UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

Estudo sobre o desenvolvimento de aplicações VOIP para plataforma Windows com criptografia

Davi de Araújo dos Santos
Rodrigo Ramos de Souza

Orientador
Geiza Maria Hamazaki da Silva

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2015

Estudo sobre o desenvolvimento de aplicações VOIP para plataforma Windows com a utilização de criptografia

Davi de Araújo dos Santos
Rodrigo Ramos de Souza

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada por:

_____
Prof.ª. Drª. Geiza Maria Hamazaki da Silva (UNIRIO)

_____
Prof. Dr. Sidney Cunha, de Lucena (UNIRIO)

_____
Ronaldo Cesar dos Santos (Analista Sênior/CASNAV)

RIO DE JANEIRO, RJ – BRASIL.
DEZEMBRO DE 2015

Agradecimentos

Agradeço à minha família que sempre me apoiou.

Aos amigos que participaram dessa jornada comigo.

À Unirio que me ensinou muito.

Rodrigo Ramos

A Deus, pela saúde, proteção, sabedoria e todas as conquistas alcançadas.

À minha família. Meus pais, José Rodrigues e Marlina, por todo apoio e incentivo em todos os momentos da graduação. Companhia fundamental para o sucesso.

Em especial ao meu irmão e mestre, Igor de Araújo, que tanto me ajudou, me cobrou e me manteve motivado e focado durante toda a graduação, do primeiro período até o projeto final. Meu grande líder nesse período. Família é a base de tudo e só tenho a agradecer.

Àqueles colegas de turma, que hoje são meus amigos e estiveram comigo durante essa jornada, tanto nos momentos sérios quanto nos de lazer e diversão.

À orientadora Geiza Hamazaki, pela paciência e tranquilidade com que nos conduziu durante todo o projeto e também como mentora da turma na graduação.

Ao Rodrigo Ramos, pelo projeto final e trabalhos realizados desde o começo da faculdade.

Aos professores e funcionários do CCET/UNIRIO pela excelência e qualidade no ensino e serviços prestados.

Àqueles familiares, amigos e colegas de trabalho que de alguma maneira estiveram comigo e me ajudaram na graduação e no meu crescimento profissional.

Muito obrigado!

Davi de Araújo dos Santos

RESUMO

Esta monografia apresenta um estudo sobre o desenvolvimento de aplicações VoIP para a realização de chamadas entre dispositivos com o sistema operacional Windows, garantindo a segurança da conversa através da cifração da voz dos participantes. Será apresentada a base teórica por trás das tecnologias utilizadas e o processo de desenvolvimento de uma aplicação VoIP com criptografia de voz, utilizando software livre.

Palavras-chave: VoIP; Criptografia; Software livre.

ABSTRACT

This work presents a study on the development of VoIP applications allowing safe conversations between devices through the use of cryptography to encipher the participants' voice message. Here it will be presented the theory behind these technologies used and the development process of a VoIP application with voice cryptography, making use of free software.

Keywords: VoIP; Cryptography; Free Software.

Índice

1 - INTRODUÇÃO

O crescente avanço da tecnologia e informatização dos processos gerou novas tendências na Gestão Corporativa em vigor em todo o mundo. É comum que assuntos importantes sejam resolvidos em chamadas telefônicas ou de vídeo, e com as descobertas sobre espionagem na rede, a segurança da informação trocada entre o remetente e os destinatários se torna de extrema importância.

O acesso à internet tem aumentado no Brasil [1] e cerca de 95 milhões de brasileiros acessa a internet [2] através de computadores e dispositivos móveis, utilizando aplicativos para troca de mensagens de texto e realização de chamadas VoIP, que dizem garantir a segurança na troca de mensagens. O Skype, por exemplo afirma usar criptografia em chamadas de **Skype** para **Skype**, mas não tem nenhuma proteção em chamadas para telefones móveis e fixos. Outros aplicativos populares como **WhatsApp** e **Facebook Messenger** não oferecem nenhum tipo de segurança ou o fazem de forma limitada. O **WhatsApp** só usa criptografia nas mensagens enviadas entre dispositivos que usam o sistema android [3].

Entre as aplicações que realizam chamadas de forma segura, em sua maioria, apresentam soluções fechadas que não permitem customizações, como por exemplo a inserção de um algoritmo criptográfico pelo desenvolvedor/usuário.

Dispositivos que suportam VoIP são vulneráveis a vária formas de ataque comuns a redes. Um exemplo é o *IP Spoofing*, onde um atacante se faz passar pelo remetente ou destinatário da chamada utilizando um endereço IP falsificado [34]. Em outro exemplo, o tráfego pode ser interceptado com propósitos maliciosos em redes desprotegidas. Os pacotes que transitam pela rede são capturados e podem ser vistos por atacantes. Conversas em tempo real podem ser ouvidas dessa forma (*Sniffing*) [35]. Uma chamada criptografada pode ser capturada, mas o atacante não será capaz de compreender o que está sendo enviado.

Este cenário motivou do desenvolvimento desta aplicação, onde a segurança da chamada é realizada através da criptografia de voz, que transforma a voz de forma que esta só possa ser compreendida pelo destinatário, e da identificação dos remetentes e destinatários, além da proteção contra modificação dos dados enviados pela rede. Neste desenvolvimento são necessários conhecimentos nas áreas de redes, criptografia e manipulação de voz.

1.1 Objetivos

No projeto é apresentado o estudo sobre o desenvolvimento de aplicações VoIP (voz sobre IP) para a realização de chamadas entre dispositivos, garantindo a identidade do remetente e do destinatário e a segurança da conversa através da cifração simétrica (AES) da voz dos participantes. Esta aplicação utiliza recursos *open source*, como Linux, Kamailio [24], PJSIP [19] e OpenSSL [21].

1.2 Organização do Texto

O presente trabalho está estruturado em capítulos e, além desta introdução é desenvolvido da seguinte forma:

- Capítulo 2: Estado da arte - Explicação teórica das tecnologias usada no projeto, como criptografia, VoIP e codificação de voz.
- Capítulo 3: O Sistema - Apresenta as ferramentas usadas no projeto e explica o processo de desenvolvimento e as dificuldades encontradas.
- Capítulo 4: Conclusões e Trabalhos futuros - Reúne as considerações finais, assinala as contribuições da pesquisa sugerindo possibilidades de continuação deste trabalho.

2 - ESTADO DA ARTE

Para o desenvolvimento de uma aplicação VoIP com criptografia foi necessário pesquisar sobre essa técnica que permite codificar mensagens de modo que esta possa ser lida somente pelo remetente e pelo destinatário. A troca de mensagens ocorre através da internet com protocolos para envio de dados que facilitam as chamadas de voz. Também é necessário capturar a voz e transformá-la de analógica para digital antes do envio. Neste capítulo é apresentada uma explicação teórica sobre as tecnologias utilizadas no projeto.

2.1 Criptografia

Criptografia é o estudo dos princípios e técnicas pelas quais uma mensagem pode ser codificada de forma a ser lida apenas pelo destinatário final. Este é um dos principais mecanismos de segurança da informação para a proteção dos riscos associados ao uso da Internet.

A história da criptografia é muito antiga, ela foi utilizada no Egito Antigo, em 1900 AC, onde um escriba usou hieróglifos fora do padrão numa inscrição. Em 600 AC, os Hebreus usavam cifras de substituição monoalfabéticas (substituição simples) [4]. Na época do Império Romano surgiu a Cifra de César, uma técnica de substituição que troca as letras do alfabeto avançando um número fixo de vezes.

Texto simples	a	b	c	d	e	f	g	h	i	j	k	l	m
Cifra	D	E	F	G	H	I	J	K	L	M	N	O	P
Texto simples	n	o	p	q	r	s	t	u	v	w	x	y	z
Cifra	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

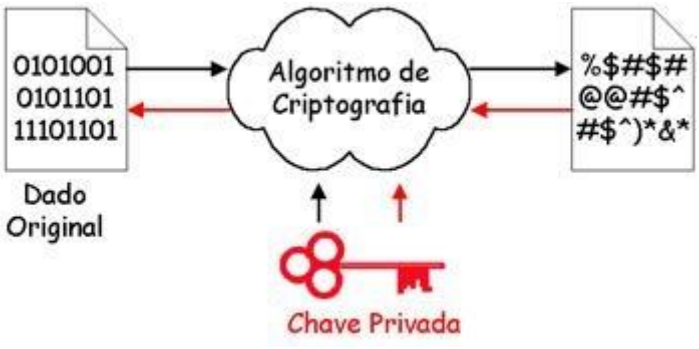
Figura 1 - Substituição original na cifra de César com deslocamento 3

Durante a segunda guerra mundial, o exército alemão utilizou a máquina eletromecânica decifração e decifração, **Enigma**, para a troca de mensagens entre suas tropas. A criptografia da Enigma foi decodificada graças aos esforços de uma equipe liderada por Alan Turing no uso da probabilidade em criptoanálise. Eles desenvolveram o primeiro computador digital programável, Colossus, que foi capaz de decifrar a máquina Enigma. Em 1948, foi desenvolvida por Claude Shannon, a Teoria Matemática da Comunicação (Teoria da Informação), que permitiu grandes avanços nos campos da criptoanálise e criptografia [5]. Em 1976 foi desenvolvido um sistema de

criptografia de chave pública que mais tarde viria a ser aperfeiçoado por pesquisadores do MIT dando origem ao algoritmo RSA.

Atualmente, com os dados digitais, o processo de criptografia é aplicado aos bits por meio de algoritmos e métodos criptográficos, como por exemplo: chave simétrica, chave assimétrica e função hash.

A **criptografia simétrica** faz uso de uma mesma chave, tanto para cifrar quanto para decifrar uma mensagem [6]. Esse método é útil em casos nos quais a informação é cifrada e decifrada por uma mesma pessoa. Sua principal vantagem é a alta velocidade de codificação e decodificação, porém esse método possui a desvantagem de necessitar da mesma chave para cifrar e decifrar a mensagem. Se houver a necessidade de troca de informação com pessoas ou equipamentos diferentes, torna-se necessário que essa chave seja enviada por um canal seguro para não comprometer a segurança da informação cifrada. Alguns métodos criptográficos que usam chave simétrica são: AES, Blowfish, RC4, 3DES, e IDEA [7][8].



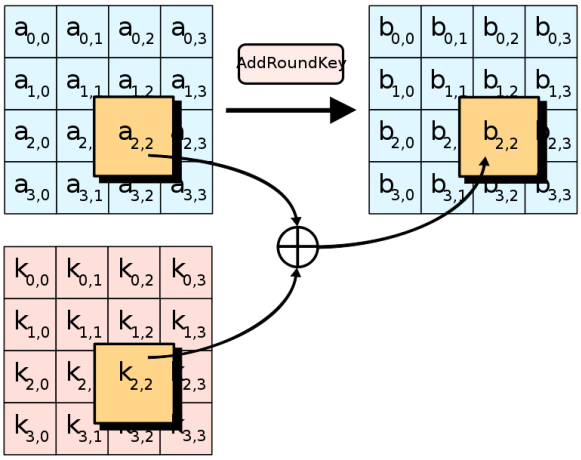
http://www.gta.ufrj.br/grad/07_1/ass-dig/TiposdeCriptografia.html

Figura 2 - Criptografia de chave simétrica

Um exemplo de algoritmo de criptografia simétrica é o AES (*Advanced Encryption Standard*) é uma cifra que opera em blocos de tamanho fixo (128 bits ou 16 bytes) e chaves de 128, 196 ou 256 bits. O algoritmo recebe como entrada um bloco de 128 bits (a mensagem), uma chave do tamanho escolhido e devolve uma saída de 128 bits (a mensagem cifrada). O AES opera

a partir de uma matriz de bytes com 4x4 posições chamado de estado. O processo de cifragem do AES ocorre em 4 estágios [48]:

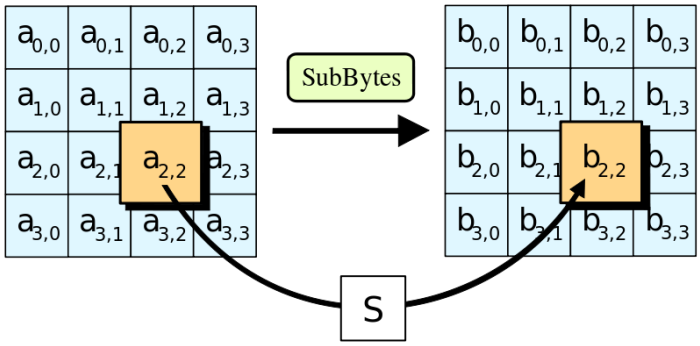
1. *AddRoundKey* - em cada round uma subchave é derivada da chave no mesmo tamanho do estado e combinada com cada byte do estado através de uma operação XOR bit a bit (figura 3).



https://en.wikipedia.org/wiki/File:AES-AddRoundKey.svg

Figura 3 - Etapa addRoundKey

2. *Subtypes* - ocorre a substituição não linear de cada byte na matriz através do *Rijndael S-Box* [49] (figura 4).

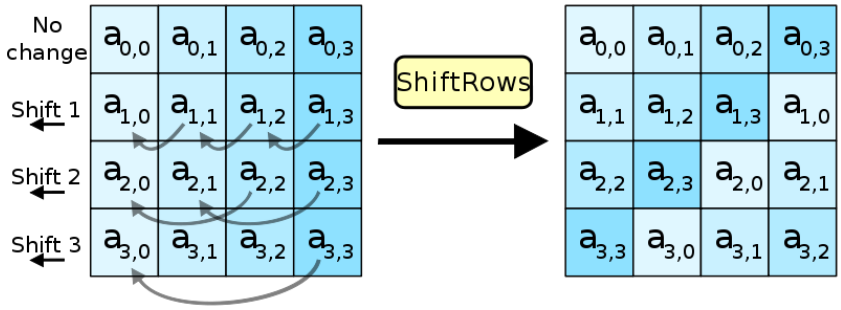


https://en.wikipedia.org/wiki/File:AES-SubBytes.svg

Figura 4 - Etapa Subtypes

3. *ShiftRows* - desloca os bytes referentes àquela linha de estados num determinado número de posições. Para os blocos de 128 e 192 bits o deslocamento é o mesmo

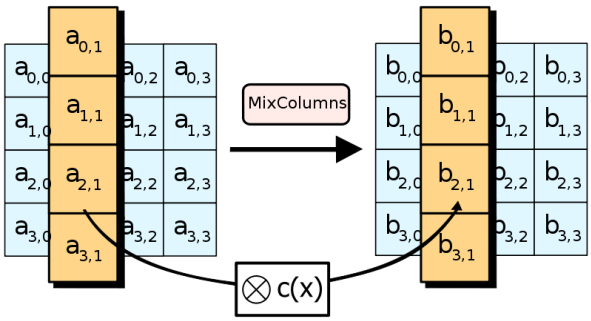
onde a primeira linha não é modificada e para as demais linhas, os bytes são deslocados n-1 linhas para a esquerda (figura 5).



https://en.wikipedia.org/wiki/File:AES-ShiftRows.svg

Figura 5- Etapa ShiftRows

4. *MixColumns* - é realizada a operação de mesclagem das colunas de estado onde os 4 bytes de cada coluna são combinados através de uma transformação linear invertível com o *ShiftRows*. Sua função utiliza os 4 bytes como entrada e gera 4 bytes de saída. Durante essa operação cada coluna é multiplicada por uma matriz interna pré-determinada cuja multiplicação pelos valores em cada bloco indica o deslocamento a esquerda a ser realizado. Em seguida é efetuada uma operação XOR do valor de deslocamento com os valores iniciais não deslocados (figura 6). No último turno do processo de cifragem, ao invés de termos um último estágio de *MixColumns*, teremos um estágio *AddRoundKey*.

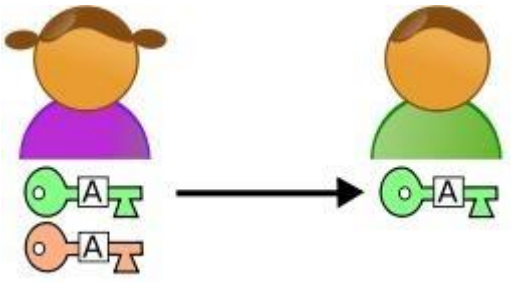


https://en.wikipedia.org/wiki/File:AES-MixColumns.svg

Figura 6 - Etapa MixColumns

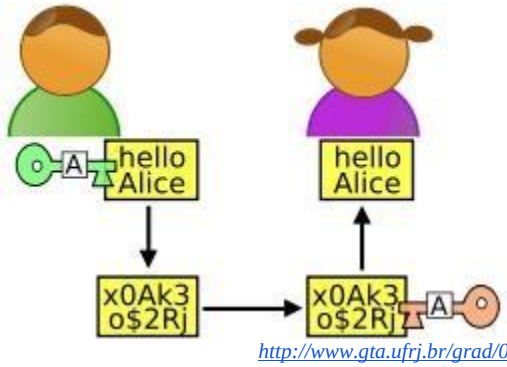
O processo inverso ocorre para decifrar a mensagem. Se a chave estiver correta, a mensagem é decifrada e a saída será idêntica a mensagem original.

Na **criptografia assimétrica** ou de **chave pública** [5], são usados dois tipos diferentes de chaves, a chave pública e a chave privada (figura 7a e 7b). A chave pública pode ser conhecida por todos, esta é usada para cifrar mensagens, e a chave privada, utilizada para decifrar as mensagens, deve ser mantida em sigilo pelo seu proprietário. Um dos algoritmos de criptografia assimétrica, RSA, gera as chaves através do produto de dois números primos. Quando se busca a confidencialidade, utiliza-se a chave pública para cifrar a mensagem e a chave privada para decifrá-la [8]. Por outro lado, se o objetivo é garantir a autenticidade, a mensagem é cifrada com a chave privada do remetente e decifrada com a chave pública do mesmo. Métodos criptográficos que usam chaves assimétricas são: RSA, DAS, ECC e Diffie-Hellman [8].



http://www.gta.ufrj.br/grad/07_1/ass-dig/TiposdeCriptografia.html

Figura 7a - A chave verde representa chave pública e a rosa, a chave privada



http://www.gta.ufrj.br/grad/07_1/ass-dig/TiposdeCriptografia.html

Figura 7b - Envio de mensagem usando criptografia assimétrica

Para grandes volumes de dados a criptografia simétrica é a mais indicada dada sua velocidade. A criptografia assimétrica por sua vez, dispensa a necessidade de um canal seguro para o compartilhamento de chaves, mas os algoritmos têm um alto custo computacional tornando-o mais lento que a criptografia simétrica. Dessa maneira, a forma mais vantajosa de uso desses métodos é combiná-los, onde a criptografia de chave simétrica é usada para a cifração da informação e a criptografia assimétrica para o compartilhamento da chave simétrica [36].

Um outro método criptográfico é a **função Hash**, uma função matemática que, quando aplicada a uma informação gera um resultado único e de tamanho fixo independentemente da entrada (geralmente chamada de mensagem). Essa saída pode ser em tamanhos que variam de 128 a 256 bits [9]. Ela permite verificar facilmente se alguma mensagem resulta num dado valor hash, mas se a mensagem é desconhecida, é muito difícil de reconstruí-la com os valores hash conhecidos, e por isso é considerada praticamente impossível de reverter. A função hash ideal possui 4 propriedades básicas:

- É rápido computar o valor hash para qualquer mensagem;
- Não é possível gerar uma mensagem a partir da saída;
- Não é possível modificar a mensagem sem modificar a saída;
- Não é possível encontrar duas mensagens diferentes com o mesmo hash.

Este método é usado para fazer verificações de integridade de um arquivo, geração de assinaturas digitais e verificação de senhas.

Assinatura digital é uma técnica utilizada para identificar e comprovar o remetente de uma mensagem. Ela baseia-se no fato de que somente o dono do par de chaves tem acesso à chave privada e, dessa maneira, a verificação é realizada utilizando a chave pública. Nesse método, um hash é gerado a partir da mensagem e, junto com a mensagem, é criptografado usando a chave

privada do remetente. Para verificar a autenticidade do documento, um novo hash deve ser gerado a partir da mensagem e comparado com o hash original que foi enviado. Se os *hashs* forem iguais, a mensagem está íntegra [10].

O **certificado digital** é um documento eletrônico assinado por uma Autoridade Certificadora (AC), e que contém informações referentes ao emissor e a uma pessoa ou entidade para a qual o certificado foi emitido. Sua função é a de vincular uma pessoa ou entidade a uma chave pública. Pode ser emitido para pessoas, empresas, equipamentos ou até um website, e ser homologado para diferentes usos, como confidencialidade e assinatura digital. As principais informações que constam em um certificado digital são: chave pública do titular; nome e endereço de e-mail; período de validade do certificado; nome da AC que emitiu o certificado; número de série do certificado digital; assinatura digital da AC. Um certificado digital serve para verificar a identificação digital de quem assina a mensagem para garantir que não ocorreu nenhuma falsificação ou adulteração. Ele garante, também, a identidade do assinante. [11].

Uma AC é responsável pela emissão dos certificados digitais. Para garantir a legitimidade do processo, é comum que as autoridades certificadoras verifiquem documentação física dos requerentes, antes de emitir o certificado. Uma AC é também responsável por emitir uma lista com certificados revogados [39].

Um certificado pode ser auto assinado, ou seja, assinado pela mesma entidade cuja identidade ele certifica. O certificado é assinado com usa própria chave privada. Se todos os envolvidos na comunicação confiam uns nos outros para proteger suas chaves privadas e podem confirmar a transferência de chaves públicas, certificados auto assinados podem diminuir os riscos. Podem ser úteis para uso em redes privadas onde poucos usuários têm acesso e a identidade de todos é garantida. Um certificado auto assinado não é automaticamente reconhecido pela maioria

dos navegadores web e não oferece nenhuma garantia em relação à identidade da empresa que está provendo o site. Esse tipo de certificado pode ser criado gratuitamente usando várias ferramentas. Esta foi a opção escolhida para o desenvolvimento da aplicação, apresentada neste trabalho, através da biblioteca OpenSSL [40] [11].

2.2 Rede e Internet

A internet é uma rede de computadores que faz a conexão de milhões de equipamentos computacionais em todo o mundo. Os dispositivos conectados a ela são conhecidos como hospedeiros e sistemas finais. Os sistemas finais se dividem em duas categorias: clientes e servidores. O programa cliente funciona em um sistema final, solicitando e recebendo um serviço de um programa servidor.

Para a troca de informações pela internet, os sistemas finais precisam executar protocolos, que são os responsáveis pelo seu controle. Os mais importantes da internet são os protocolos TCP (*Transmission Control Protocol*) e o IP (*Internet Protocol*), que juntos são conhecidos como TCP/IP. Existem outros protocolos, da camada de aplicação, tais como o SMTP para e-mail e o HTTP para a Web. A camada de aplicação, além dos protocolos de comunicação, possui métodos de interface usados em comunicações processo a processo através da internet. Para a gerência e transferência de dados entre hospedeiros numa rede cliente-servidor ou P2P são utilizados os protocolos da camada de transporte, como TCP e UDP[12].

Para a comunicação entre os sistemas finais existem dois tipos de serviços: os orientados à conexão e os não orientados à conexão. No primeiro grupo, os programas, tanto cliente como servidor, trocam entre si pacotes de controle antes do envio de pacotes com os dados reais. Esse processo é conhecido como apresentação (*handshaking*) e permite que servidor e cliente se

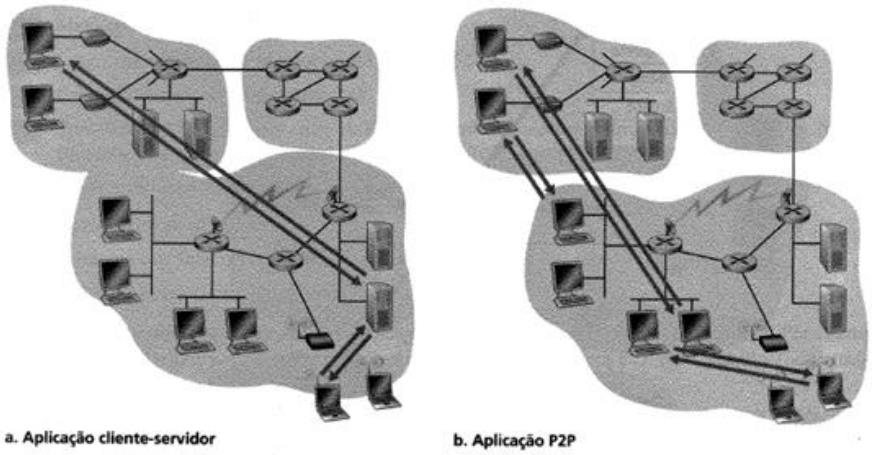
preparem para uma grande quantidade ("rajada") de pacotes. Quando concluída a apresentação, diz-se que foi estabelecida a conexão entre os sistemas finais (hospedeiros).

Nos serviços orientados à conexão estão os protocolos Telnet (*login* remoto), SMTP, FTP (*File Transfer Protocol*, para a transferência de arquivos) e o HTTP, da camada de aplicação, e o TCP, da camada de transporte [12].

Já nos serviços não orientados à conexão, não há esse processo de apresentação entre sistemas finais, que desejam enviar pacotes entre si, simplesmente ele os enviam. Esses dados podem ser entregues em maior velocidade pois não há a fase de apresentação, de controle de fluxo, nem controle de congestionamento, porém não é uma forma confiável de transmissão, dado que uma fonte não tem certeza que os dados chegaram à outra.

O serviço não orientado a conexão é denominado Protocolo de Datagrama do Usuário (*User Datagram Protocol* - UDP).

Com esses serviços pode-se falar de dois modelos de comunicação pela internet, o modelo cliente/servidor e o modelo P2P (*peer to peer*).

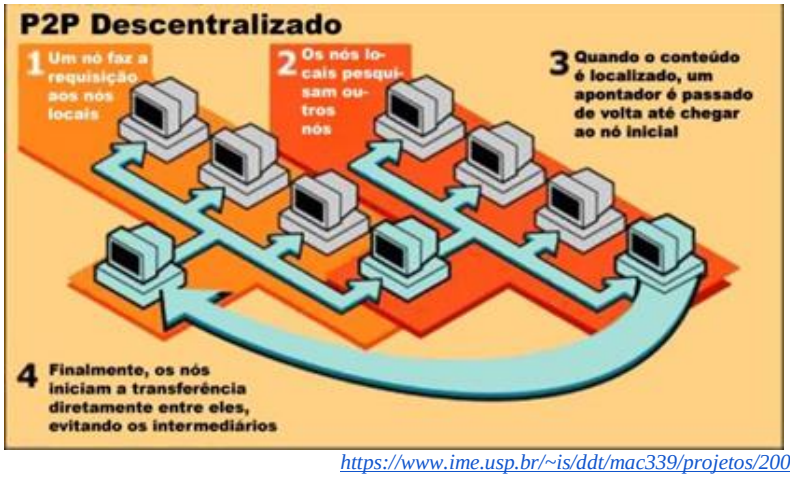


Kurose, J. F. e Ross, K. W. (2006) Redes de Computadores e a Internet, Pearson, 5ª Edição
Figura 8 - Modelos Cliente/Servidor e Modelo P2P

No modelo **cliente-servidor** (figura 8a), orientado à conexão, há um hospedeiro servidor em funcionamento atendendo as requisições dos hospedeiros clientes, que podem ou não estar em funcionamento. Após receber a requisição do cliente, esse servidor envia o objeto requisitado para o cliente. Os clientes não se comunicam diretamente entre si nesse modelo.

Diferentemente do modelo cliente servidor, o **P2P** (figura 8b) consiste na comunicação direta entre duas máquinas para o compartilhamento de arquivos ou serviços, e dessa maneira cada estação participante possui as mesmas capacidades e responsabilidades. O modelo de rede p2p pode ser descentralizado ou híbrido.

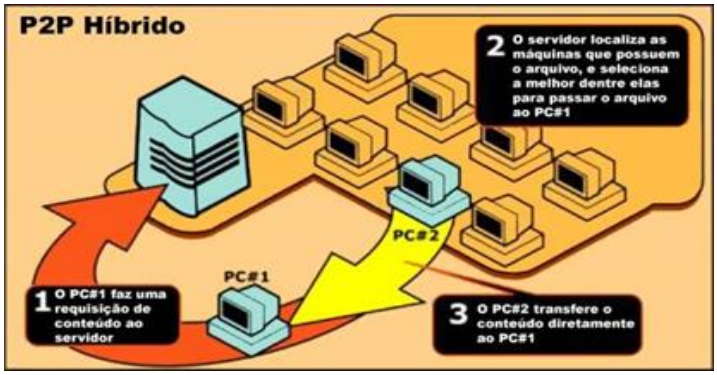
O modelo descentralizado (figura 9) opera sem a ajuda de qualquer tipo de servidor ou repositório central de informações. Todos os nós podem assumir a condição de cliente ou servidor simultaneamente. Os nós (dispositivos computacionais) fazem as solicitações para os nós conhecidos ao seu alcance e estes conectam-se a outros nós até que o recurso solicitado seja alcançado. A partir do momento que isso acontece, o nó solicitante recebe um apontador para o nó que possui o recurso e estabelece com ele uma conexão direta para a realização da transação sem a necessidade de intermediários.



https://www.ime.usp.br/~is/ddt/mac339/projetos/2001/demais/bello/intro2.html

Figura 9 - Modelo descentralizado de P2P

No modelo híbrido (figura 10), são usados servidores para tarefas como autenticação do usuário, serviços de diretório e mapeamento da disponibilidade dos recursos, encontrando a melhor maneira de responder a uma demanda ou se é possível cumpri-la naquele momento. Os nós conectam-se ao servidor para a realização da transação ou de algum serviço, então o servidor busca nos outros nós a solicitação do demandante e devolve a este uma informação de modo a permitir a conexão direta entre o nó demandante e o nó que possui o serviço ou recurso.



https://www.ime.usp.br/~is/ddt/mac339/projetos/2001/demais/bello/intro2.html
Figura 10 - modelo p2p híbrido

O modelo híbrido é o modelo utilizado na aplicação desenvolvida. Um servidor guarda as informações dos usuários, como endereço IP, chave pública, status online, e a comunicação entre eles é realizada diretamente.

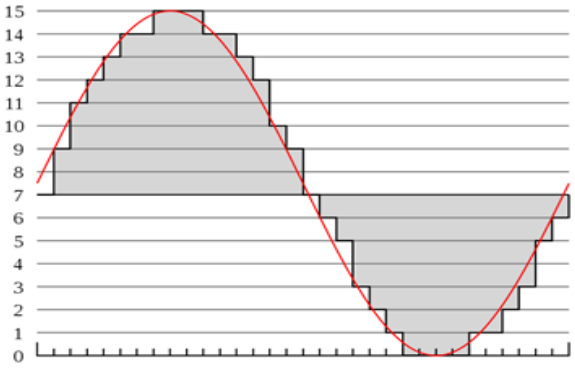
2.3 Codificação de Voz

Para enviar uma mensagem de voz através da internet é necessária antes convertê-la de analógica para digital. Essa conversão pode ser feita usando PCM (*pulse-code modulation* ou modulação por código de pulso), um método para representar digitalmente amostras de sinais analógicos. PCM é o codec padrão para CDs de áudio por permitir pouca ou nenhuma compressão[31].

Um sistema PCM trabalha com uma taxa de 64kbps [32] [33]. Dependendo da taxa de bits disponível para cada usuário, pode ser necessário o uso de outros codecs para reduzir a taxa de bits para um valor menor.

Uma *stream* PCM é uma representação de um sinal analógico, em que a magnitude do sinal é medida em intervalos regulares, gerando amostras (*samples*) que são quantizadas para o valor mais próximo em uma lista de valores digitais.

As *streams* têm duas propriedades básicas que determinam sua fidelidade com o sinal analógico original: a taxa de amostragem, que é o número de vezes por segundo que amostras são criadas, e a taxa de bits (*bit depth/ bit rate*), que determina a quantidade de valores digitais possíveis para cada amostra.



https://en.wikipedia.org/wiki/Pulse-code_modulation

Figura 11 - Amostragem para PCM

Na figura 11, é apresentado o gráfico da função seno, uma onda, em vermelho. A onda está sendo amostrada e quantizada para PCM. As amostras são tiradas em intervalos regulares, mostrados como pontos do eixo x. Para cada amostra, um dos valores disponíveis (pontos no eixo y) é escolhido por algum algoritmo. Isso produz uma representação do sinal de entrada que pode ser facilmente codificado como dados digitais para armazenamento ou manipulação. Na figura 11, os valores das amostras são 7,9,11,12,13,14,15,15,15,14, etc. Transformando esses valores para binário temos: 0111, 1001, 1011, 1100, 1101, 1110, 1111,1111,1111, 1110.

Para decodificar, o processo é o inverso. Após cada período de amostragem, o próximo valor é lido e o sinal de saída é movido para o novo valor.

Depois de codificado, pode ser necessário comprimir as amostras que tem taxa de 64kbps para taxas menores. Para isso existem vários codecs. A biblioteca VoIP do projeto, PJSIP (seção 3.1.2), usa o codec AMR-NB (*Adaptive multi-rate Narrow Band*), que permite uma compressão de dados maior do que o PCM.

O AMR-NB realiza os dois processos: a conversão do sinal analógico para digital e a compressão desse sinal para uma taxa de bits menor.

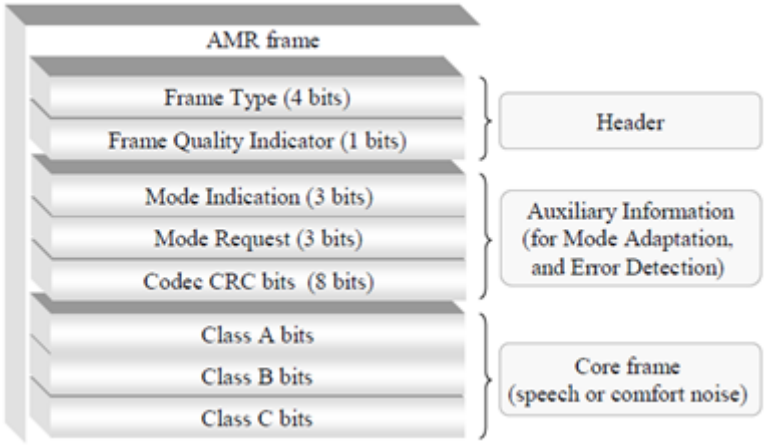
Como seu nome já indica, o AMR pode trabalhar com várias taxas de bits (12.2, 10.2, 7.95, 7.40, 6.70, 5.90, 5.15 e 4.75 kbps), possui 8 codecs diferentes e seleciona a melhor taxa dependendo da condição da rede, se o sinal estiver fraco (perde de pacotes, lentidão, *jitter* alto) utiliza taxas de bits menores para facilitar a transmissão.

O AMR gera amostras a cada 20 ms, o que significa que ele pode mudar a taxa de bits a cada 20 ms e cada amostra pode possuir uma taxa de bits diferente.

O **pacote de dados do AMR** (figura 12) é dividido em cabeçalho (*header*), informação auxiliar (*auxiliary information*) e bloco principal (*core frame*).

O cabeçalho é dividido em Tipo e Indicador de qualidade. O Tipo pode indicar o uso de um dos 8 codecs, um bloco com lixo ou um bloco vazio. O indicador de qualidade sinaliza se o bloco é bom ou ruim (isso depende do sinal da chamada: sinal fraco, bloco ruim, sinal bom, bloco bom).

A informação auxiliar é dividida em indicação de modo, pedido de modo e campo de verificação de erros (CRC codec), sendo utilizada para correção de erros e mudança de taxa de bits.



https://en.wikipedia.org/wiki/Adaptive_Multi-Rate_audio_codec

Figura 12 - Pacote de Dados do codec AMR

O Bloco principal é usado para enviar a voz codificada, sendo divido em Classes A, B e C. O motivo para dividir a voz em classes é que estas podem estar sujeitas a diferentes proteções contra erros na rede. A classe A contém os dados mais sensíveis e quaisquer erros nesses bits podem gerar falhas na voz depois de decodificada. Os bits da classe A estão protegidos pelo campo CRC para evitar erros. A classe B contém dados pouco importantes presentes em todas as taxas de conversão e a classe C contém dados pouco importantes presentes apenas em taxas de conversão altas.

2.4 VoIP

VoIP é uma abreviação de "*Voice over Internet Protocol*", também conhecido, entre outras denominações como telefonia IP, telefonia em banda larga ou voz sobre banda larga. Uma tecnologia relativamente recente, que trabalha com a digitalização da voz e sua transmissão pela internet[13].

O primeiro software a realizar este tipo de serviço foi o *Internet Phone Software*, em 1995, construído pela empresa Vocaltec Inc., em Israel. A ideia era o desenvolvimento de um sistema que permitisse a utilização dos recursos multimídia de um computador doméstico para iniciar conversas de voz pela internet. A qualidade era muito baixa, com muitos cortes e atrasos[14].

Existem basicamente três formas de utilização do VoIP, que são elas, Telefones IP, Dispositivos ATA e computador para computador.

Os telefones IP são aparelhos específicos, bem semelhantes aos aparelhos comuns, que se conectam diretamente em um roteador e tem embutidos *hardwares* e *softwares* para a realização de uma ligação pela internet (ligação IP).

Dispositivos ATA (adaptador telefônico analógico) são adaptadores que possibilitam a conexão de um telefone comum a um computador ou diretamente à conexão de internet para usar VoIP. Este dispositivo faz a conversão do sinal analógico para o digital possibilitando a transmissão pela internet.

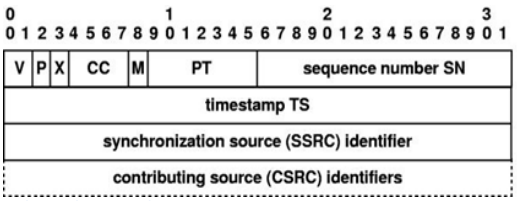
A transmissão de computador para computador é a mais utilizada. Sendo necessário o software, um microfone, alto falantes, placa de som e uma conexão com a internet. Para que seja mais efetivo é necessária uma conexão rápida e confiável. Celulares e Smartphones, por exemplo, enquadram-se nessa categoria de utilização do VoIP, pois fazem uso de softwares que permitem o uso do VoIP.

Antes de ser abordado tecnicamente como é realizado a comunicação com VoIP, vale a pena verificar como funciona uma ligação telefônica comum. Este tipo de rede usa comutação de circuitos e funciona da seguinte maneira:

1. Um receptor ouve o sinal de discagem, sinal de que existe uma conexão com o provedor de telefonia.

2. O usuário disca o número do destino.
3. A chamada é direcionada através do comutador do operador local para o destino.
4. Uma conexão é estabelecida entre o usuário de origem e o destino usando vários comutadores interconectados pelo caminho.
5. O telefone no destino alerta a chegada de uma chamada.
6. Usuário no destino aceita a chamada.
7. A conexão abre o circuito.
8. Depois de algum tempo, a chamada é terminada por um dos lados.
9. Ao fim da chamada, o circuito é fechado e o caminho usado entre origem e destino é liberado, permitindo a realização de novas conexões.

O VoIP, por fazer uso da internet, trabalha sobre uma rede de dados e comutação de pacotes e não de circuitos. Dessa maneira, utiliza alguns protocolos para a comunicação, envio de dados e conversão de sinal. Para o transporte de pacotes é usado o protocolo UDP (*User Datagram Protocol*), enquanto para sinalização de chamadas são utilizados, entre outros, os protocolos SIP e o H.323. No transporte de mídia são usados os protocolos RTP (*Real Time Protocol*) [15], que define o modo como deve ser realizada a fragmentação do fluxo de dados de mídia, e o RTCP (*Real Time Transport Control Protocol*), que realiza o controle por meio de envio periódico de pacotes de controle a todos os participantes da chamada (conexão). Em uma chamada VoIP, o RTP envia os pacotes de voz fragmentado e o cabeçalho (figura 13) indica como os pacotes devem ser reconstruídos. O RTCP gerencia a chamada, permitindo detectar perda de pacotes e compensar por eventuais atrasos.

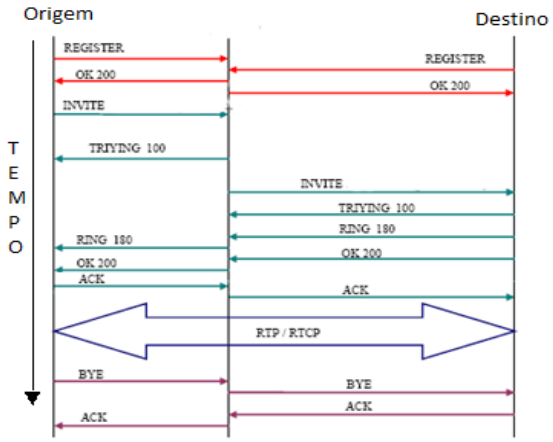


https://prof.hti.bfh.ch/myf1/www/projects/polyphem/www/documents/projectwork-techreport-mediainternet.html

Figura 13 - Cabeçalho RTP

Inicialmente, com o uso de protocolos de sinalização de chamada, digita-se o número ou o contato é selecionado para a execução da chamada. Uma vez atendida, é aberta uma conexão entre os dispositivos e a conversa, iniciando assim a comunicação.

Após a abertura da conexão, o próximo passo do funcionamento do VoIP é a conversão do sinal analógico de voz em sinal digital. Uma vez digitalizado esse sinal, este é dividido em datagramas e enviado pela rede para o destinatário final. Como mencionado anteriormente, este processo de envio dos datagramas de áudio ocorre com o uso dos protocolos RTP e RTCP [15].



http://www.en.voipforo.com/SIP/SIP_example.php

Figura 14 - Visão geral de uma chamada VoIP com SIP.

A figura 14 demonstra a progressão de uma chamada VoIP usando o protocolo SIP, ambos os lados da conversa se registram (REGISTER) e, em seguida, a origem sinaliza a chamada para o destino (RING). Depois que o destino aceita a chamada, a conversa é realizada e os pacotes são enviados usando o protocolo RTP/RTCP. No fim ambos os lados saem da conversa (BYE).

3 - A Aplicação

Neste capítulo serão apresentadas as tecnologias utilizadas para o desenvolvimento da aplicação e o processo de implementação. Esta aplicação foi desenvolvida utilizando o Sistema operacional Windows 7, a Linguagem de programação C++ e o Visual Studio, uma IDE da Microsoft para desenvolvimento de aplicativos. Foi acoplada a biblioteca *open source* PJSIP, que possui as funcionalidades para implementação das chamadas VoIP.

O servidor para registro e direcionamento de chamadas é o Kamailio. Este servidor foi instalado numa máquina virtual, *Oracle Virtual Box,* com a distribuição Debian, do Linux. Para a segurança da comunicação foram utilizadas as bibliotecas libsrtp e OpenSSL, que disponibilizam funções para criptografia e certificação, respectivamente.

Inicialmente a aplicação usaria a biblioteca *Open Source* OPAL [28], com uma implementação do protocolo H323, para as funcionalidades VoIP, mas ela se mostrou incompatível com o Visual Studio 2013 e durante as tentativas de *build*, a OPAL exibia vários erros, principalmente sobre a falta de componentes. Sem suporte adequado para a utilização da

biblioteca não foi possível solucionar os problemas apresentados. A OPAL foi abandonada e sua sucessora H323PLUS [29] foi escolhida. Os mesmos problemas foram encontrados e mais uma vez a biblioteca foi substituída. Dessa vez pela PJSIP [18], que implementa o protocolo SIP, atualizada e com suporte.

Para um melhor entendimento será realizada uma breve exposição dos padrões acima citados: H.323 e SIP.

H.323 é um padrão da *International Communication Union* (ITU) [41] para promover comunicação audiovisual através de redes IP. Ele define uma série de padrões para a transmissão de pacotes de dados multimídia. É um protocolo robusto, focado na comunicação multimídia, incluindo transferência de áudio, vídeo e dados [42]. Ele descreve o uso de outros protocolos da ITU e IETF. O núcleo de quase todo sistema h.323 contém:

- H.225 Registro, admissão, status e sinalização de chamadas; este padrão é utilizado para prover resolução de endereços, serviços de controle de admissão e estabelecer comunicação.
- H.245 é responsável pelo controle de comunicação multimídia.
- RTP é utilizado para envio de dados multimídia.
- H.235 é o protocolo de segurança. Tanto para sinalização de chamada como para comunicação multimídia.

O protocolo SIP [43] descreve um método de iniciar e terminar sessões de usuário online, incluindo transferência de conteúdo multimídia (vídeo e áudio conferencia, mensagens instantâneas, jogos online). Ele lida com a conversa inicial entre dispositivos, permitindo que realizem operações como: descobrir outros dispositivos, convidar alguém para participar de uma sessão, comunicar a disponibilidade de um dispositivo e definir o tipo de sessão [45]. O SIP não

transmite o conteúdo da sessão, protocolos como o RTP são usados para isso. É uma implementação mais simples, com apenas seis métodos, e mais flexível que o H.323. A maior escalabilidade do SIP facilita a integração com aplicações da internet. Ele é independente da camada de transporte, funciona com TCP, UDP ou qualquer outro protocolo da camada de transporte [44] e oferece a possibilidade de comunicação com outras aplicações que o utilizam.

3.1 Tecnologias Estudadas

Para a implementação deste projeto, foram necessários a pesquisa e o estudo de diversas tecnologias e ferramentas.

A comunicação entre aplicações é realizada pelo protocolo SIP (*Session Initiation Protocol*), protocolo de comunicação para sinalização e controle de sessões de comunicação multimídia, pode ser usado para criar, modificar e terminar sessões. Este protocolo foi usado no projeto através da PJSIP [19], uma biblioteca livre e *Open Source*, que combina os protocolos em uma API de comunicação de alto nível, portátil e adequada para qualquer tipo de sistema, de *desktops* a dispositivos móveis. A PJSIP é desenvolvida e mantida por Teluu Ltda. Sua primeira versão, 0.2, foi lançada em 2005 e atualmente a biblioteca está na versão 2.0.

A PJSIP provê o necessário (sinalização de chamada, codificação de voz, controle de chamada) para o desenvolvimento de uma aplicação de comunicação multimídia em tempo real, possuindo muitos recursos SIP. Esta é customizável e modular, os recursos não utilizados podem ser desligados.

Para registrar e localizar cada usuário foi utilizado o servidor Kamailio SIP Server [24], capaz de lidar com várias chamadas por segundo. Alguns de seus recursos são: TCP assíncrono,

UDP e comunicação segura via TLS. O Kamailio roda em Linux e neste projeto foi usada a versão Debian, uma distribuição do Linux muito utilizada em PCs e servidores.

Para garantir a segurança na troca de informações, a PJSIP usa a libsrtp, uma implementação livre do protocolo SRTP [18]. O SRTP (*Secure Real-time Transport Protocol*) [18] é um perfil de segurança para o protocolo RTP (*Real-time Transport Protocol*) que adiciona confidencialidade, autenticação de mensagens e integridade. Para criptografar e descriptografar o fluxo de dados, este protocolo usa AES como algoritmo de criptografia padrão.

Afim de garantir a autenticidade dos usuários registrados, faz-se necessário o uso de TLS (*Transport Layer Security*) e SSL (*Secure Socket Layer*) [21], que são protocolos de transporte desenvolvidos para prover segurança de comunicações. Eles utilizam criptografia assimétrica para autenticar o receptor da mensagem e negociar uma chave de sessão simétrica, que será usada para criptografar dados transmitidos entre origem e destino.

3.2 Desenvolvimento

Escolhida a biblioteca, o processo de desenvolvimento seguiu os passos abaixo:

1. Realizar a configuração e *build* da PJSIP [25] [26].
2. Adicionar dependências da biblioteca ao projeto do Visual Studio.
3. Instalar o servidor SIP para registrar usuários (anexo II). Este foi criado no sistema Linux Debian em uma máquina virtual dentro do sistema Windows, onde o programa foi desenvolvido.

    3.1. Cadastrar usuários no servidor usando o comando "*kamctl add usuario@dominio senha*". Dois usuários foram criados, denominados teste1

(*kamctl add teste1@kamailio.org senha*) e teste2 (*kamctl add teste2@kamailio.org pass*).

4. Desenvolver o código da chamada. Foram realizados dois *builds* em códigos diferentes, um para o recebedor de chamada e outro para o chamador, um para cada usuário cadastrado.

5. Adicionar ao campo use_srtp a string “PJMEDIA_SRTP_MANDATORY” e inserir o *path* da libsrtp ao projeto. A libsrtp é um componente da PJSIP para uso do protocolo de segurança SRTP [20].

6. Alterar, no servidor, o registro dos usuários para indicar que a chamada utiliza criptografia. Os usuários, que inicialmente estavam registrados como **sip**, devem ser registrados como **sips** (figura 15). A partir desse momento a chamada fará uso do algoritmo de criptografia AES-128 suportado pela libsrtp. O Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (NIST) [47] afirma que o algoritmo de criptografia AES com chaves de 128 bits é seguro até, pelo menos, 2030 [46].

```
void Pjaux::register_on_server() //função pra fazer o registro no servidor

{
    pj_status_t status;
    pjsua_acc_config cfg; // configuração de conta
    cfg.id = pj_str("sips:" SIP_USER "@" SIP_DOMAIN); // id de registro com sips
    cfg.reg_uri = pj_str("sips:" "192.168.56.101"); // sips passando o ip do servidor
    cfg.cred_info[0].realm = pj_str("*"); // ?
    cfg.cred_info[0].scheme = pj_str("digest"); // ?
    cfg.cred_info[0].username = pj_str(SIP_USER);
    cfg.cred_info[0].data_type = PJSIP_CRED_DATA_PLAIN_PASSWD;
    cfg.cred_info[0].data = pj_str(SIP_PWD);
    cfg.use_srtp = PJMEDIA_SRTP_MANDATORY; // aqui obriga a usar srtp
    cfg.srtp_secure_signaling = 0; // deveria ser usado com valor 1 ou 2 porém qdo mexi nessas configs,
    foi antes de add srtp e por isso tinha q usar valor 0
    status = pjsua_acc_add(&cfg, PJ_TRUE, &acc_id); // faz o registro com as configurações anteriores
    if (status != PJ_SUCCESS) error_exit("Error adding account", status);

}
```

Figura 15 - Registro de usuário indicando uso de criptografia

7. Incrementar a segurança na camada de transporte usando TLS, através da biblioteca OpenSSL. Para isso é necessário fazer alterações nas configurações da PJSIP.

   7.1. Instalar a OpenSSL win 32 e referenciá-la na PJSIP.

7.2. Fazer um novo *build* da PJSIP.

7.3. Configurar Transporte TLS, passando como parâmetros o certificado, a chave e a senha do servidor *proxy* do SIP, seguindo as instruções presentes em [28]. Esse certificado é usado somente na configuração do TLS. Na aplicação desenvolvida, a criação do certificado foi realizada por meio da OpenSSL.

7.4. Instalar o módulo TLS no servidor, fazer um novo *build* do servidor

7.5. Indicar nas configurações do servidor o caminho do certificado do servidor *proxy* do SIP.

7.6. Copiar o certificado e chave, acima citados, para o lado cliente, que os referencia na configuração do TLS.

Após essas alterações, a aplicação pode realizar chamadas de voz usando criptografia nas camadas de aplicação e transporte, fazendo uso de ferramentas livres para desenvolvimento de aplicações VoIP disponíveis na web.

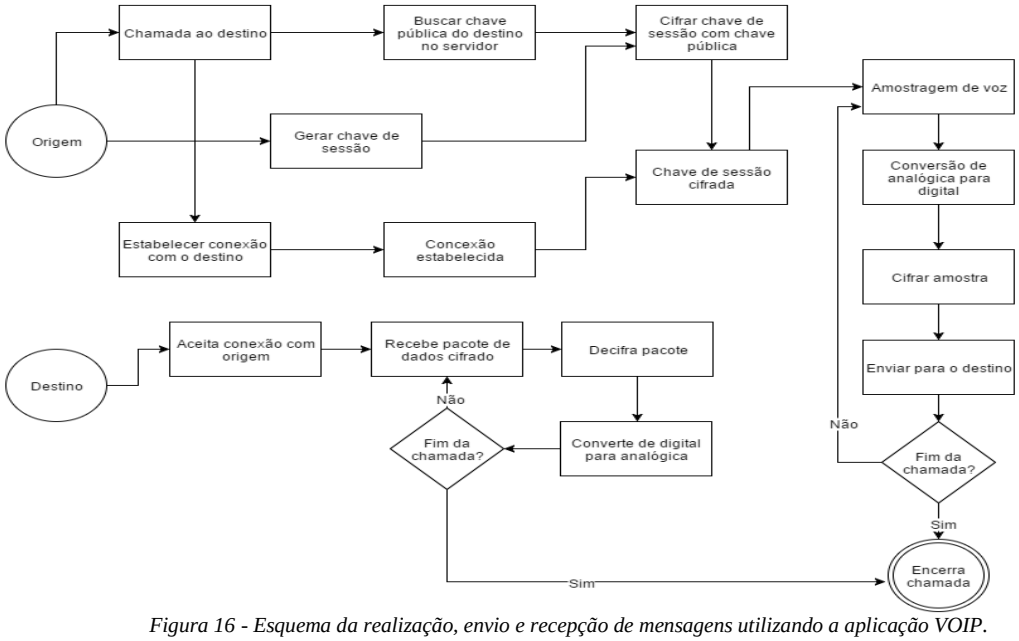


Figura 16 - Esquema da realização, envio e recepção de mensagens utilizando a aplicação VOIP.

A figura 16 apresenta o funcionamento da aplicação. Com ambos os lados da conversa registrados no servidor SIP, ocorre a chamada ao destino, onde a origem seleciona o destino da conversa e envia para o servidor um pedido para iniciar a conexão. Nesse pedido é enviado a assinatura digital da origem gerado com um certificado auto assinado. O servidor confirmar a identidade do usuário da origem e responde com os dados do destino (endereço IP, chave pública, status online) e gera uma chave de sessão que será usada para cifrar a conversa quando a conexão for estabelecida. As chaves são geradas com o protocolo SRTP que cria chaves de 128 bits usando o algoritmo de criptografia simétrica AES (seção2.1). No destino, a identidade do usuário da origem é novamente checada. Quando uma conexão segura, entre ambos os lados, é estabelecida, a chave de sessão é decifrada e a conversa é iniciada. Nesse momento, a amostragem de voz é feita, os pacotes são cifrados usando a chave de sessão criada e são enviados para ambos os lados. O codec para codificação de voz (seção 2.3) utilizado na aplicação, AMR, gera 160 amostras em frames de 20 ms, o *bitrate* pode variar de 4.75 kbps a 12.2 kbps de acordo com a qualidade da conexão [50]. Durante a conversa, para ambos os lados, a voz é capturada e convertida para um sinal digital que é cifrado usando a chave de sessão e enviada. O processo contrário é realizado para que o áudio possa ser compreendido. Quando a conversa termina, a conexão é finalizada e chave de sessão é destruída.

3.2.1 Testes

Para o funcionamento da aplicação na plataforma Windows é necessária a instalação da biblioteca OpenSSL win32, o framework .net e a utilização dos certificados digitais, que são armazenados em “C:/certs”.

Para a realização dos testes foram usadas 3 diferentes configurações de computadores, um desktop com o servidor de SIP Kamailio, dois notebooks com o sistema operacional Windows 7 e um notebook com o sistema operacional Windows 10. A figura 17 mostra a comunicação entre as máquinas. Ambos os clientes se comunicam com o servidor para registro e a conversa é realizada diretamente entre eles.

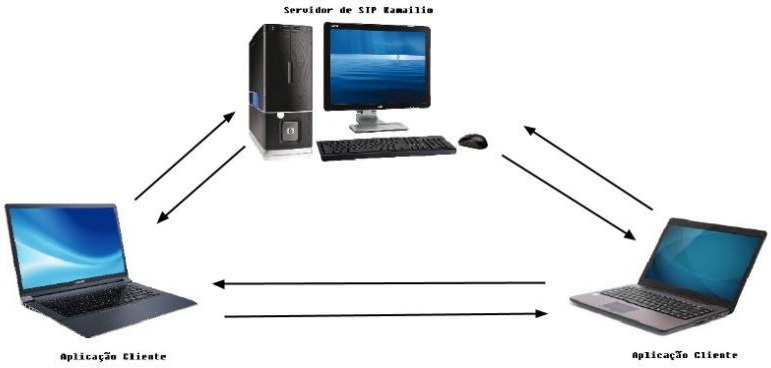


Figura 17 - Comunicação entre clientes e servidor SIP

Dois usuários foram cadastrados no servidor, o usuário teste1 com a senha, “senha”, e o usuário teste2 com a senha “pass”.

No teste com certificados, o certificado, a chave e a senha do certificado são respectivamente: cert.pem, key.pem e SACIS. A senha da chave do certificado é gerada na criação do certificado pela OpenSSL e é opcional, oferecendo uma camada extra de segurança. Certificados, chaves e senhas genéricas foram utilizadas para esse teste, aparecendo como “gen” nas tabelas 1 e 2.

Os dois tipos de teste realizados foram: teste de registro no servidor e o teste de chamada.

O objetivo do **teste de registro no servidor** é verificar o funcionamento da função de registrar usuário no servidor, ou seja, uma espécie de *login* do usuário no servidor de SIP, de maneira a poder realizar e receber chamadas.

O primeiro teste não levou em conta o certificado, somente o nome de usuário e senha. A tabela 1 mostra o resultado desse teste.

Usuário	Senha	Resultado
teste1	senha	sucesso
teste1	pass	falha
aleatório	senha	falha
aleatório	pass	falha
teste2	senha	falha
teste2	pass	sucesso

Tabela 1 - Teste de registro no servidor

Após a instalação do módulo de TLS, o foco do teste foi garantir a verificação do certificado digital por parte do servidor.

Os certificados foram gerados no lado servidor por meio da biblioteca OpenSSL. Primeiro criou-se uma autoridade certificadora (demoCA) com certificado auto assinado e a partir dela foram gerados os certificados de usuário, para o domínio kamailio.org, que foi o utilizado no cadastro dos usuários. Como o certificado utilizado é o mesmo para todos os usuários, foi tomado por base o usuário teste1. A tabela 2 mostra o resultado desse teste:

Certificado	Chave	Senha da Chave	Resultado
cert.pem	key.pem	“SACIS”	Sucesso
gen.pem	key.pem	“SACIS”	Falha
cert.pem	Sem chave	“SACIS”	Falha
cert.pem	key.pem	“gen”	Falha
Sem certificado	key.pem	“SACIS”	Falha
Sem certificado	Sem chave	“SACIS”	Falha
cert.pem	key.pem	Sem senha	Falha
cert.pem	gen.pem	“SACIS”	Falha
gen.pem	gen.pem	“SACIS”	Falha
gen.pem	gen.pem	Sem senha	Falha

Tabela 2 - Teste de verificação de certificado

O registro é verificado pela função “kamctl monitor” no servidor. A figura 18 mostra os dois usuários registrados no servidor pela linha “registered_users”.

```
Up since:: Fri Jan 15 14:01:46 2016
Up time:: 28751 [sec]

Transaction Statistics:
tmx:UAS_transactions = 0
tmx:UAC_transactions = 0
tmx:inuse_transactions = 0

Stateless Server Statistics:
sl:sent_replies = 14
sl:sent_err_replies = 0
sl:received_ACKs = 0

UsrLoc Stats:
usrloc:location-contacts = 2
usrloc:location-expires = 5
usrloc:location-users = 2
usrloc:registered_users = 2
```

Figura 18 - Usuários registrados no servidor

O **teste de chamada** teve o objetivo de verificar o funcionamento da biblioteca PJSIP na realização de chamadas VoIP. Foi executado de duas maneiras, a primeira em ambiente local, com duas aplicações na mesma máquina comunicando-se e em ambiente desacoplado, com uma máquina com o servidor de SIP e outras duas máquinas com a aplicação cliente cada uma.

Na figura 19 é possível ver a comunicação entre as aplicações.

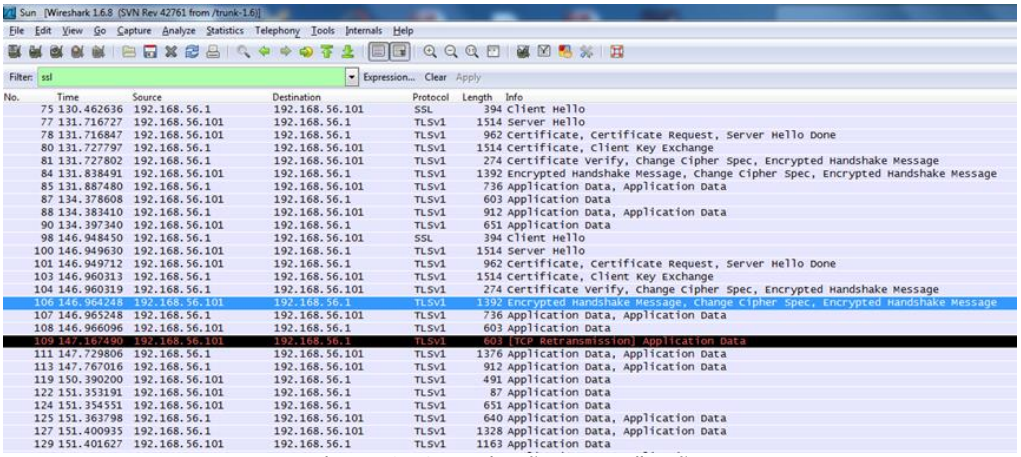
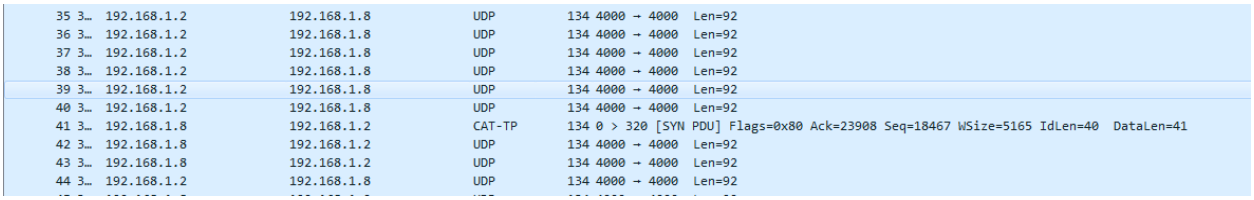


Figura 19 - Comunicação entre aplicações

No segundo cenário, o com máquinas desacopladas, um desktop foi utilizado como o servidor de SIP e outros dois notebooks foram usados como clientes. Os notebooks fazem o registro no servidor e depois passam a se comunicar entre si. Os notebooks possuíam sistema

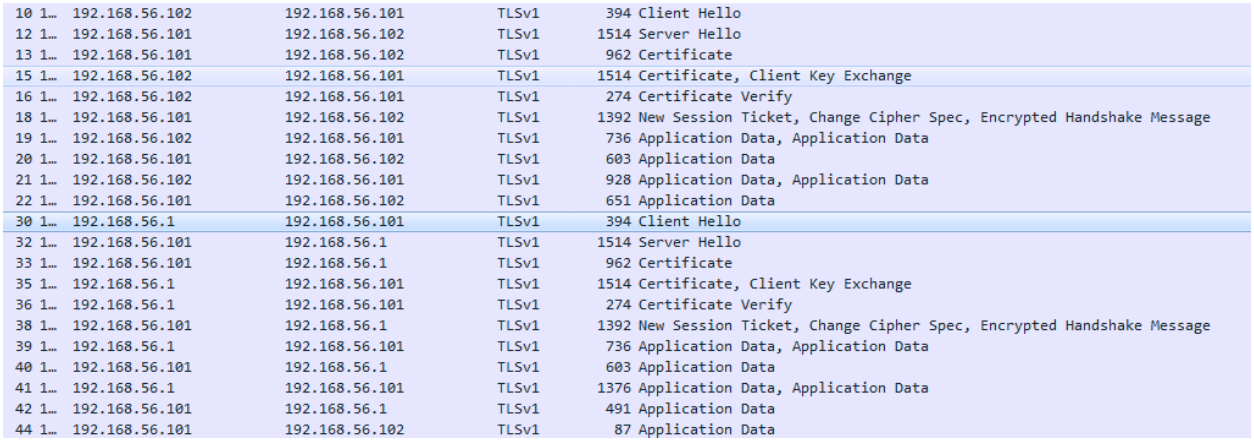
operacional Windows 7. O IP do servidor de SIP é o 192.168.56.101, enquanto o dos clientes é 192.168.1.2 e 192.168.1.8. A comunicação entre os clientes usa o transporte UDP, como é apresentado na figura 20.



35 3…	192.168.1.2	192.168.1.8	UDP	134	4000 → 4000 Len=92
36 3…	192.168.1.2	192.168.1.8	UDP	134	4000 → 4000 Len=92
37 3…	192.168.1.2	192.168.1.8	UDP	134	4000 → 4000 Len=92
38 3…	192.168.1.2	192.168.1.8	UDP	134	4000 → 4000 Len=92
39 3…	192.168.1.2	192.168.1.8	UDP	134	4000 → 4000 Len=92
40 3…	192.168.1.2	192.168.1.8	UDP	134	4000 → 4000 Len=92
41 3…	192.168.1.8	192.168.1.2	CAT-TP	134	0 > 320 [SYN PDU] Flags=0x80 Ack=23908 Seq=18467 WSize=5165 IdLen=40 DataLen=41
42 3…	192.168.1.8	192.168.1.2	UDP	134	4000 → 4000 Len=92
43 3…	192.168.1.8	192.168.1.2	UDP	134	4000 → 4000 Len=92
44 3…	192.168.1.2	192.168.1.8	UDP	134	4000 → 4000 Len=92

Figura 20 - Comunicação entre aplicações usando UDP

A comunicação com o servidor de SIP faz uso do transporte TLS, o *handshake*, a verificação do certificado e a troca de chave com os dois clientes (figura 21). Nesse caso, o servidor de SIP permanece com o IP 192.168.56.101, mas os clientes aqui aparecem com os IPs 192.168.56.102 e 192.168.56.1, que são utilizados na comunicação com o servidor.



10 1…	192.168.56.102	192.168.56.101	TLSv1	394	Client Hello
12 1…	192.168.56.101	192.168.56.102	TLSv1	1514	Server Hello
13 1…	192.168.56.101	192.168.56.102	TLSv1	962	Certificate
15 1…	192.168.56.102	192.168.56.101	TLSv1	1514	Certificate, Client Key Exchange
16 1…	192.168.56.102	192.168.56.101	TLSv1	274	Certificate Verify
18 1…	192.168.56.101	192.168.56.102	TLSv1	1392	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
19 1…	192.168.56.102	192.168.56.101	TLSv1	736	Application Data, Application Data
20 1…	192.168.56.101	192.168.56.102	TLSv1	603	Application Data
21 1…	192.168.56.102	192.168.56.101	TLSv1	928	Application Data, Application Data
22 1…	192.168.56.101	192.168.56.102	TLSv1	651	Application Data
30 1…	192.168.56.1	192.168.56.101	TLSv1	394	Client Hello
32 1…	192.168.56.101	192.168.56.1	TLSv1	1514	Server Hello
33 1…	192.168.56.101	192.168.56.1	TLSv1	962	Certificate
35 1…	192.168.56.1	192.168.56.101	TLSv1	1514	Certificate, Client Key Exchange
36 1…	192.168.56.1	192.168.56.101	TLSv1	274	Certificate Verify
38 1…	192.168.56.101	192.168.56.1	TLSv1	1392	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
39 1…	192.168.56.1	192.168.56.101	TLSv1	736	Application Data, Application Data
40 1…	192.168.56.101	192.168.56.1	TLSv1	603	Application Data
41 1…	192.168.56.1	192.168.56.101	TLSv1	1376	Application Data, Application Data
42 1…	192.168.56.101	192.168.56.1	TLSv1	491	Application Data
44 1…	192.168.56.101	192.168.56.102	TLSv1	87	Application Data

Figura 21 - Comunicação TLS

Adicionalmente foi feito um teste extra de comunicação com o uso do sistema operacional Windows 10 por parte de um dos clientes, e assim como no teste anterior a chamada funcionou.

3.4 Problemas Encontrados

Durante a execução do projeto, vários obstáculos tiveram que ser ultrapassados. Inicialmente, a biblioteca escolhida para a implementação da comunicação de voz, a H.323 Plus,

uma implementação livre do protocolo h.323, se mostrou incompatível com a versão mais recente do *visual studio* e todas as tentativas de *build* resultavam em inúmeros erros. Essa incompatibilidade ocorreu porque a biblioteca estava há muito tempo sem ser atualizada e o *visual studio* fazia atualizações no código, o que gerou muitos erros. Depois de tentativas malsucedidas para adaptar a H.323 esta biblioteca foi substituída pela PJSIP onde foram encontrados os seguintes problemas:

Erro de link com a PJSIP - O erro de link do projeto foi resolvido adicionando a biblioteca "ws2_32.lib" como dependência do projeto. Na data desse problema, a biblioteca "ws2_32.lib" não estava presente na lista de dependências necessárias de acordo com a documentação.

Registro de Usuários no servidor - Ao tentar realizar o registro com um usuário no servidor, era retornado o seguinte erro: **"credential failed to authenticate (pjsip_efailedcredential) [status=171100]".** Este foi resolvido com o cadastro manual dos usuários. De posse de nome e senha de usuário, o registro ocorreu sem problemas. A documentação disponível não deixa explícita a necessidade prévia de cadastro do usuário no servidor e a biblioteca não disponibiliza nenhum tipo de recurso para tal.

Dispositivo de áudio não encontrado - Na tentativa de chamada ocorreu o erro a seguir: ***error retrieving default audio device parameters unable to find default audio device <PJMEDIA_EAUD_NODEFDEV>***. A biblioteca precisa de um *headset* como device principal de áudio no computador para funcionar. Entretanto, essa informação não foi encontrada na documentação disponível pela PJSIP e só foi descoberta pesquisando em fóruns e afins. Ao executar a aplicação, o dispositivo de áudio disponível no computador utilizado deveria ser selecionado automaticamente, mas não era o que acontecia, a aplicação apresentava o erro já citado. Durante o desenvolvimento da aplicação, não foi encontrado o que causava esse erro. Este

é um erro comum na PJSIP. Para o desenvolvimento em sistemas Linux o erro pode ser corrigido instalando pacotes de desenvolvimento para ALSA (*Advanced Linux Sound Architecture*) [51]. No Windows, não existe ainda uma solução para esse problema que não seja utilizar um *headset*.

Incompatibilidade entre PJSIP e OpenSSL de 64 bits - Para poder usar o módulo de TLS, a PJSIP deve ser recompilada com dependências da OpenSSL sendo referenciadas. A biblioteca SIP não estava reconhecendo a OpenSSL e por isso, não era possível utilizar o módulo TLS. O problema nesse caso foi que o ambiente utilizava o Sistema Operacional Windows 7 - 64 bits e a versão da OpenSSL instalada, também era de 64 bits. A PJSIP requer o uso da versão de 32 bits.

Os problemas encontrados poderiam ter sido resolvidos rapidamente se houvesse uma boa documentação da biblioteca e seus componentes. Entretanto essa situação é algo comum no uso de software livre, onde a documentação depende dos próprios usuários.

4 - CONCLUSÕES E TRABALHOS FUTUROS

Dado o cenário atual de comunicação via aplicativos de trocas de mensagens como WhatsApp, Facebook Messenger e Skype, a segurança se torna fundamental. Das aplicações disponíveis atualmente, poucas oferecem algum tipo de segurança. E as que oferecem não permitem que o usuário faça alterações ou realize testes na criptografia utilizada para confirmar a segurança do aplicativo. Com o estudo apresentado, tem-se o passo a passo, além dos conteúdos teóricos necessários para a construção de uma aplicação VoIP com segurança a partir da combinação de ferramentas livres disponíveis na web que pode ser modificada por seus usuários. Uma outra vantagem é que, por utilizar o protocolo SIP, existe a possibilidade de comunicação com outras aplicações que utilizem o mesmo protocolo.

A implementação deste conteúdo foi árdua dado que muitas iniciativas de ferramentas livres, como as primeiras bibliotecas usadas no projeto (Capítulo 3), são descontinuadas ou falta documentação. Também é comum que ferramentas livres exijam a instalação de ferramentas de

terceiros que podem não funcionar ou estarem mal documentadas, o que prejudica o desenvolvimento.

Dado que as bibliotecas estejam disponíveis, integrar suas funcionalidades não é trivial dado que requer conhecimento técnico de cada parte da aplicação (criptografia, codificação de voz, protocolos de rede). Para entender o papel de cada uma das ferramentas no desenvolvimento é necessário todo um conhecimento teórico e prático sobre as áreas de rede, segurança e VoIP.

O experimento foi executado localmente, deixando em aberto a implementação em um servidor público e dedicado, e o desenvolvimento da interface da aplicação, para que esta possa ser utilizada por qualquer usuário. Também como trabalho futuro, está a checagem de que o algoritmo de criptografia AES utilizado no projeto segue as especificações da NIST [47]. Além disso, com o projeto será possível realizar os testes de performance da comunicação com outras cifras disponíveis, por exemplo RSA e curvas elípticas, e a inserção de algoritmos de criptografia proprietários na PJSIP, o que adicionará um nível maior de segurança à aplicação ao dar ao usuário o poder de escolha.

5 - REFERÊNCIAS BIBLIOGRÁFICAS

[1] Globo.com. Pela 1ª vez, acesso à internet chega a 50% das casas no Brasil, diz pesquisa http://g1.globo.com/tecnologia/noticia/2015/09/pela-1-vez-acesso-internet-chega-50-das-casas-no-brasil-diz-pesquisa.html , acessado em janeiro de 2016.

[2] Portal Brasil. Número de brasileiros na internet subiu para 95,4 milhões em 2014 http://www.brasil.gov.br/ciencia-e-tecnologia/2015/11/numero-de-brasileiros-na-internet-subiu-para-95-4-milhoes-em-2014 , acessado em janeiro de 2016.

[3] Ventura, Felipe. Gizmodo Brasil. A criptografia no WhatsApp não significa que todas as suas mensagens estão protegidas. http://gizmodo.uol.com.br/criptografia-whatsapp/, acessado em novembro de 2015.

[4] Cohen, Fred. A Short History of Cryptography. http://all.net/edu/curr/ip/Chap2-1.html, acessado em outubro de 2015.

[5] Biografia de Claude Shannon. https://www.nyu.edu/pages/linguistics/courses/v610003/shan.html, acessado em outubro de 2015.

[6] Oliveira, Ronielton Rezende. Criptografia Simétrica e Assimétrica: Os principais algoritmos de cifragem. http://www.ronielton.eti.br/publicacoes/artigorevistasegurancadigital2012.pdf, acessado em outubro de 2015.

[7] DOUGLAS, R. Stinson. Cryptography Theory and Practice, Chapman & Hall/CRC Press, 3rd. edition, Nov 2005.

[8] SCHNEIER, B. Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition. Editora John Wiley and Sons.

[9] Grupo de Teleinformática e Comunicação/UFRJ. Função Hashing. http://www.gta.ufrj.br/grad/09_1/versao-final/assinatura/hash.htm, acessado em outubro de 2015.

[10] Gazzarrrini, Rafael. TecMundo. O que é assinatura digital? http://www.tecmundo.com.br/web/941-o-que-e-assinatura-digital-.htm, acessado em outubro de 2015.

[11] Info Wester. Entendendo a certificação digital. http://www.infowester.com/assincertdigital.php, acessado em outubro de 2015.

[12] Kurose, J. F. e Ross, K. W. (2006) Redes de Computadores e a Internet, Pearson, 5ª Edição.

[13] Grupo de Teleinformática e Comunicação/UFRJ. História do Voip. http://www.gta.ufrj.br/grad/07_1/voip/historia.htm, acessado em outubro de 2015.

[14] Info Wester. Tecnologia VoIP. http://www.infowester.com/voip.php, acessado em outubro de 2015.

[15] Rouse, Margaret. Tech Target. Real-Time Transport Protocol (RTP) definition http://searchnetworking.techtarget.com/definition/Real-Time-Transport-Protocol, acessado em outubro de 2015.

[16] KELLY, V.T - VoIP for Dummies - Wiley Publishing Inc. 2005.

[17] SYED A. A., ILYAS M. VoIP HANDBOOK: Applications, Technologies, Reliability, and Security – CRC Press, 2008.

[18] Visual Studio. https://www.visualstudio.com/, acessado em outubro de 2015

[19] PJSIP.org. About PJSIP. http://www.pjsip.org/about.htm, acessado em outubro de 2015

[20] LibSRTP. About libSRTP. http://srtp.sourceforge.net/srtp.html, acessado em outubro de 2015.

[21] OpenSSL.org. https://www.openssl.org/, acessado em outubro de 2015.

[22] Harris, J.K. Undersanding SSL/TLS or What is SSL Certificate ad what does it do for me? https://computing.ece.vt.edu/~jkh/Understanding_SSL_TLS.pdf, acessado em outubro de 2015.

[23] IETF.org. The Transport Layer Security (TLS) Protocol Version 1.2. https://tools.ietf.org/html/rfc5246, acessado em outubro de 2015.

[24] Kamailio.org. http://www.kamailio.org/w/, acessado em outubro de 2015.

[25] PJSIP.org. Getting Started: Building for Microsoft Windows. http://trac.pjsip.org/repos/wiki/Getting-Started/Windows, acessado em outubro de 2015.

[26] PJSIP.org. Build Preparation.
http://trac.pjsip.org/repos/wiki/Getting-Started/Build-Preparation, acessado em outubro de 2015.

[27] PJSIP.org. Configuring PJSIP with TLS.
https://trac.pjsip.org/repos/wiki/TLS, acessado em outubro de 2015.

[28] NIL-Network Information Library. Configuring TLS support in Kamailio 3.1 – How to.
http://nil.uniza.sk/network-security/tls/configuring-tls-support-kamailio-31-howto, acessado em outubro de 2015.

[29] OPAL VoIP. http://www.opalvoip.org/, acessado em outubro de 2015.

[30] H.323+. http://www.h323plus.org/, acessado em outubro de 2015.

[31] IANA.org. Media Types.
http://www.iana.org/assignments/media-types/media-types.xhtml, acessado em janeiro de 2016.

[32] PCM. http://wiki.multimedia.cx/index.php?title=PCM, acessado em janeiro de 2016.

[33] Library of Congress “Linear Pulse Code Modulated Audio (LPCM)".
http://www.digitalpreservation.gov/formats/fdd/fdd000011.shtml, acessado em janeiro de 2016.

[34] Rouse, Margaret. Tech target. IP spoofing (IP address forgery or a host file hijack) definition.
http://searchsecurity.techtarget.com/definition/IP-spoofing, acessado em Janeiro de 2016.

[35] Nakamura, Emilio Tissato. Paulo Licio de Geus. In: Novatec. Segurança em redes cooperativas. Editora Novatec, 2007

[36] Teixeira, Fábio Augusto Alves. Sistema de armazenamento e compartilhamento de informações com segurança - ambiente Windows PC. UNIRIO, 2014.

[37] Steffen, Cherie L. VoIP innovations. 6 Common Ways to Suffer a VoIP Attack. http://blog.voipinnovations.com/blog/6-common-ways-to-suffer-a-voip-attack

[38] The Inside Out Security Blog. The Definitive Guide to Cryptographic Hash Functions. http://blog.varonis.com/the-definitive-guide-to-cryptographic-hash-functions-part-1/

[39] bry.com.br. O que é uma Autoridade Certificadora e quais são confiáveis? http://www.bry.com.br/duvidas-frequentes/o-que-uma-autoridade-certificadora-e-quais-so-confiveis/, acessado em janeiro de 2016

[40] Kassner, Michael. SSL/TLS certificates: What you need to know. http://www.techrepublic.com/blog/data-center/ssl-tls-certificates-what-you-need-to-know/, acessado em janeiro de 2016

[41] International communication union. H.323: Packet-based multimedia communications systems. https://www.itu.int/rec/T-REC-H.323/en, acessado em janeiro de 2016.

[42] Rouse, Margaret. Tech target. H.323 definition. http://searchnetworking.techtarget.com/definition/H323, acessado em janeiro de 2016.

[43] IETF. SIP: Session Initiation Protocol. https://www.ietf.org/rfc/rfc3261.txt, acessado em janeiro de 2016

[44] Galvita Alexander. Why SIP is Better Than H.323. http://blog.trueconf.com/reviews-comparisons/why-sip-better-than-h323.html, acessado em janeiro de 2016.

[45] Bradbury, Danny. SIP can revolutionise telephony networks but must overcome interoperability issues.
http://www.computerweekly.com/feature/SIP-can-revolutionise-telephony-networks-but-must-overcome-interoperability-issues, acessado em janeiro de 2016

[46] Blue Krypt Cryptographic Key Length Recommendation.
http://www.keylength.com/en/4/, acessado em janeiro de 2016.

[47] NIST. http://www.nist.gov/itl/csd/, acessado em janeiro de 2016.

[48] NIST. Federal Information Processing Standards Publication 197, ADVANCED ENCRYPTION STANDARD (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, acessado em janeiro de 2016

[49] Daemen, Joan; Rijment, Vincent. The Rijndael Block Cipher. NIST, 2003.

[50] CouthIT. AMR-NB Overview. http://www.couthit.com/codec-amr-nb.asp

[51] Alsa Project. Advanced Linux Sound Architecture (ALSA) project homepage.
http://www.alsa-project.org/main/index.php/Main_Page, acessado em janeiro de 2016
[52] kamailio SIP server wiki. Install and Maintain Kamailio v4.2.x from GIT.
http://www.kamailio.org/wiki/install/4.2.x/git, acessado em janeiro de 2016.

Anexo I – Código da aplicação

```
#include <pjlib.h>
#include <pjlib-util.h>
#include <pjnath.h>
#include <pjsip.h>
#include <pjsip_ua.h>
#include <pjsip_simple.h>
#include <pjsua-lib/pjsua.h>
#include <pjmedia.h>
#include <pjmedia-codec.h>
#include "Pjaux.h"
#include<iostream>
using namespace std;
static pjsua_acc_id acc_id;
#define THIS_FILE "Pjaux.cpp"
#define SIP_DOMAIN "kamailio.org" // Domínio de SIP usado
#define SIP_USER "teste2" // Usuário
#define SIP_PWD "pass" // Senha do usuário

//Outro user teste1 - senha - teste2 - pass


Pjaux::Pjaux()
{
}


Pjaux::~Pjaux()
{
}

pj_status_t Pjaux::createPjsua() //Cria user agent
{
	pj_status_t status = pjsua_create();
	if (status != PJ_SUCCESS) error_exit("Error in pjsua_create()", status);
	return status;
}
```

```
void Pjaux::error_exit(const char *title, pj_status_t status)
{
	pjsua_perror(THIS_FILE, title, status);
	pjsua_destroy();
	system("PAUSE");
	exit(1);
}

void Pjaux::initPjsua() // Função que inicia o user agent
{

	//Inicia estrutura de configuração
	pj_status_t status = createPjsua();
	pjsua_config cfg;
	pjsua_config_default(&cfg);


	cfg.cb.on_incoming_call = &on_incoming_call;
	cfg.cb.on_call_media_state = &on_call_media_state;
	cfg.cb.on_call_state = &on_call_state;

	pjsua_logging_config log_cfg;
	pjsua_logging_config_default(&log_cfg);
	log_cfg.console_level = 4;


	status = pjsua_init(&cfg, &log_cfg, NULL); //Função que inicia o user agent
	if (status != PJ_SUCCESS) error_exit("Error in pjsua_init()", status);
	system("PAUSE");

	//Inicia configuração do transporte
	pjsua_transport_config tcfg;
	pjsua_transport_config_default(&tcfg);


	tcfg.port = 5080;

	//Transporte UDP
	status = pjsua_transport_create(PJSIP_TRANSPORT_UDP, &tcfg, NULL);
	if (status != PJ_SUCCESS) error_exit("Error creating transport", status);
```

```
	//Transporte TLS e suas configurações
	pj_str_t caFile = { "C:\\certs\\certs\\kamailio.org\\cert.pem", 10000 };

	tcfg.tls_setting.cert_file = caFile;
	tcfg.tls_setting.privkey_file = { "C:\\certs\\certs\\kamailio.org\\key.pem", 10000 };
	tcfg.tls_setting.password = {"SACIS", 1000};

	status = pjsua_transport_create(PJSIP_TRANSPORT_TLS, &tcfg, NULL);
	if (status != PJ_SUCCESS) error_exit("Error creating transport", status);

	status = pjsua_start();
	if (status != PJ_SUCCESS) error_exit("Error starting pjsua", status);
	system("PAUSE");

}


void Pjaux::register_on_server()  //Função para fazer o registro no servidor

{
	pj_status_t status;
	pjsua_acc_config cfg;
	pjsua_acc_config_default(&cfg);
	cfg.id = pj_str("sips:" SIP_USER "@" SIP_DOMAIN);
	cfg.reg_uri = pj_str("sips:" "192.168.56.101");
	cfg.cred_count = 1;
	cfg.cred_info[0].realm = pj_str("*");
	cfg.cred_info[0].scheme = pj_str("digest");
	cfg.cred_info[0].username = pj_str(SIP_USER);
	cfg.cred_info[0].data_type = PJSIP_CRED_DATA_PLAIN_PASSWD;
	cfg.cred_info[0].data = pj_str(SIP_PWD);
	cfg.use_srtp = PJMEDIA_SRTP_MANDATORY; //Configuração que obriga a usar SRTP
	cfg.srtp_secure_signaling = 0;
	status = pjsua_acc_add(&cfg, PJ_TRUE, &acc_id);
	if (status != PJ_SUCCESS) error_exit("Error adding account", status);

}


void Pjaux::on_incoming_call(pjsua_acc_id acc_id, pjsua_call_id call_id, pjsip_rx_data *rdata)
```

```
//Função para tratar e receber a chamada
{
	pjsua_call_info ci;
	PJ_UNUSED_ARG(acc_id);
	PJ_UNUSED_ARG(rdata);
	pjsua_call_get_info(call_id, &ci);
	PJ_LOG(3, (THIS_FILE, "Incomming call from %. *S!!", (int)ci.remote_info.slen,
ci.remote_info.ptr));

	//Resposta automatica 200/ok - futuramente pode vir por interface (botão)
	pjsua_call_answer(call_id, 200, NULL, NULL);
}

void Pjaux::on_call_state(pjsua_call_id call_id, pjsip_event *e)
//Função chamada quando o estado da chamada muda
{
	pjsua_call_info ci;
	PJ_UNUSED_ARG(e);

	pjsua_call_get_info(call_id, &ci);
	PJ_LOG(3, (THIS_FILE, "Call %d state=%.*s", call_id,(int)ci.state_text.slen,ci.state_text.ptr));
}

void Pjaux::on_call_media_state(pjsua_call_id call_id) //Função que trata mudança de estado da mídia
{
	pjsua_call_info ci;

	pjsua_call_get_info(call_id, &ci);

	if (ci.media_status == PJSUA_CALL_MEDIA_ACTIVE) {
 //Com a mídia ativa, conecta a chamada ao device de áudio

		pjsua_conf_connect(ci.conf_slot, 0);
		pjsua_conf_connect(0, ci.conf_slot);
	}
}

void Pjaux::call() //Função que realiza chamada
{
	pj_status_t status1;
```

```
        pj_str_t uri = pj_str("sips:teste1@192.168.56.101"); //Endereco de quem vai receber a chamada

        status1 = pjsua_call_make_call(acc_id, &uri, 0, NULL, NULL, NULL);
//Função que realiza a chamada
        if (status1 != PJ_SUCCESS) error_exit("Error making call", status1);
}

int main()
{
        Pjaux p = Pjaux();
        p.initPjsua();
        p.register_on_server();
        p.call();
        system("PAUSE");
        return 0;
}
```

Anexo II – Instalação do Servidor SIP

Para instalar o servidor de SIP kamailio no Debian é necessário ter acesso root.

Os pacotes abaixo são necessários para o funcionamento do servidor. Ao lado de cada pacote está o comando usado no gerenciador de pacotes do Debian para baixá-lo.

- git client: apt-get install git
- gcc compiler: apt-get install gcc
- flex - apt-get install flex
- bison - apt-get install bison
- libmysqlclient-dev - apt-get install libmysqlclient-dev
- make - apt-get install make
- libssl - apt-get install libssl-dev
- libcurl - apt-get install libcurl4-openssl-dev
- libxml2 - apt-get install libxml2-dev
- libpcre3 - apt-get install libpcre3-dev

Em seguida, é necessário criar um diretório onde os arquivos serão armazenados.

```
mkdir -p /usr/local/src/kamailio-4.2

cd /usr/local/src/kamailio-4.2
```

Para baixar o kamailio pelo GIT, são usados os seguintes comandos.

```
git clone --depth 1 --no-single-branch git://git.sip-router.org/kamailio kamailio

cd kamailio

git checkout -b 4.2 origin/4.2
```

Caso a versão do git não suporte o parâmetro –***no-single-branch***, basta removê-lo.

O passo seguinte é compilar os arquivos de configuração,

```
make cfg
```

E ativar o modulo MySQL. Para isso, edita-se o arquivo **modules.lst**

```
nano -w modules.lst
```

Adicionar **db_mysql** à variável **include_modules**.

```
include_modules= db_mysql
```

Em seguida, é necessário salvar o arquivo e sair.

Depois disso, é hora de compilar o kamailio:

```
make all
```

Após compilado, o servidor pode ser instalado com o seguinte comando:

```
make install
```

Os binários e executáveis, listados abaixo, foram instalados em:

```
/usr/local/sbin
```

- kamailio - servidor SIP Kamailio
- kamdbctl - script para criar e gerenciar bancos de dados.
- kamctl - script para gerenciar e controlar o Kamailio.
- kamcmd - CLI – ferramenta de linha de comando para se comunicar com o Kamailio.

Para poder usar os binários da linha de comando, o local onde foram instalados deve estar na variável de ambiente *PATH*. É possível checar essa informação com o comando '*echo $PATH*'. Caso o local não esteja no *PATH* e *bash* esteja sendo usado, é preciso abrir o arquivo '/root/.bash_profile' e adicionar no fim:

```
PATH=$PATH:/usr/local/sbin

export PATH
```

Os módulos do Kamailio se encontram em:

```
/usr/local/lib/kamailio/modules/
```

Em sistemas 64 bits, o local pode ser /usr/local/lib64.

Documentação e arquivos *readme* estarão em:

```
/usr/local/share/doc/kamailio/
```

Páginas *man* foram instaladas em:

```
/usr/local/share/man/man5/

/usr/local/share/man/man8/
```

O arquivo de configuração foi instalado em:

```
/usr/local/etc/kamailio/kamailio.cfg
```

O próximo passo é a criação da base de dados MySQL. Para isso é necessário usar o *script* para *setup* da base de dados. Primeiro, editamos o arquivo ***kamctrlc*** para especificar o tipo de servidor da base de dados.

```
nano -w /usr/local/etc/kamailio/kamctlrc
```

Encontrar a variável DBENGINE e defini-la como MYSQL.

```
DBENGINE=MYSQL
```

Depois de modificar o arquivo, é preciso rodar o *script* para criar a base de dados usada pelo kamailio

```
/usr/local/sbin/kamdbctl create
```

O script vai criar uma base de dados chamada Kamailio, que contém as tabelas necessárias para o servidor, e dois usuários:

- **kamailio** - (senha padrão 'kamailiorw') – usuário com acesso total à base de dados.

- **kamailioro** - (senha padrão 'kamailioro') – usuário com acesso somente de leitura à base de dados.

As senhas podem ser alteradas no arquivo kamctlrc.

É preciso editar o arquivo de configuração para utilizar VoIP e para ativar o uso do MySQL. As instruções para essas alterações estão nos comentários do arquivo.

```
/usr/local/etc/kamailio/kamailio.cfg
```

Se a senha dos usuários do MySQL foi alterada, é necessário atualizar os valores do parâmetro '*db_url'* no arquivo de configuração.

Para iniciar/parar o servidor de uma forma mais simples, pode ser usado o *script* init.d. Um exemplo desse *script* pode ser encontrado em:

```
/usr/local/src/kamailio-4.2/kamailio/pkg/kamailio/deb/debian/kamailio.init
```

Basta copiar o arquivo para */etc/init.d/kamailio* e mudar as permissões:

```
cp /usr/local/src/kamailio-4.2/kamailio/pkg/kamailio/deb/debian/kamailio.init /etc/init.d/kamailio

chmod 755 /etc/init.d/kamailio
```

E editar os arquivos, atualizando os valores das variáveis $DAEMON e $CFGFILE:

```
DAEMON=/usr/local/sbin/kamailio

CFGFILE=/usr/local/etc/kamailio/kamailio.cfg
```

Também é necessário copiar um arquivo de configuração para */etc/default/ directory*. O arquivo a ser copiado pode ser encontrado em:

```
/usr/local/src/kamailio-4.2/kamailio/pkg/kamailio/deb/debian/kamailio.default
```

Depois de copiado, o arquivo deve ser renomeado para 'kamailio' e o valor da variável RUN_KAMAILIO deve ser alterado para YES.

É preciso criar um diretório para o arquivo pid:

```
mkdir -p /var/run/kamailio
```

A configuração padrão é para rodar o servidor com o usuário "kamailio" e grupo "kamailio". É necessário criar o usuário:

```
adduser --quiet --system --group --disabled-password \

        --shell /bin/false --gecos "Kamailio" \

        --home /var/run/kamailio kamailio



# set ownership to /var/run/kamailio

chown kamailio:kamailio /var/run/kamailio
```

Com isso, é possível iniciar/parar o servidor com os seguintes comandos:

```
/etc/init.d/kamailio start

/etc/init.d/kamailio stop
```

Com tudo preparado, agora é possível iniciar o serviço VoIP, criando as contas de usuários.

Uma nova conte pode ser criada usando a ferramenta ‘kamctl’ através do comando 'kamctl add <username> <password> <email>'.

```
kamctl add teste1 senha teste1@kamailio.org
```

Se a variável de ambiente SIP_DOMAIN for solicitada, executar uma das duas opções abaixo:

```
1. export SIP_DOMAIN=mysipserver.com

2. edit '/usr/local/etc/kamailio/kamctlrc' (or per user file, like '/root/.kamctlrc') and add:

SIP_DOMAIN=mysipserver.com
```