocket Secure vylepšuje standardní WebSocket tím, že používá zabezpečení pomocí protokolu TLS, obdobně jako protokol HTTPS. Sestavení spojení z počátku probíhá pomocí nezabezpečené varianty, v průběhu handshaku dojde ke změně na zabezpečenou variantu. WebSocket Secure využívá bezpečnostního modelu popsaného v RFC 6454. Zabezpečená komunikace využívá certifikátu, který je umístěn na straně serveru. Výhodou je, že WebSocket Secure využívá portu 443, což je standardní port pro protokol HTTPS tudíž nemusí být povoleny žádné dodatečné porty na firewallu. Prefix k rozpoznání WebSocket Secure je WSS. Použitím WebSocket Secure lze předcházet útoku typu MiTM a podvržení identity jedné z komunikujících stran.

1.8 SIP over WebSocket

WebRTC popisuje způsob, jakým se prohlížeč stane koncovým bodem komunikace, ale nikoliv jako koncový bod SIP. Existují aplikace napsané v jazyce JavaScript, které používají WebSocket přenos pro vytvoření WebRTC relace tak, aby aplikace byla schopná komunikovat se standardními SIP klienty. Tato metoda funguje stejně jako klasický WebSocket, ale místo původních zpráv se přenáší skrze WebSocket SIP komunikace. WebSocket zprávy lze přepravovat buď v textových, nebo binárních rámcích, z tohoto důvodu musí SIP WebSocket klient i SIP WebSocket servery přijímat textové i binární rámce. Každá zpráva SIP musí být provedena v rámci jedné WebSocket zprávy a zároveň WebSocket zpráva nesmí obsahovat více než jednu SIP zprávu [1].

Pro použití WebSocket SIP podprotokolu musí klient v zahajovací zprávě v poli Sec-WebSocket-Protocol uvést hodnotu sip. Následně je nutné, aby server zprávu opětoval a hodnota v poli Sec-WebSocket-Protocol se shodovala. Po úspěšné výměně zahajovacích zpráv se může začít s přenosem SIP zpráv.

Příklad klientovy zahajovací zprávy:

```
GET / HTTP/1.1 Host: server.diploma.com
```

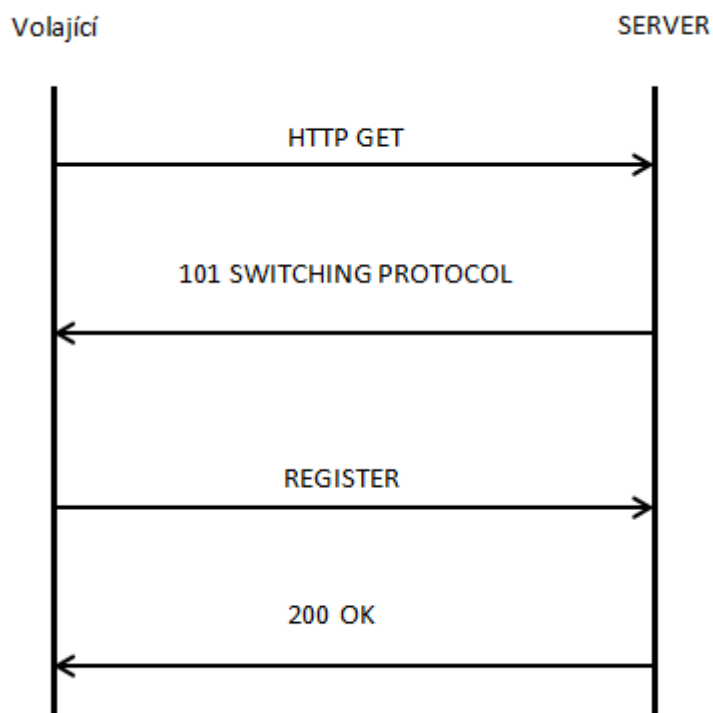
```
Upgrade: websocket Connection:
```

```
Upgrade Sec-WebSocket-Key: dGhlIHhnbXBsZSBub25jZQ ==
```

```
Origin: http://diploma.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13
```

Příklad odpovědi ze strany serveru:

```
HTTP/1.1 101 Switching Protocol
Upgrade: websocket Connection : Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK + xOo =
Sec-WebSocket-Protocol: sip
```



Obrázek 1.18: Registrace uživatele využitím WebSocketu

1.9 XMPP/Jingle

XMPP je komunikační protokol založen na jazyku XML. Ve své podstatě je to streamovací protokol, který umožňuje výměnu fragmentů XML mezi libovolnými dvěma koncovými body sítě. Na rozdíl od většiny protokolů instant messagingu je XMPP otevřeným standardem, který umožňuje uživatelům přístup k sítím pomocí jiných protokolů.

XMPP byl původně nazýván protokolem Jabber, ale toto bylo technické chybné označení, protože Jabber byl jednoduše názvem aplikace používající XMPP. Dnes je XMPP základním protokolem pro aplikace jako např. WhatsApp Messenger a Google Talk a stal se také oficiálním standardem IM ministerstva obrany Spojených států [11][16].

2 FreeSWITCH

Projekt FreeSWITCH vznikl v roce 2006 a za jeho zrodem stojí skupina nespokojených vývojářů projektu Asterisk. Tito vývojáři nebyli spokojeni s politikou a směrem vývoje, kterým se projekt Asterisk ubíral. Oproti Asterisku má FreeSWITCH mnoho vylepšení a již od začátku vývoje byl koncipován se zaměřením na jednoduchost, modularitu a škálovatelnost. FreeSWITCH také nabízí možnost propojení s komunikačními technologiemi a protokoly, jako jsou např. Skype, WebRTC, SIP nebo H.323.

FreeSWITCH se skládá ze stabilního jádra, napsaném v programovacím jazyce C. Na toto jádro se vážou nezávislé moduly, pomocí kterých lze funkcionalitu ústředny FreeSWITCH dále rozšiřovat. Tyto moduly s jádrem komunikují skrze zprávy "events" [5].

2.1 FreeSWITCH a WebRTC

Podpora WebRTC je v ústředně FreeSWITCH zajištěna od verze 1.4 beta, která byla vydána v roce 2014. Od této verze přibyla podpora protokolu SIP over WebSocket, která je pro WebRTC komunikaci nezbytnou součástí. SIP over WebSocket byl implementován do základního modulu mod_sofia, který zajišťuje koncovým bodům SIP komunikace interakci s jádrem ústředny. Pro zpracování HTTP požadavků slouží moduly mod_httapi a mod_http_cache [17]. Oproti Asterisku implementuje FreeSWITCH svůj vlastní signalizační protokol Verto avšak zároveň umožňuje použít i standardní komunikaci SIP.

Tabulka 2.1: FreeSWITCH a podpora WebRTC

FreeSWITCH verze 1.8 LTS	
Podpora WebRTC	ANO
Metody signalizace	Verto, SIP
Transportní metody	Verto over WebSocket, SIP over WebSocket
Podpůrné protokoly	ICE, STUN, TURN, SRTP, AVPF
WebRTC audio	ANO
Podporované audio kodeky	g711, g722, iSAC, iLBC
WebRTC video	ANO
Podporované video kodeky	VP8
WebRTV videokonference	ANO
JavaScript knihovny	JsSIP, sipml5, Verto

3 Asterisk

Asterisk je open-Source software, který umožňuje implementaci telefonní ústředny pomocí běžného HW. Původně byl Asterisk vyvíjen pouze jako telefonní systém pro malou firmu. Dnes se jedná o univerzální „nástroj“ pro budování telefonních systémů. V dnešní době Asterisk nenalezneme pouze u IP PBX, ale i ve VoIP bránách, systémech call center, konferenčních řešeních, hlasové poště a mnoha dalších aplikacích. Asterisk má kolem sebe širokou komunitu, která dle informací na oficiálních stránkách projektu, čítá přes 86 000 registrovaných uživatelů a vývojářů, kteří přispěli k tomu, že dnes Asterisk patří mezi jeden z nejrozsáhlejších komunikačních projektů na světě

Asterisk je napsán v programovacím jazyce C a skládá se z jádra a z modulů, se kterými jádro komunikuje. Tyto moduly poskytují jádru ovladače pomocí, kterých může Asterisk do určité míry ovlivňovat chování externích programů a zařízení tak, aby mezi nimi usadil komunikaci. Stejně jako v případě ústředny FreeSWITCH lze funkcionalitu Asterisku rozšířit pomocí rozšiřujících modulů. Tyto moduly jsou distribuovány jak od vývojářů Asterisku, tak i od členů komunity. Instalace modulů je volitelná a při instalaci Asterisku může uživatel ovlivnit, který moduly budou nainstalovány [3].

3.1 Asterisk a WebRTC

Podpora WebRTC je v Asterisku zajištěna od verze 11. Původně však byla pro WebRTC zajištěna pouze podpora audia. To se změnilo s vydáním verze 14, která poskytla potřebné kodeky i pro podporu videa. Od verze 15 byla přidána podpora videokonferencí a za stabilní LTS verzi je nyní uváděna verze 16. Implementaci samotného WebRTC v Asterisku zajišťují integrovaný HTTP server, který zpracovává HTTP požadavky a přídatné moduly, které umožňují komunikaci přes WebSocket [3][14].

Tabulka 3.1: Asterisk a podpora WebRTC

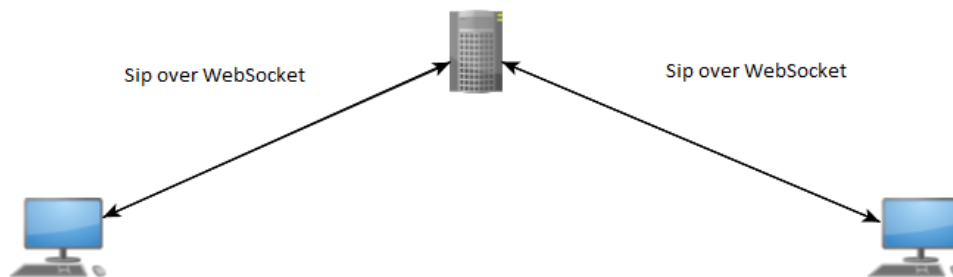
Asterisk verze 1.6 LTS	
Podpora WebRTC	ANO
Metody signalizace	SIP
Transportní metody	SIP over WebSocket
Podpůrné protokoly	ICE, STUN, TURN, SRTP, AVPF
WebRTC audio	ANO
Podporované audio kodeky	g711, g722
WebRTC video	ANO
Podporované video kodeky	VP8
WebRTV videokonference	NE
JavaScript knihovny	JsSIP, sipml5

4 Praktická realizace

WebRTC bylo prakticky realizováno na open-source platformách Asterisk a FreeSWITCH. Tyto platformy byly vybrány z důvodu velké rozšířenosti v praxi, přehledné dokumentace a široké podpory komunity.

Logická topologie (uvedena na obrázku č.4.1), na které je realizována praktická část této práce, se skládá ze dvou počítačů, na kterých je nainstalován operační systém Ubuntu verze 16.04. Xenial Xerus. Tyto počítače využívají klienti, kteří přes webový prohlížeč Mozilla Firefox přistupují na webový server s integrovaným WebRTC klientem Sipml5 a server, který plní roli signalizačního serveru a Apache web serveru. Roli signalizačního serveru budou plnit PBX Asterisk a FreeSWITCH, jejichž instalaci a konfiguraci se tato práce věnuje.

Jako signalizační protokol byl zvolen signalizační protokol SIP, jehož komunikace mezi klienty a serverem je přenášena transportní metodou SIP over WebSocket.



Obrázek 4.1: Logická topologie

4.1 Konfigurace PBX Asterisk

Tato kapitola se zabývá vysvětlením jednotlivých konfiguračních souborů a nezbytných parametrů. Také zde čtenář nalezne analýzu hovoru a sestavení spojení společně s analýzou komunikace v paketovém analyzátoru Wireshark. Postup instalace a konfigurace je rozveden v příloze B.

4.1.1 Konfigurace vestavěného HTTP serveru

WebRTC klient Sipml5 používá WebSocket jako transportní metodu pro výměnu signalizačních zpráv. Pro zprovoznění komunikace mezi Asterisk serverem a Sipml5 klientem je třeba na straně Asterisk serveru vytvořit HTTP server, ke kterému se bude klient připojovat a sestavovat WebSocket spojení. Asterisk pro tento případ obsahuje vlastní integrovaný HTTP server. Pro konfiguraci tohoto HTTP serveru slouží konfigurační soubor `http.conf` [2]. Níže je

uvedený příklad konfigurace HTTP serveru, která zajistí naslouchání asterisk procesu na portu 8088, povolí použití protokolu TLS a také specifikuje certifikát, který bude použit pro ověření komunikace.

```
[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlscipher=AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk/keys/asterisk.pem
```

- enable – hodnota parametru enable určuje, zda bude povoleno HTTP/HTTPS
- bindaddr – hodnota tohoto parametru specifikuje adresu, na kterou se váže HTTP protokol. Adresa 0.0.0.0 představuje tzv. obecnou adresu
- bindport – hodnota tohoto parametru určuje port, na kterém naslouchá HTTP server
- tlsenable – hodnota tohoto parametru určuje, jestli bude povoleno použití TLS protokolu, který zabezpečuje komunikaci.
- tlsservercipherorder – hodnota tohoto parametru specifikuje kryptografické šifry, které bude HTTPS server podporovat
- tlsbindaddr – hodnota tohoto parametru představuje adresu, na kterou se váže HTTPS protokol a port, na kterém bude HTTPS server naslouchat.
- tlscertfile – hodnotou tohoto parametru specifikujeme cestu k certifikačnímu souboru

Jestli server naslouchá na specifikovaném portu, můžeme ověřit tímto příkazem, který je ovšem nutno spustit z konzole Asterisku:

```
http show status
```

Výstup:

```
HTTP Server Status:
```

```
Prefix:
```

```
Server: Asterisk/certified/13.13-cert4
```

```
Server Enabled and Bound to 0.0.0.0:8088
```

```
websocket_enabled=false
```

```
HTTPS Server Enabled and Bound to 0.0.0.0:8089
```

4.1.2 Konfigurace uživatelských účtů

PBX Asterisk využívá pro konfiguraci uživatelských účtů soubor `pjsip.conf`. V tomto souboru jsou specifikovány parametry jednotlivých uživatelských účtů. Tyto účty se dělí do několika sekcí. Jelikož WebRTC klient Sipml5 využívá WebSocket jako transportní metodu pro připojení k HTTP/HTTPS serveru, musí být i uživatelský účet nakonfigurován tak, aby jako transportní metodu využíval právě WebSocket. V tomto případě bude WebSocket používat identifikátor přenosu „Secure WebSocket“ `wss`.

Transportní metoda, kterou uživatelé využívají, se definuje v souboru `etc/asterisk/pjsip.conf`. Definice transportní metody je uvedena níže:

```
[transport_wss] ;název sekce  
type=transport ;typ sekce  
protocol=wss ;protokol  
bind=0.0.0.0:5067 ;adresa a port, na kterém PJSIP naslouchá
```

K transportní metodě se vážou tzv. entity, které reprezentují jednotlivé uživatelské účty, a které tuto transportní metodu využívají. K těmto účtům se následně přistupuje přes WebRTC klienta skrze webový prohlížeč. Každá entita se skládá z několika objektů např. `Aor`, `auth` a `endpoint`. Popis těchto objektů a jejich význam je popsán níže:

```
[1000] ;název sekce  
type=aor ;typ sekce  
max_contacts=1 ;maximální počet registrovaných zařízení  
remove_existing=yes
```

```
[1000] ;název sekce  
type=auth ;typ sekce  
auth_type=userpass ;typ ověřování  
username=1000 ;uživatelské jméno
```

password=1000 ;uživatelské heslo

Výše uvedená konfigurace objektů aor a auth zajišťuje, že entita bude známá jako 1000 a pro ověření bude používat heslo 1000. Tuto konfiguraci využívá objekt endpoint, který odkazuje na objekty aor a auth jako na své konfigurační parametry. Popis jednotlivých parametrů objektu endpoint je uveden níže:

```
[1000] ;název sekce
type=endpoint ;typ sekce
transport=transport_wss ;transportní metoda
aors=1000 ;přiřazení sekce AOR
auth=1000 ;přiřazení sekce AUTH
use_avpf=yes ;použití zpětnovazebních zpráv pro RTCP
media_encryption=dtls ;typ šifrování médií
dtls_ca_file=/etc/asterisk/keys/ca.crt ;cesta k certifikátu
dtls_cert_file=/etc/asterisk/keys/asterisk.pem ;cesta k
certifikátu
dtls_verify=fingerprint ;typ ověřování
dtls_setup=actpass ;typ akceptovatelného připojení pro DTLS
ice_support=yes ; podpora ICE
media_use_received_transport=yes
rtcp_mux=yes
context=default ;přiřazení účtu do kontextu
disallow=all ;zakázání všech povolených kodeků
allow=alaw, VP8 ;povolení kodeku alaw a vp8
```

Pro pochopení výše uvedené konfigurace objektu endpoint zde v krátkosti uvedu význam výše uvedených kroků.

- Je vytvořen koncový bod 1000, který odkazuje na vytvořené objekty aor a auth
- use_avpf=yes zajišťuje, aby byl použit profil AVPF, který podporuje protokol SRTP
- jako šifrovací metoda je zvolena metoda DTLS, ke které jsou spjaty certifikáty a klíče
- je povolena podpora mechanismu ICE

- `media_use_received_transport=yes` říká Asterisku, aby pro odesílání dat využil stejný transport, skrz který data obdržel;
- `rtcp_mux=yes` povoluje přenos RTP a RTCP eventů skrze stejný socket
- `context=default` definuje, že příchozí požadavky na tento koncový bod, budou obslouženy dle kontextu „default“
- explicitně jsou povoleny pouze kodeky `alaw` a `ulaw`

4.1.3 Konfigurace dialplánu

Aby bylo možno provést hovor, je zapotřebí vytvořit směrovací plán.

Níže uvedené řádky definují kontext „default“, který při vytočení čísla 200 přehraje jeden, ze základních zvukových souborů a poté hovor zavěsí. Dále je zde základní pravidlo pro volání linky 1000 a 2000, při vytočení identických čísel, za pomoci PJSIP stacku. Definice kontextu se nachází v souboru `/etc/asterisk/extensions.conf`.

Příklad konfigurace kontextu "default":

```
[default]
exten=> 200,1,Answer()
exten=> 200,2,NoOp(Dovolali jste se na ustrednu)
exten=> 200,3,Playback(demo-congrats)
exten=> 200,4,Hangup()
exten =>1000,1,Dial(PJSIP/1000)
exten =>2000,1,Dial(PJSIP/2000)
```

4.1.4 Předpřipravení webového prohlížeče

Jelikož základní nastavení webových prohlížečů neumožňuje připojení pomocí Secure WebSocketu k serveru, který využívá vlastně podepsaný certifikát, je nutno importovat vlastně podepsaný certifikát do prohlížeče.

4.1.4.1 Konverze certifikátu

Prohlížeče ve většině případů nepodporují certifikáty ve formátu `.pem`. Jelikož certifikát v tomto formátu se generuje pomocí Asterisku, je nutné ho převést do formátu, který webový prohlížeč podporuje. Jeden z těchto formátů je např. formát `.p12`. Pro konverzi certifikátů je vhodné použít nástroj `OpenSSL`. Postup pro konverzi je popsán v příloze E.

4.1.5 Povolení výjimky pro wss transport se serverem

Pro povolení wss transportu pro server s vlastně podepsaným certifikátem, je třeba přejít na stránku `https://<IP_Asterisk_serveru>:8089/ws` a zde přidělit tomuto spojení výjimku. Toto

je nutnou podmínkou, jelikož webové prohlížeče v základním nastavení neumožňují připojení pomocí Secure WebSocketu k serveru.

4.1.6 Analýza testovacího hovoru

Testovací hovor jsem realizoval přes webového klienta Sipml5, jehož integrace je popsána v příloze D. Po vyplnění základních údajů, jejichž vzor je uveden níže, jsem provedl testovací video hovor.

Vzor údajů vyplněných v kolonce registration:

- Display Name: 1000
- Private Identity*: 1000
- Public Identity*: sip:1000@<IP_Asterisk_serveru>
- Password: 1000
- Realm*: <IP_Asterisk_serveru>

Po vyplnění registračních údajů je zapotřebí specifikovat v export módu, WebSocket pro transport a to v kolonce „WebSocket Server URL“

Vzor specifikace WebSocketu

WebSocket Server URL: wss://<IP_Asterisk_serveru>:8089/ws

Po vyplnění potřebných údajů a specifikace WebSocketu je vše připraveno k registraci. Po přihlášení je klient informován hláškou "connected"

Po zaregistrování se zobrazí v konzoli Asterisku informační výpis (viz obrázek č.4.3).

```
== WebSocket connection from '192.168.150.134:53578' for protocol 'sip' accepted using version '13'  
== Endpoint 1000 is now Reachable  
-- Added contact 'sips:1000@192.168.150.134:53578;transport=ws;rtcweb-breaker=no' to AOR '1000' with expiration of 200 seconds  
osboxes*CLI> █
```

Obrázek 4.3: Informace o registraci klienta

Z výpisu uvedeného na obrázku č.4.3 vyčteme, že bylo sestaveno nové WebSocket spojení s koncovým bodem 192.168.150.134:53578, které bude použito pro přenos protokolu SIP. Tento koncový bod se registruje jako pjsip endpoint 1000, využívající transportní metodu ws.

Testovací hovor se v prostředí webového klienta Sipml5 uskutečňuje v nabídce „Call“. Zde lze specifikovat typ hovoru Video, nebo Audio. Po zavolání na koncový účet je uživatel vyzván povolení přístupu k mikrofonu a kameře. O průběhu hovoru je klient informován informačními hláškami "Call in progress", která informuje o pokusu sestavit spojení. A po úspěšném sestavení SIP komunikace se informační hláška mění na hlášku "In Call".

Po úspěšném sestavení hovoru lze v konzoli Asterisku sledovat průběh hovoru, který je uveden na obrázku č.4.4.

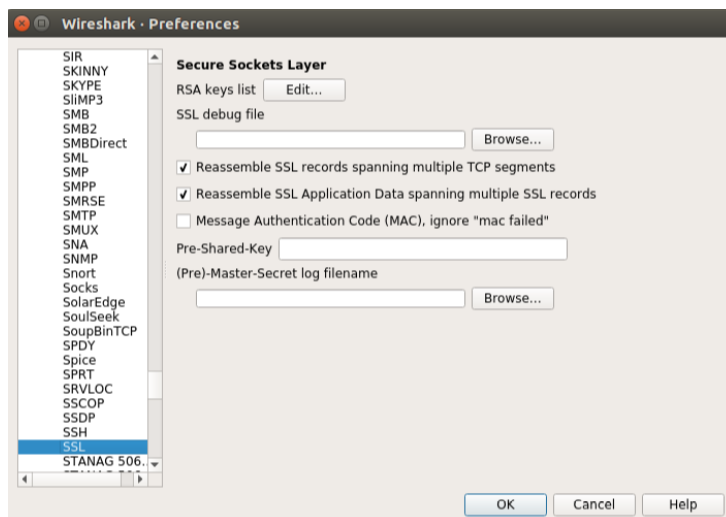
```
-- Channel PJSIP/1000-00000001 left 'simple_bridge' basic-bridge <f3fe6e1c-5
42e-473b-ad46-cd763502c9f7>
-- Channel PJSIP/2000-00000000 left 'simple_bridge' basic-bridge <f3fe6e1c-5
42e-473b-ad46-cd763502c9f7>
== Spawn extension (default, 1000, 1) exited non-zero on 'PJSIP/2000-00000000'
== Setting global variable 'SIPDOMAIN' to '192.168.150.129'
== DTLS ECDH initialized (automatic), faster PFS enabled
-- Executing [2000@default:1] Dial("PJSIP/1000-00000002", "PJSIP/2000") in n
ew stack
== DTLS ECDH initialized (automatic), faster PFS enabled
-- Called PJSIP/2000
-- PJSIP/2000-00000003 is ringing
-- PJSIP/2000-00000003 is ringing
> 0x7f6ee0071c20 -- Strict RTP learning after remote address set to: 192.
168.134.134:44324
-- PJSIP/2000-00000003 answered PJSIP/1000-00000002
> 0x7f6ee003f770 -- Strict RTP learning after remote address set to: 192.
168.150.134:50211
-- Channel PJSIP/2000-00000003 joined 'simple_bridge' basic-bridge <67ae1783
-93be-4830-8f52-5f70d3e32fe5>
-- Channel PJSIP/1000-00000002 joined 'simple_bridge' basic-bridge <67ae1783
-93be-4830-8f52-5f70d3e32fe5>
> 0x7f6ee0071c20 -- Strict RTP learning after ICE completion
> 0x7f6ee003f770 -- Strict RTP learning after ICE completion
> 0x7f6ee003f770 -- Strict RTP switching to RTP target address 192.168.15
0.134:50211 as source
> 0x7f6ee003f770 -- Strict RTP learning complete - Locking on source addr
ess 192.168.150.134:50211
```

Obrázek 4.4: Průběh hovoru v PBX Asterisk

Z výpisu průběhu hovoru uvedeného na obrázku č.4.4 můžeme vidět použitou šifru, která byla použita pro výměnu klíčů a šifrování. V našem případě byla vybrána šifra DTLS ECDH. Dále můžeme vidět vstup do číslovacího plánu s kontextem default. Můžeme si také povšimnout toho, že číslovací plán, postupuje přesně podle pravidel, které jsou specifikovány v souboru *extensions.conf*.

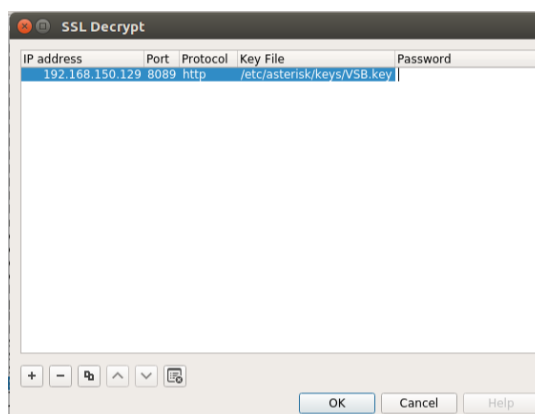
4.1.7 Analýza komunikace

Analýzu síťového provozu jsem provedl ve volně dostupném paketovém analyzátoru Wireshark. Pro analýzu síťového provozu je nutno spustit tento analyzátor s právy uživatele root. Pokud budeme chtít analyzovat WebRTC komunikaci, v našem případě analyzovat SIP zprávy, které přenášíme na stranu WebRTC klienta přes WebSocket, budeme muset upravit HTTP server na straně Asterisku. Standardně se při výměně klíčů (tzv. TCP handshake) používá „nejméně silnější“ společná šifra, mezi klientem a serverem. Mezi takové patří zejména šifry využívající algoritmus ECDH (Elliptic-curve Diffie–Hellman). Šifry využívající ECDH algoritmy znemožňují ověřit autentičnost klienta, a tedy i přes fakt, že vlastníme soukromý klíč nechráněný heslem, se nám nepodaří komunikaci dešifrovat. Proto je nutné vynutit striktně



Obrázek 4.7: Import klíče pro dešifrování komunikace

Klikneme na tlačítko Edit.. a nastavíme parametry dle předlohy uvedené na obrázku č.4.8.



Obrázek 4.8: Vložení klíče

- IP address - Zadáme IP adresu serveru/klienta, mezi kterými probíhá komunikace, kterou chceme dešifrovat.
- Port - Je třeba definovat port, na kterém má probíhat komunikace, kterou chceme dešifrovat
- Protocol - Jako protokol zadáme „http“. Jelikož navázání WebSocket spojení je interpretováno jako požadavky na HTTP upgrade.
- Key File - Tato kolonka slouží pro zadání cesty k soukromému klíči.

- Password - Tato kolonka slouží pro vepsání před-sdíleného klíče, pakliže by byl klíč zašifrován.

Po vložení klíče již lze vidět dešifrované zprávy uvnitř WebSocketu.

No.	Time	Source	Destination	Protocol	Length	Info
33387	828.441619523	192.168.150.130	192.168.150.129	SIP	967	Request: REGISTER s...
28376	729.143766220	192.168.150.129	192.168.150.134	SIP	647	Status: 200 OK (1 ...
28375	729.142004930	192.168.150.134	192.168.150.129	SIP	967	Request: REGISTER s...
28364	728.975879248	192.168.150.129	192.168.150.134	SIP	695	Status: 401 Unautho...
28363	728.971838779	192.168.150.134	192.168.150.129	SIP	967	Request: REGISTER s...
28339	728.353967586	192.168.150.129	192.168.150.130	SIP	647	Status: 200 OK (1 ...
28329	728.351368937	192.168.150.130	192.168.150.129	SIP	967	Request: REGISTER s...
28326	728.317999329	192.168.150.129	192.168.150.130	SIP	695	Status: 401 Unautho...
28325	728.315526415	192.168.150.130	192.168.150.129	SIP	967	Request: REGISTER s...
27208	698.812650573	192.168.150.129	192.168.150.134	SIP	647	Status: 200 OK (1 ...

Masking-Key: 3a841129
Masked payload
Payload

```

Session Initiation Protocol (REGISTER)
  Request-Line: REGISTER sip:192.168.150.129 SIP/2.0
  Method: REGISTER
  Request-URI: sip:192.168.150.129
  [Resent Packet: False]
  Message Header
  ▶ Via: SIP/2.0/WSS df7jal231s0d.invalid;branch=z9hG4bKD1qMqXZ65L2uKnh167TH2Q00p6pE9P3m;rpport
  ▶ From: "1000"<sip:1000@192.168.150.129>;tag=ZF20TpdYfUxQE0ukT73J
  ▶ To: "1000"<sip:1000@192.168.150.129>
  ▶ Contact: "1000"<sips:1000@df7jal231s0d.invalid;rtcweb-breaker=no;transport=wss>;expires=200;
  Call-ID: b4b7b903-8176-f799-b95a-07b79dd1dddd
  ▶ CSeq: 24061 REGISTER
  Content-Length: 0
  Max-Forwards: 70
  ▶ [truncated]Authorization: Digest username="1000", realm="asterisk", nonce="1553381401c30cd68:
  User-Agent: IM-client/OMA1.0 sipML5-v1.2016.03.04
  Organization: Doubango Telecom

```

Obrázek 4.9 Dešifrovaná komunikace

Na obrázku č.4.9 můžeme vidět dešifrovanou SIP zprávu typu REGISTER. Můžeme si povšimnout, že žádost posílá klient 192.168.150.134 a také můžeme vidět, že jako transportní metoda se používá Secure WebSocket.

Za zmínění stojí také zpráva HTTP, která obsahuje požadavek na upgrade. Tuto zprávu můžeme vidět na obrázku č.4.10. Zpráva se používá pro sestavení WebSocket spojení mezi klientem a serverem. WebSocket se na straně serveru navazuje na portu 8089, toto je specifikováno v popisu HTTP jako Host: 192.168.150.134:8089. V popisu samotného WebSocketu lze poté najít i definici vnořeného protokolu. „Sec-WebSocket-protocol: sip“. Tímto jsme si ověřili, že WebRTC využívá pro přenos signalizačních zpráv, protokol SIP tunelovaný v protokolu HTTP.

```

Hypertext Transfer Protocol
  GET /ws HTTP/1.1\r\n
  Host: 192.168.150.129:8089\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Sec-WebSocket-Version: 13\r\n
  Origin: https://192.168.150.129\r\n
  Sec-WebSocket-Protocol: sip\r\n
  Sec-WebSocket-Extensions: permmessage-deflate\r\n
  Sec-WebSocket-Key: cS7z+laPfkHMB2/Cgk++Iw==\r\n
  Connection: keep-alive, Upgrade\r\n
  Pragma: no-cache\r\n
  Cache-Control: no-cache\r\n
  Upgrade: websocket\r\n
  \r\n
  [Full request URI: https://192.168.150.129:8089/ws]
  [HTTP request 1/1]
  [Response in frame: 22]

```

Obrázek 4.10: Zpráva HTTP

4.2 Konfigurace PBX FreeSWITCH

Tato kapitola se zabývá vysvětlením jednotlivých konfiguračních souborů a parametrů, také zde čtenář nalezne analýzu hovoru a sestavení spojení společně s analýzou komunikace v paketovém analyzátoru Wireshark. Detailní postup instalace a konfigurace PBX FreeSWITCH jsou dále popsány v příloze C.

4.2.1 Konfigurace modulu Sofia

Termínem Sofia se v PBX FreeSWITCH označuje klient, který používá protokol SIP pro manipulaci s koncovým účtem. Modul Sofia, který je jeden z hlavních modulů, které PBX FreeSWITCH využívá, zajišťuje konektivitu klientům s koncovými účty.

Pro zajištění podpory WebRTC je nutností povolit modulu Sofia naslouchání WebSocket spojení na určitém portu. Ke konfiguraci modulu Sofia slouží konfigurační soubor soubor `/usr/local/freeswitch/conf/sip_profiles/internal.xml`.

Zajištění naslouchání WebSocket spojení definují tyto parametry, které specifikují cestu k certifikátům a definují port, na kterém se bude naslouchat WebSocket spojení.

```
<param name="tls-cert-dir" value="/usr/local/freeswitch/certs"/>
<param name="wss-binding" value=":7443"/>
```

Ověření zda modul Sofia naslouchá WebSocket spojení zajišťuje příkaz:

```
sofia status profile internal
```

Ve výstupu tohoto příkazu byste měli dohledat tento parametr:

```
WSS-BIND-URL
sips:mod_sofia@<freeswitch_server_IP>:7443;transport=wss
```

4.2.2 Konfigurace uživatelských účtů

PBX FreeSWITCH již v základním nastavení obsahuje předefinované koncové účty. Každý takový účet je specifikován svým vlastním souborem xml. Koncové účty se nacházejí v adresáři `/usr/local/freeswitch/conf/directory/default`. V tomto souboru můžeme pro každý koncový účet specifikovat parametry, jako jsou např. uživatelské jméno, heslo, ID, nebo kontext.

4.2.3 Konfigurace základního kontextu

PBX FreeSWITCH již po instalaci obsahuje základní kontext, který se nazývá default. Veškerá směrovací pravidla vztahující se k tomuto kontextu jsou uložena v konfiguračním souboru `/usr/local/freeswitch/conf/dialplan/default.xml`. Kontext default obsahuje směrovací pravidla, které umožňují směrování mezi přednastavenými koncovými účty. Pro naše účely budeme využívat volání na číslo 9196, které dle vstupních směrovacích pravidel poskytuje volanému echo v reálném čase, viz ukázka níže:

```
<extension name="echo">
```

```

<condition field="destination_number" expression="^9196$">
<action application="answer"/>
<action application="echo"/>
</condition>
</extension>

```

4.2.4 Testovací hovor

V případě pobočkové ústředny FreeSWITCH se registrace uživatelů přes webového WebRTC klienta nijak neliší.

Úspěšně registrovaní uživatelé se v případě pobočkové ústředny FreeSWITCH vypisují příkazem

```
sofia status profile internal reg
```

Z výpisu registrovaných uživatelů uvedeného na obrázku č.4.11 můžeme vyčíst, zda je klient dostupný, z jaké IP adresy a portu je přihlášený, ke kterému uživatelskému účtu a případně další informace, jaký je typ transportní metody atd.

```

Registrations:
=====
Call-ID:      d4a636e5-cf17-dabc-96b4-db93d07b52ae
User:         1000@192.168.150.135
Contact:      "1000" <sips:1000@df7jal23ls0d.invalid;rtcweb-breaker=no;transpo
rt=wss;fs_nat=yes;fs_path=sips%3A1000%40192.168.150.130%3A49786%3Brtcweb-breaker
%3Dno%3Btransport%3Dwss>
Agent:        IM-client/OMA1.0 sipML5-v1.2016.03.04
Status:       Registered(TLS-NAT)(unknown) EXP(2019-03-30 17:23:23) EXPSECS(18
9)
Ping-Status:  Reachable
Ping-Time:    0.00
Host:         osboxes
IP:           192.168.150.130
Port:         49786
Auth-User:    1000
Auth-Realm:   192.168.150.135
MWI-Account:  1000@192.168.150.135
Total items returned: 1

```

Obrázek 4.11: Výpis registrovaných uživatelů

Průběh samotného navazování spojení mezi dvěma koncovými body je uveden na obrázku č.4.12. Povšimnout si můžeme navazování spojení protokolu DTLS. Který začíná procesem HANDSHAKE a končí ustanovením spojení (status READY). Tento protokol byl použit pro přenos RTP/RTCP

```

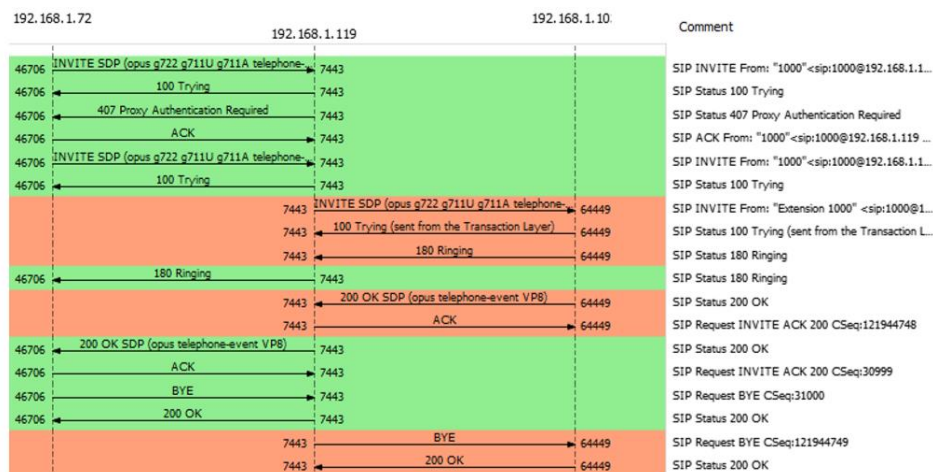
2019-04-09 16:36:29.014172 [NOTICE] sofia.c:7401 Ring-Ready sofia/internal/1001@df7jal23ls0d.invalid!
2019-04-09 16:36:29.014172 [INFO] switch_ivr_originate.c:1246 Sending early media
2019-04-09 16:36:29.441024 [WARNING] switch_core_media.c:4181 NO candidate ACL defined, Defaulting to wan.auto
2019-04-09 16:36:29.486430 [INFO] switch_core_media.c:8690 Activating Audio ICE
2019-04-09 16:36:29.486430 [NOTICE] switch_rtp.c:4799 Activating RTP audio ICE: 0a900395:XxbUs6wZT852AG0x 37.48.42.67:56796
2019-04-09 16:36:29.486430 [INFO] switch_core_media.c:8742 Skipping RTCP ICE (Same as RTP)
2019-04-09 16:36:29.486430 [INFO] switch_rtp.c:3669 Activate RTP/RTCP audio DTLS client
2019-04-09 16:36:29.486430 [INFO] switch_rtp.c:3832 Changing audio DTLS state from OFF to HANDSHAKE
2019-04-09 16:36:29.486430 [NOTICE] sofia_media.c:92 Pre-Answer sofia/internal/1000@192.168.150.135!
2019-04-09 16:36:37.109737 [NOTICE] switch_rtp.c:1280 Auto Changing audio stun/rtp/dtls port from 37.48.42.67:56796 to 192.168.150.130:50353 idx:1
2019-04-09 16:36:37.384392 [INFO] switch_rtp.c:3206 Changing audio DTLS state from HANDSHAKE to SETUP
2019-04-09 16:36:37.409860 [INFO] switch_rtp.c:3113 audio Fingerprint Verified.
2019-04-09 16:36:37.409860 [INFO] switch_rtp.c:4104 Activating audio Secure RTP SEND
2019-04-09 16:36:37.409860 [INFO] switch_rtp.c:4082 Activating audio Secure RTP RECV
2019-04-09 16:36:37.409860 [INFO] switch_rtp.c:3155 Changing audio DTLS state from SETUP to READY

```

Obrázek 4.12: Nastavení parametrů hovoru

4.2.5 Analýza komunikace

Zachycená komunikace se nijak neliší oproti komunikaci ve schématu s využitím pobočkové ústředny Asterisk, Komunikace je opět celá skryta v transportním protokolu TCP. Pro dešifrování je třeba do programu Wireshark naimportovat příslušný klíč, který byl použit pro šifrování komunikace stejně jako v případě použití PBX Asterisk. Tento postup je uveden v podkapitole 4.1.7. Na obrázku č.4.13 je ukázka již dešifrované standardní SIP komunikace, zachycující video-hovor mezi WebRTC klienty s registrovanými účty 1000 a 1001. Princip této SIP komunikace je naprosto standardní. Povšimnout si můžeme, že jako kodek pro audio byl použit opus, který je integrován v PBX FreeSWITCH a měl by poskytnout větší kvalitu hovorů než kodek g711a. Pro video byl použit kodek VP8.



Obrázek 4.13: SIP flow

4.3 Zhodnocení

PBX Freeswitch implementuje WebRTC pomoci rozšiřujících modulů `mod_httapi` a `mod_http_cache`, které zajišťují zpracování HTTP požadavků. Tím se PBX FreeSWITCH liší od PBX Asterisk, jelikož zde se WebRTC implementuje za použití integrovaného HTTP serveru. Jako signalizační protokol pro WebRTC je v případě PBX FreeSWITCH použit protokol SIP, nebo signalizační protokol Verto, jehož použití umožňuje realizaci videokonferenčních hovorů.

Jak již bylo zmíněno PBX Asterisk implementuje WebRTC za použití integrovaného HTTP serveru, a transportní metody SIP over WebSocket. Díky této transportní metodě dochází k tunelování standardního signalizačního SIP protokolu v protokolu HTTP.

Závěr

V teoretickém úvodu této práce byl představen projekt společnosti Google, který nese název WebRTC. Ve zkratce byla představena jeho historie a jeho nesporné výhody, konkrétně absence rozšiřujících plug-inů třetích stran a nutnosti instalace klientských aplikací. Dále byly představeny protokoly a mechanismy, které jsou s projektem WebRTC spjaty, a které jsou implementovány přímo do webového prohlížeče pomocí jazyka HTML5 a JavaScript kódu. Dále byla představena architektura WebRTC API a její nejdůležitější rozhraní, které zajišťují funkcionalitu WebRTC klientů. Důraz byl kladen také na představení a popsání protokolů a mechanismů, které WebRTC využívá a které jsou pro zajištění komunikace nezbytné. Z pohledu zabezpečení provozu jsou detailně popsány mechanismy a protokoly, které projekt WebRTC používá pro zajištění autentizace, integrity a důvěry dat, konkrétně byl popsán především protokol DTLS-SRTP a princip výměny klíčů.

Po obeznámení se základními mechanismy a API rozhraními, je představen princip komunikace mezi klienty a možnosti signalizace. Jelikož WebRTC blíže nespécifikuje signalizační protokol, může být signalizace určena typem signalizačního serveru, který je nasazen v komunikační trase. Ten může dále sloužit jednak jako brána pro signalizaci i média, anebo jako pobočková ústředna nabízející další komunikační služby. Právě PBX ústřednám je věnovaná druhá část této práce, která je zaměřena na podporu WebRTC v PBX Asterisk a FreeSWITCH.

V praktické části této práce jsem provedl konfiguraci WebRTC na uvedených pobočkových ústřednách a jednotlivé konfigurace jsem zdokumentoval v podobě návodu. Tento návod popisuje krok po kroku konfiguraci WebRTC a následnou analýzu funkčního provozu z pohledu jeho zabezpečení. Po dodržení veškerých kroků v návodu bude čtenář schopen zrealizovat videohovor pomocí WebRTC klienta Sipml5 skrze konfigurované open-source PBX, kdy závěry práce mohou být také využity při výuce předmětů zaměřených na zabezpečení multimediálního provozu a jako vhodný základ pro vytvoření moderní komunikační infrastruktury na rozhraní WebRTC.

Práce může být dále rozšířena o implementaci tzv. WebRTC brány, která slouží k propojení WebRTC klientů, kteří komunikují skrze jiný komunikační protokol, nebo o kombinaci WebRTC brány a PSTN (Public Switched Telephone Networks) brány a propojit tak hovor z prohlížeče s PSTN telefonem.

Díky velkému potenciálu a nesporným výhodám, které WebRTC přináší, lze očekávat, že se projekt WebRTC bude v budoucnu rozšiřovat a zaujme místo stávajících aplikací jako je např. Skype.

Seznam použitých zdrojů

- [1] A Study of WebRTC Security [online]. [cit. 2019-04-15]. Dostupné z: <https://webrtc-security.github.io/>
- [2] Asterisk 16 Configuration_res_pjsip [online]. [cit. 2019-04-15]. Dostupné z: https://wiki.asterisk.org/wiki/display/AST/Asterisk+16+Configuration_res_pjsip
- [3] Asterisk Ready To Get Started With Asterisk? [online]. [cit. 2019-04-15].
- [4] Datagram-tls [online]. [cit. 2019-04-15]. Dostupné z: <https://blog.cryptographyengineering.com/2012/01/10/attack-of-week-datagram-tls/>
- [5] FreeSWITCH [online]. [cit. 2019-03-20]. Dostupné z: <https://freeswitch.org/>
- [6] Nejpoužívanější zkratky z oblasti počítačových sítí [online]. [cit. 2019-04-29]. Dostupné z: <http://www.businessit.cz/cz/zkratky-site-lan-wi-fi-wan-nat-dns-dhcp-dos-ssid-wps-wmm.php>
- [7] SIP header [online]. [cit. 2019-04-15]. Dostupné z: <https://www.3cx.com/blog/voip-howto/sip-invite-header-fields/>
- [8] SIP methods [online]. [cit. 2019-04-15]. Dostupné z: <https://www.3cx.com/pbx/sip-methods/>
- [9] SIP protocol [online]. [cit. 2019-04-15]. Dostupné z: <https://www.3cx.com/pbx/sip/>
- [10] SRTP Extension for DTLS [online]. [cit. 2019-04-15]. Dostupné z: <https://tools.ietf.org/html/rfc5764>
- [11] WebRTC - signaling protocol [online]. [cit. 2019-04-15]. Dostupné z: <https://bloggeek.me/signaling-protocol-webrtc/>
- [12] WebRTC Browser APIs and Protocols [online]. [cit. 2019-04-15]. Dostupné z: <https://hpbn.co/webrtc/>
- [13] WebRTC supported platforms and browsers [online]. [cit. 2019-04-15]. Dostupné z: <https://webrtc.org/>
- [14] WebRTC tutorial using SIPML5. Asterisk [online]. [cit. 2018-10-20]. Dostupné z: <https://wiki.asterisk.org/wiki/display/AST/WebRTC+tutorial+using+SIPML5>
- [15] WebSocket protocol [online]. [cit. 2019-04-15]. Dostupné z: <https://tools.ietf.org/html/rfc6455>

- [16] What-is-XMPP-and-how-does-it-work [online]. [cit. 2019-04-15].
Dostupné z: <https://www.quora.com/What-is-XMPP-and-how-does-it-work>
- [17] WebRTC. FreeSWITCH [online]. [cit. 2019-02-05]. Dostupné z:
<https://freeswitch.org/confluence/display/FREESWITCH/WebRTC>

Seznam příloh

<i>Příloha A:</i>	<i>Instalace Apache a konfigurace Apache serveru.....</i>	<i>LX</i>
<i>Příloha B:</i>	<i>Instalace a konfigurace pobočkové ústředny Asterisk</i>	<i>lxiv</i>
<i>Příloha C:</i>	<i>Instalace pobočkové ústředny FreeSWITCH.....</i>	<i>lxx</i>
<i>Příloha D:</i>	<i>Integrace WebRTC klienta</i>	<i>lxxiii</i>
<i>Příloha E:</i>	<i>Přípravení webového prohlížeče a uskutečnění hovoru.....</i>	<i>lxxv</i>

Příloha A: *Instalace Apache a konfigurace Apache serveru*

V této příloze je popsán postup pro instalaci serveru Apache, které využívá vlastně podepsané certifikáty vygenerovány pomocí OpenSSL. Výsledkem dodržení níže uvedených kroků bude plně funkční Apache server, který bude dále využíván pro integraci WebRTC klienta. Postup integrace WebRTC klienta je uveden v příloze D.

Instalace Apache serveru

Před samotnou instalací doporučuji stáhnout aktualizace balíčků, které operační systém využívá.

```
sudo apt-get update
```

Pokračujeme instalací samotného Apache serveru.

```
sudo apt-get install apache2
```

Po dokončení instalace upravíme soubor `/etc/apache2/apache2.conf` a to konkrétně položku `ServerName`. Zde můžeme doplnit doménové jméno, nebo IP adresu našeho serveru. Na konec souboru tedy doplníme tento řádek:

```
ServerName <server_domain_or_IP>
```

Aby se nám změny projeví je zapotřebí provést restart Apache serveru.

```
sudo service apache2 restart
```

Nyní provedeme kontrolu funkčnosti Apache serveru. Spustíme si webový prohlížeč a zadáme adresu `http://<your_server_FQDN_or_IP_address>`. Měla by se zobrazit úvodní stránka Apache2 web serveru.

Vygenerování SSL certifikátu

SSL kombinuje veřejný certifikát a soukromý klíč. Soukromý klíč se využívá k šifrování obsahu zaslaného klientům. Klíč je uložen na straně serveru a je nutností ho uchovat v tajnosti. Veřejný certifikát jak už z názvu vypovídá je veřejně sdílený s klienty, kteří využívají služby serveru. Obsahem veřejného certifikátu je také veřejný klíč, který používá klient pro dešifrování obsahu podepsaného tajným klíčem.

Pro vytvoření vlastně podepsaného certifikátu se soukromým klíčem použijeme nástroj OpenSSL. Vytvoření provedeme tímto příkazem:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout  
etc/ssl/private/apache.key -out /etc/ssl/certs/apache.crt
```

Pro porozumění výše uvedenému příkazu zde uvádím význam jednotlivých parametrů.

- `openssl` - základní nástroj pro vytvoření a správu klíčů a certifikátů
- `req` - podřízený příkaz kterým specifikujeme použití standardu X.509 pro vytvoření infrastruktury, dle níž se vytvoří dvojice soukromého a veřejného klíče;
- `-x509` - rozvíjí podřízený příkaz `req` a specifikuje, že chceme vytvořit podepsaný certifikát namísto generování požadavku na podepsání certifikátu

- `-nodes` - podřízený příkaz, kterým říkáme, že chceme přeskočit přístupovou fázi. Ve výsledku nám zruší vynucení ověření pomocí hesla při restartování Apache serveru.
- `-days 365` - definuje dobu platnosti vygenerovaného certifikátu
- `-newkey rsa:1024` - určuje, že chceme spolu s certifikátem vytvořit i nový klíč potřebný pro podepsání certifikátu. Tento klíč bude vytvořen pomocí kryptografického algoritmu RSA a bude mít délku 1024 bitů;
- `-keyout` - podřízený příkaz, kterým specifikujeme cestu, kam se umístí vytvořený soukromý klíč
- `-out` - podřízeným příkaz, kterým specifikujeme, kam se umístí vytvořený certifikát

Po zadání příkazu bude nutné zodpovědět otázky, které se týkají našeho serveru. Otázky a příklady odpovědí by mohly vypadat takto:

```
Country Name (2 letter code) [AU]:CZ
State or Province Name (full name) [Some-State]:Czech republic
Locality Name (eg, city) :Ostrava
Organization Name (eg, company) [Internet Widgits Pty Ltd]:VSB
Organization Unit Name (eg, section) []:VSB
Common Name (e.g. server FQDN or YOUR name) []:<server_IP_address>
Email Address []:admin@<your_domain>.com
```

Po zodpovězení otázek budou oba soubory vytvořeny v zadaných cestách. Certifikát v adresáři `/etc/ssl/certs` pod názvem `apache.crt` a soukromý klíč s názvem `apache.key` v adresáři `/etc/ssl/private/`.

Dále vytvoříme skupinu parametrů, která bude použita pro vytvoření Diffie-Hellman algoritmu. Tento algoritmus se používá při vyjednávání parametrů během navazování spojení.

```
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 1024
```

Konfigurace Apache serveru pro použití vlastně podepsaného certifikátu

Ze všeho nejdříve je zapotřebí vytvořit konfigurační soubor, ve kterém budeme definovat nastavení protokolu SSL. Soubor vytvoříme v adresáři `/etc/apache2/conf-available/`. Vytvořený soubor pojmenujeme `ssl-param.conf`.

Obsahem souboru definujeme, jak bude pro protokol SSL použit náš certifikát. Zkopírujte níže uvedené řádky do souboru `ssl-param.conf`. Nutno podotknout, že níže uvedená konfigurace pochází ze serveru `https://cipherli.st/`, kde je volně přístupná konfigurace nejen pro Apache, ale i pro nginx a Lighttpd.

Dalším krokem bude tedy zkopírování níže uvedených řádků do souboru `ssl-param.conf`.

```
SSLCipherSuite ECDH+AESGCM:EDH+AESGCM:AES256+ECDH:AES256+EDH
SSLProtocol All -SSLv2 -SSLv3
SSLHonorCipherOrder On
```

Instalace Apache a konfigurace Apache serveru

```
includeSubdomains; preload"
Header always set Strict-Transport-Security "max-age=63072000;
includeSubdomains"
Header always set X-Frame-Options DENY
Header always set X-Content-Type-Options nosniff
SSLCompression off
SSLSessionTickets Off SSLUseStapling on
SSLStaplingCache "shmcb:logs/stapling-cache(150000)"
SSLOpenSSLConfCmd DHParameters "/etc/ssl/certs/dhparam.pem"
```

Následně upravíme soubor */etc/apache2/sites-available/default-ssl.conf*, tak aby využíval námi vytvořený certifikát a soukromý klíč.

```
echo "" > /etc/apache2/sites-available/default-ssl.conf
```

Do souboru přidáme níže uvedenou konfiguraci a následně doplníme k poli `ServerName` IP adresu či doménové jméno našeho serveru.

```
<IfModule mod_ssl.c>
<VirtualHost _default_:443>
ServerAdmin dvo0148@vsb.cz
ServerName 192.168.150.135
DocumentRoot /var/www/html
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
SSLEngine on
SSLCertificateFile      /etc/ssl/certs/apache.crt
SSLCertificateKeyFile  /etc/ssl/private/apache.key

<FilesMatch "\.(cgi|shtml|phtml|php)$">
SSLOptions +StdEnvVars
</FilesMatch>
<Directory /usr/lib/cgi-bin>
SSLOptions +StdEnvVars
</Directory>
BrowserMatch "MSIE [2-6]" \
nokeepalive ssl-unclean-shutdown \
downgrade-1.0 force-response-1.0
```

```
</VirtualHost>
```

```
</IfModule>
```

Aby se změny v nastavení projeví je nutno povolit SSL moduly a následně restartovat službu Apache.

```
sudo a2enmod ssl && a2enmod headers && a2ensite default-ssl &&  
a2enconf ssl-params && apache2ctl configtest sudo a2enmod  
headers
```

```
sudo service apache2 restart
```

Na závěr zkontrolujeme, jestli je služba Apache řádně spuštěna, tedy ve stavu running.

```
sudo service Apache2 status
```

Příloha B: *Instalace a konfigurace pobočkové ústředny Asterisk*

V této příloze je popsána instalace pobočkové ústředny Asterisk ve verzi 16 LTS. Po dodržení níže uvedených kroků bude výsledkem nainstalovaný WebRTC server, který bude obsahovat všechny potřebné predispozice, pro jeho následnou konfiguraci.

Instalace pobočkové ústředny Asterisk

Před začátkem samotné instalace se přepneme na uživatele s administrátorským oprávněním, tedy root.

```
sudo su
```

Nyní se přesuneme do složky, kde budeme Asterisk kompilovat.

```
cd /usr/src
```

Stáhneme Asterisk.

```
wget
http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-
16-current.tar.gz
```

Provedeme extrakci balíčku.

```
tar zxvf asterisk-16-current.tar.gz
```

Vstoupíme do složky z vyextrahovaného balíčku.

```
cd asterisk-16*
```

Před kompilací Asterisku ještě stáhneme a následně zkompilujeme a nainstalujeme PJPROJECT, což je balíček obsahující podporu PJSIP.

```
git clone git://github.com/asterisk/pjproject pjproject
cd pjproject
./configure --prefix=/usr --enable-shared --disable-sound
--disable-resample --disable-video --disable-opencore-amr
CFLAGS='-O2 -DNDEBUG'
make dep
make
make install
ldconfig
ldconfig -p |grep pj
cd ..
```

Nyní zkompilujeme Asterisk.

```
contrib/scripts/get_mp3_source.sh
contrib/scripts/install_prereq install
ITU-T telephone code: 420
```

Po zkompilování přistoupíme k základní konfiguraci Asterisku.

```
./configure
make menuselect
make
make install
make samples
make config
ldconfig
```

Nyní jsme připraveni spustit Asterisk.

```
/etc/init.d/asterisk start
```

Tvorba certifikátu pro protokol TLS

Pro zajištění kryptografického zabezpečení přenosu zpráv koncových bodů WebSocket spojení je nutností vytvořit certifikáty a klíče pro protokol TLS. Vytvoření certifikátů a klíčů pro protokol TLS je také podmínkou toho, aby bylo možno použít Secure WebSocket (wss).

Vytvoříme si složku, kde budeme vygenerované klíče uchovávat

```
mkdir /etc/asterisk/keys
```

Přejdeme do složky contrib/scripts, která obsahuje skript skript pro vytvoření vlastního podepsaného certifikátu.

```
cd /usr/src/asterisk-16*
cd /contrib/scripts
```

Spustíme skript, pomoci něhož vygenerujeme vlastní podepsaný certifikát.

```
./ast_tls_cert -C <your IP or Domain Name> -O "<name of your company>" d /etc/asterisk/keys
```

Po zavolání skriptu, budete několikrát vyzváni pro zadání hesla. První heslo je pro `/etc/asterisk/keys/ca.key`. Po zadání hesla dojde k vytvoření `/etc/asterisk/keys/ca.crt`. Po zadání druhého hesla dojde k vytvoření `/etc/asterisk/keys/asterisk.key`.

Soubor `/etc/asterisk/keys/asterisk.crt` se vygeneruje automaticky. Po třetím zadání hesla dojde k vytvoření souboru `/etc/asterisk/keys/asterisk.pem`. Tento soubor je výsledkem

kombinace souborů `asterisk.key` a `asterisk.crt`. Vytvořené certifikáty je poté možno najít v adresáři `/etc/asterisk/keys`.

Obsah složky si můžeme vypsát použitím příkazu

```
ls /etc/asterisk/keys
```

Složka by měla obsahovat tyto soubory:

```
asterisk.crt
asterisk.csr
asterisk.key
asterisk.pem
ca.cfg
ca.crt
ca.key
tmp.cfg
```

Konfigurace vestavěného HTTP serveru

Asterisk obsahuje integrovaný HTTP server. Na tomto serveru otevřeme port, na kterém následně Asterisk bude naslouchat a navazovat WebSocket spojení.

Přejdeme tedy do složky `etc/asterisk`

```
cd /ect/asterisk
```

Upravíme konfigurační soubor `http.conf` dle níže uvedeného vzoru:

```
[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlscipher=AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk/keys/asterisk.pem
```

Použitím této konfigurace zajistíme naslouchání `asterisk` procesu na portu 8088, povolili jsme použití protokolu TLS a také jsme specifikovali certifikát, který bude použit pro ověření komunikace.

Po provedení změn v souboru `http.conf` restartujeme modul HTTP z konzole asterisku

```
reload http
```

Jestli server naslouchá na specifikovaném portu, můžeme ověřit tímto příkazem, který je ovšem nutno spustit z konzole Asterisku

```
http show status
```

Měli byste dostat tento výstup:

```
HTTP Server Status:
```

```
Prefix:
```

```
Server: Asterisk/certified/13.13-cert4
```

```
Server Enabled and Bound to 0.0.0.0:8088
```

```
websocket_enabled=false
```

```
HTTPS Server Enabled and Bound to 0.0.0.0:8089
```

Konfigurace uživatelských účtů

PBX Asterisk využívá pro konfiguraci uživatelských účtů soubor `pjsip.conf`.

Pro definování transportní metody, kterou budou uživatelé využívat, upravte soubor `/etc/asterisk/pjsip.conf` dle následujícího postupu:

```
[transport_wss]
type=transport
protocol=wss
bind=0.0.0.0:5067
```

Po definování transportní metody, kterou jsme provedli v předešlém kroku, vytvoříme entitu v rámci `pjsip.conf`, která bude reprezentovat uživatelský účet koncového uživatele. Entita se bude skládat z objektů `Aor`, `auth` a `endpoint`. Pro konfiguraci entity rozšíříme `pjsip.conf` o následující řádky:

```
[1000]
type=aor
max_contacts=1
remove_existing=yes
```

```
[1000]
type=auth
```

```
auth_type=userpass
username=1000
password=1000
```

Přidáním těchto řádků jsme zajistili, že entita bude známá jako 1000 a pro ověření bude používat heslo 1000.

Dalším krokem bude vytvoření koncového bodu, který bude odkazovat na objekty aor a auth jako na své konfigurační parametry. Pro konfiguraci koncového bodu doplníme soubor pjsip.conf o následující řádky:

```
[1000]
type=endpoint
transport=transport_wss
aors=1000
auth=1000
use_avpf=yes
media_encryption=dtls
dtls_ca_file=/etc/asterisk/keys/ca.crt
dtls_cert_file=/etc/asterisk/keys/asterisk.pem
dtls_verify=fingerprint
dtls_setup=actpass
ice_support=yes
media_use_received_transport=yes
rtcp_mux=yes
context=default
disallow=all
allow=alaw, VP8
```

Stejným způsobem nakonfigurujeme entity pro účet 2000.

Konfigurace dialplánu

Aby bylo možno provést hovor, je zapotřebí vytvořit směrovací plán.

Níže uvedené řádky definují kontext „default“, upravte soubor `/etc/asterisk/extensions.conf` dle níže uvedeného vzoru:

```
[default]
exten=> 200,1,Answer()
exten=> 200,2,NoOp(Dovolali jste se na ustrednu)
exten=> 200,3,Playback(demo-congrats)
exten=> 200,4,Hangup()
exten =>1000,1,Dial(PJSIP/1000)
exten =>2000,1,Dial(PJSIP/2000)
```

Dodržením výše uvedených kroků, jsme provedli instalaci a konfiguraci PBX Asterisk, která nyní umožní spojení skrze WebSocket s WebRTC klientem. Aby se projevíly změny, které jsme konfigurací způsobili, je nutné provést restartování modulu dialplan:

```
dialplan reload
```

Pro uskutečnění testovacího hovoru je mít zapotřebí integrovaný WebRTC klient. Touto problematikou se zabývá příloha D.

Příloha C: *Instalace pobočkové ústředny FreeSWITCH*

V této příloze je popsán postup pro instalaci pobočkové ústředny FreeSWITCH. Výsledkem dodržení níže uvedených kroků bude plně funkční pobočková ústředna FreeSWITCH se všemi potřebnými predispozicemi pro následnou konfiguraci.

Instalace pobočkové ústředny FreeSWITCH

Přepneme se na uživatele s právy root.

```
sudo su
```

Pokračujeme instalací balíčků nezbytných pro úspěšnou kompilaci pobočkové ústředny FreeSwitch. Zároveň také nainstalujeme programy, které budeme dále používat.

```
apt-get -y install autoconf automake devscripts gawk g++  
libjpeg-dev libncurses5-dev liblua5.2-dev lua-sec lua-socket git  
libtool make python-dev gawk pkg-config libtiff-dev libperl-dev  
libgdbm-dev libdb-dev gettext libssl-dev libcurl4-openssl-dev  
libpcre3-dev libspeex-dev libspeexdsp-dev libsqlite3-dev  
libedit-dev libldns-dev libpq-dev memcached libmemcached-dev  
libopus-dev vim tshark curl subversion libsndfile-dev
```

Po dokončení instalace si stáhneme ze stránek yasm.tortall.net/Download.html poslední verzi modulárního assembler projektu yasm, který poté vyextrahujeme a nainstalujeme.

```
tar -zxvf yasm-1.3.0.tar.gz
```

```
cd yasm*
```

```
./configure
```

```
make
```

```
make install
```

Přejdeme do složky, kde budeme FreeSWITCH kompilovat a následně stáhneme FreeSWITCH.

```
cd /usr/src/
```

```
wget -c files.freeswitch.org/freeswitch-releases/freeswitch-  
1.8.5.tar.gz
```

Po úspěšném stažení vyextrahujeme balíček.

```
tar -zxvf freeswitch-1.8.5.tar.gz
```

Přejdeme do složky z vyextrahovaného balíčku

```
cd freeswitch-1.8.5
```

Zkompilujeme FreeSWITCH. U kompilace může nastat problém v podobě chybějících balíčků v takovém případě je nutností balíček nainstalovat a spustit proces kompilace znovu.

```
./configure
make
make install
make cd-sounds-install
make cd-moh-install
```

Nyní spustíme FreeSwitch

```
cd /usr/local/freeswitch/bin
./freeswitch
```

Generování certifikátů pro protokol TLS

Pokud chceme využívat komunikace skrze Secure WebSocket, musíme si vygenerovat vlastní certifikát. Pokud ovšem chceme zprovoznit komunikaci pouze skrze nezabezpečený WebSocket, postačí nám dvojice základních certifikátů, které vzniknout automaticky při kompilaci pobočkové ústředny FreeSWITCH. Jedná se o soubory „dtls-srtp.pem“ a „wss.pem“, které nalezneme v adresáři /usr/local/freeswitch/certs/.

Přesuneme se do adresáře /usr/local/freeswitch/certs

```
cd /usr/local/freeswitch/certs
```

Smažeme původní certifikát wss.pem

```
sudo rm -f wss.pem
```

Jelikož FreeSWITCH neobsahuje skript na tvorbu certifikátů, využijeme skript, který jsme použili v případě pobočkové ústředny Asterisk. Skript je přiložený v příloze v této práci pod názvem gen_crt. Skript je v podstatě stejný, jen jsou upraveny názvy vygenerovaných výstupních souborů.

Spustíme skript.

```
./gen_ssl_crt -C <your IP or Domain Name> -O "<name of your company>" -d /usr/local/freeswitch/certs
```

Certifikáty zkontrolujeme v cílovém adresáři /usr/local/freeswitch/certs

```
ls /usr/local/freeswitch/certs
```

Obsah adresáře by měl být následující:

```
wss.crt
wss.csr
```

```
wss.key  
wss.pem  
ca.cfg  
ca.crt  
ca.key  
tmp.cfg
```

Konfigurace modulu Sofia

Pro zajištění podpory WebRTC je nutností povolit modulu Sofia naslouchání WebSocket spojení na určitém portu. Ke konfiguraci modulu Sofia slouží konfigurační soubor soubor `/usr/local/freeswitch/conf/sip_profiles/internal.xml`.

Doplňte tyto řádky:

```
<param name="tls-cert-dir" value="/usr/local/freeswitch/certs"/>  
<param name="wss-binding" value=":7443"/>
```

Nyní je zapotřebí načíst znovu konfigurační soubory, to se provede z konzole FreeSWITCH tímto příkazem:

```
Reloadxml
```

Ověření provedeme pomocí příkazu `sofia status profile internal`. Ve výstupu tohoto příkazu byste měli dohledat tento parametr:

```
WSS-BIND-URL    sips:mod_sofia@<freeswitch_server_IP>:7443;transport=wss
```

Výsledkem dodržení všech předchozích kroků je nainstalovaná a nakonfigurovaná pobočková ústředna PBX FreeSWITCH. Pro provedení testovacího hovoru je zapotřebí mít integrovaného WebRTC klienta. Postup pro integraci WebRTC klienta je uveden v příloze D.

Příloha D: *Integrace WebRTC klienta*

V této příloze je popsán postup pro integraci WebRTC klienta. V současné době lze na Internetu najít mnoho volně dostupným WebRTC klientů, já si vybral klienta Sipml5 z důvodu přehledné dokumentace a širokého zázemí v podobě aktivní komunity. Predispozicí pro úspěšnou integraci WebRTC klienta je mít nainstalován Apache server s vlastně podepsanými SSL certifikáty. Postup pro instalaci Apache serveru je popsán v příloze A. Po dodržení níže uvedených kroků bude úspěšně integrovaný WebRTC klient.

Ze všeho nejdříve stáhneme z webových stránek <http://www.doubango.org/sipml5> zdrojové kódy Sipml5 klienta. Stáhnutý .zip soubor vyextrahujeme.

Přesuneme se do složky sipml5-master.

```
cd sipml5-master
```

V adresáři vymažeme soubor index.html.

```
rm index.html
```

Následně přejmenujeme soubor call.html na index.html.

```
mv call.html index.html
```

Nyní všechny soubory přesuneme do složky /var/www/html/, kde zároveň nahradíme původní soubor index.html.

Abychom zajistili projevení všechny změn, je zapotřebí restartovat Apache server.

```
sudo service apache2 restart
```

Nyní spustíme webový prohlížeč a zadáme adresu našeho Apache serveru `https://<FQDN_or_server_IP>`. Můžeme si povšimnout upozornění, které říká, že náš Apache server nevyužívá certifikát, který je podepsaný od důvěryhodné certifikační autority. Abychom mohli přistoupit na náš Apache server, je zapotřebí přidat výjimku. Po přidání výjimky se nám zobrazí registrační stránka Sipml5 klienta.

Registration

Display Name:

Private Identity:

Public Identity:

Password:

Realm:

* Mandatory Field

Video enabled

Call control

Obrázek D.1: Rozhraní klienta Sipml5

Příprava webového prohlížeče

Tento návod je konkrétně uveden pro pobočkovou ústřednu Asterisk, postup je téměř totožný s postupem pro pobočkovou ústřednu FreeSWITCH.

Přejdeme do adresáře, kde se nachází Asteriskem vygenerované klíče a certifikáty

```
cd /etc/asterisk/keys
```

V tomto adresáři použijeme příkaz nástroje OpenSSL pro konverzi mezi formáty

```
openssl pkcs12 -export -out <název_výstupního_certifikátu>.p12 -  
inkey asterisk.key -in asterisk.crt -certfile ca.crt
```

Po zadání příkazu budete vyzváni k zadání hesla, které bude součástí námi vytvořeného certifikátu. Zadejte heslo, které jste nastavili při generování certifikátu.

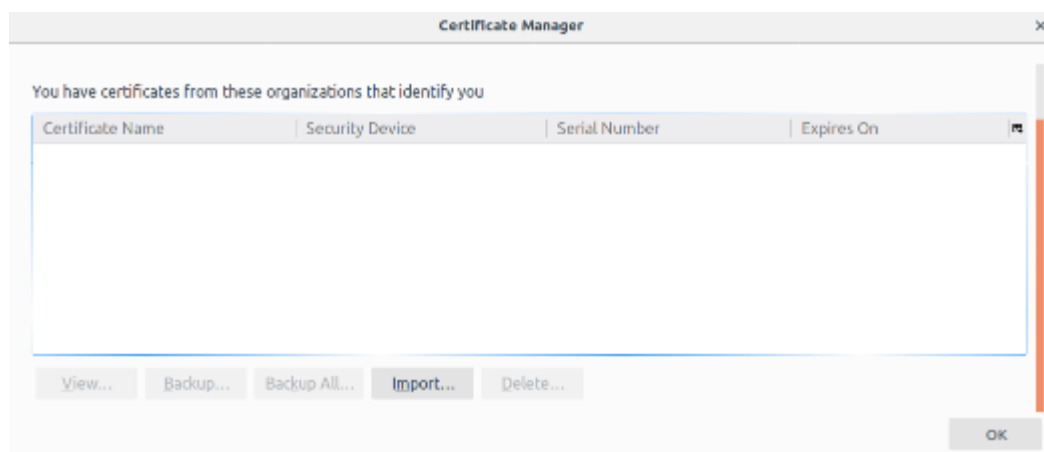
Ve složce `/etc/asterisk/keys` byste měli najít nově vytvořený certifikát `<název_výstupního_certifikátu>.p12`

Importování certifikátu do webového prohlížeče lze provést v nastavení prohlížeče. Níže uvedený postup je popsán pro webový prohlížeč Mozilla Firefox, který je na Ubuntu, který jsem použil pro testování komunikace již defaultně nainstalován.

V pravém horním rohu otevřeme nastavení a přejdeme do nastavení certifikátů:

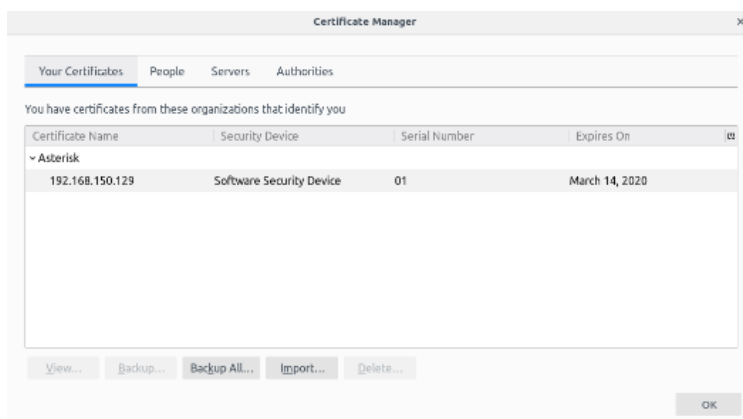
Settings -> Preference -> Privacy and security -> View Certificates

Po stisknutí nabídky view certificates se nám zobrazí okénko s přehledem vložených certifikátů.



Obrázek E.1: Manažer certifikátů

V manažeru certifikátů klikneme na tlačítko import a vyhledáme náš certifikát ve formátu .p12. Po vložení certifikátu se nám zobrazí náš certifikát v certifikačním manažeru.



Obrázek E.2: Vložený certifikát

Přejdeme na stránku https://<IP_Asterisk_serveru>:8089/ws a zde přidělíme tomuto spojení výjimku.

Přejdeme na stránky našeho Apache serveru. WebRTC klienta nakonfigurujeme dle následujícího postupu, který je uveden pro linku 1000. V kolonce „Registration“ vyplníme následující údaje:

- Display Name: 1000
- Private Identity*: 1000
- Public Identity*: sip:1000@<IP_Asterisk_serveru>
- Password: 1000
- Realm*: <IP_Asterisk_serveru>

Poté přejdeme do expert módu, kliknutím na tlačítko „Expert mode“ a specifikujeme náš WebSocket pro transport. Kolonku „WebSocket Server URL“ vyplňte následujícím způsobem:

- WebSocket Server URL: wss://<IP_Asterisk_serveru>:8089/ws

Nastavení uložíme kliknutím na tlačítko „Save“ a vrátíme se na stránku s oknem registrace. Při stisknutí tlačítka „LogIn“, byste měli být nahoře informováni o úspěšném zaregistrování statusem „Connected“

Stejným způsobem zaregistrujeme z jiného zařízení WebRTC klienta k účtu 2000. A provedeme testovací video-hovor mezi klienty 1000 a 2000, pomoci ovládacího okna Sipml5 klienta.

Otevřeme nabídku „Call“ a vybereme „Video“. Budeme dotázáni, zda chceme dané stránce povolit přístup k mikrofonu. Povolení přidělíme kliknutím na tlačítko „Allow“.

Následně byste měli v ovládacím okně Sipml5 klienta vidět status „Call in progress“. Při úspěšném sestavení SIP komunikace, bude status odpovídat stavu „In Call“. V tomto stavu již dochází k RTP přenosu.