Centro Universitário de Brasília
Instituto CEUB de Pesquisa e Desenvolvimento - ICPD

FABRÍCIO SANTOS CARDOSO

SEGURANÇA EM STREAMS MULTIMÍDIA

Brasília
2013

FABRÍCIO SANTOS CARDOSO

SEGURANÇA EM STREAMS MULTIMÍDIA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/ICPD) como pré-requisito para obtenção de Certificado de Conclusão de Curso de Pós-graduação *Lato Sensu* em Redes de Computadores com Ênfase em Segurança

Orientador: Prof. MSc. Marco Araújo

Brasília
2013

FABRÍCIO SANTOS CARDOSO

SEGURANÇA EM STREAMS MULTIMÍDIA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/ICPD) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Pós-graduação *Lato Sensu*.

Orientador: Prof. MSc. Marco Araújo

Brasília, ___ de _____ de 2013.

Banca Examinadora

_____

Prof. Dr. Nome completo

_____

Prof. Dr. Nome completo

À Deus, por me ensinar a me tornar um ser humano melhor a cada dia.

AGRADECIMENTOS

Agradeço a Deus, por toda a paciência e persistência que me foram gentilmente cedidas.

Agradeço ao meu orientador, Marco Antônio, por todo o conhecimento dispensado.

Agradeço a minha família e namorada, pelo apoio e carinho ilimitados.

“A desconfiança é a mãe da segurança.”
Madeleine Scudéry

RESUMO

O presente trabalho propõe o estudo da segurança da informação aplicada aos fluxos multimídia, ou *streams*. Estes são parte considerável do tráfego de internet nos dias atuais, seja em forma de áudio, seja em vídeo. Com esse pensamento, buscou-se desenvolver um método que possa assegurar a devida autenticação e transmissão segura do fluxo pela internet, meio naturalmente inseguro. O trabalho capturou a história e desenvolvimento dos fluxos e como são formados. Para a segurança, foram utilizados vários conceitos de segurança da informação, tais como a criptografia, os certificados e as assinaturas digitais. Com esses conceitos consolidados, partiu-se para a busca de ferramentas que atendessem a uma solução totalmente *open source.* O resultado foi um conjunto de ferramentas que trabalham em função da autenticação do cliente que vai utilizar o fluxo, bem como o servidor, e a posterior transmissão segura das informações, caso haja sucesso na autenticação. A união de todo o conhecimento adquirido ao longo do curso propiciou a elaboração de uma solução que atende não somente aos fluxos multimídia, mas também a várias outras aplicações que necessitem de segurança da informação para seu efetivo funcionamento.

Palavras-chave: Fluxos multimídia. Segurança da informação. Código aberto.

ABSTRACT

This work suggests a study of information security applied to multimedia streams. These are a considerable part of internet traffic nowadays, either them being audio or video. With that in mind, we seeked to develop a method that could assure proper authentication and safe transmission of the stream through the internet, a naturally unsafe environment. This work captured the history and development of multimedia streams and how they are formed. For the sake of safety, various information security concepts were used, such as cryptography, digital certificates and digital signatures. With those concepts internalized, the objective became to search for tools that could enable a completely open source solution. The result was a set of tools that work to authenticate the client, server and further transmit information safely, if authentication was successful. The union of all the knowledge obtained throughout the course enabled the creation of a solution that not only supports multimedia streams, but also a pack of other applications that might need information security to their effective behavior.

Key words: Multimedia streams. Information security. Open source.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AC – Autoridade Certificadora

ACT – Autoridade de Carimbo de Tempo

ADSL – Asymmetric Digital Subscriber Line

AM – Amplitude Modulation

AR – Autoridade de Registro

BSD – Berkeley Software Distribution

DPC – Declaração de Prática de Certificado

FM – Frequency Modulation

FTP – File Transfer Protocol

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

ICP – Infraestrutura de Chaves Públicas

IP – Internet Protocol

LCR – Lista de Certificados Revogados

ITU-T – International Telecommunication Union for Telecommunication

MD5 – Message-Digest Algorithm 5

PHP – Hypertext Preprocessor

PKCS – Public-key Cryptography Standards

RSA – Rivest Shamir Adleman

RTP – Real-Time Transport Protocol

RTCP – Real Time Transport Control Protocol

RTSP – Real Time Streaming Protocol

SHA – Secure Hash Algorithm

SSL – Secure Sockets Layer

TCP – Transport Control Protocol

TLS – Transport Layer Security

UDP – User Datagram Protocol

URL – Uniform Resource Locator

W3C – World Wide Web Consortium

XOR – Ou Exclusivo

SUMÁRIO

INTRODUÇÃO

O ser humano, desde seu princípio, busca comunicar-se com seus pares e a natureza, para melhor entender o mundo que o cerca e evoluir. A partir dessa necessidade de interação, surgiram diversas formas de comunicação, seja por intermédio da voz e audição – áudio –, seja por intermédio da visão – vídeo. Com o passar dos anos, essa comunicação se desenvolveu de forma surpreendente e deu origem a alguns meios bastante conhecidos, tais como o rádio, a televisão e a internet. Esta, em especial, habilitou um *boom* na disseminação de informação de várias formas e com amplo alcance, pois, como uma rede de computadores de âmbito global, alcançaria diversas pessoas em diferentes países.

No início da era de popularização da internet, em meados da década de 90, as informações disponibilizadas eram, basicamente, texto e imagens organizados nos poucos sítios disponíveis. Em pouco tempo, a evolução da grande rede de computadores se deu, entre outros tipos, por meio da inserção de conteúdos multimídia em suas páginas. Dessa forma, surgiram os primeiros fluxos – *streams* – digitais de áudio e vídeo, com o propósito de se publicar conteúdo digital de uma forma dinâmica, rápida e mais próxima do que o rádio e televisão oferecem.

A ideia inicial seria a de prover um modo de se acessar o conteúdo radiofônico ou televisivo sem o uso dos aparelhos físicos, de forma alternativa e, em vários casos, com melhor qualidade. Um exemplo dessa tecnologia são as rádios digitais, *streams* de áudio que, quando acessadas por meio de um *link*, como uma URL, proporcionam a cobertura de uma estação de rádio que está disponível nas frequências AM ou FM, tradicionais em rádios analógicos.

Entretanto, há casos em que esses fluxos digitais podem ser desejáveis apenas para acesso restrito. É o caso, por exemplo, de uma rádio ou canal de vídeo digital que possui conteúdo exclusivo e de maior qualidade que as demais. Eles poderiam cobrar pelo serviço diferenciado que oferecem. Outra aplicação seria no ensino à distância, que permitiria o acesso às aulas em áudio ou vídeo, sejam elas gravadas ou transmitidas ao vivo, apenas a professores, tutores e alunos.

Com essas implicações em mente, o presente estudo tem como objetivos compreender como se dá o processo da segurança em *streams* e propor uma

solução para o problema. Como objetivo específico, será explicado o que são os *streams*, como a segurança da informação pode lhes ser aplicada e será desenvolvida uma solução segura de transmissão de fluxos que exemplifique os conceitos apresentados.

Como metodologia concebida para alcançar esses objetivos, procedeu-se da seguinte maneira: realizou-se, primeiramente, uma pesquisa sobre quais ferramentas poderiam ser utilizadas para o desenvolvimento de baixo custo da solução de segurança. Os aplicativos selecionados foram o *Nginx* – servidor *Web* e *proxy* reverso que habilita a criação de empacotadores *Secure Sockets Layer* – SSL – e o *Icecast* – ferramenta servidora de *streams* de áudio e vídeo.

Para os resultados, espera-se demonstrar com este estudo a importância da segurança em *streams*. Nos dias atuais, o uso de fluxos multimídia é constante. Com o rápido avanço nas tecnologias de transmissão de dados, as redes estão cada vez mais capazes de suportar esse tipo de tráfego. Entretanto, a geração de conteúdo multimídia demanda tempo e dinheiro. Há diversas empresas que trabalham com o ramo de transmissões de áudio em fluxo, tais como o *last.fm*, da CBS *Interactive*. Essas empresas trabalham com fluxos diferenciados. Empresas que desejam obter sucesso comercial ou sigilo confiável devem considerar o uso de serviços diferenciados na transmissão digital multimídia, seja por meio do ganho de qualidade ou pelo tipo de informação propagada. A segurança da informação é, nesse caso, um poderoso aliado para alcançar esses objetivos.

O presente trabalho está, então, organizado em três capítulos.

No primeiro capítulo, é apresentado o conceito de *streams* multimídia. São apresentados o seu histórico, aplicações, quais são os seus tipos e como se dá a sua formatação. Ademais, são explicados os conceitos de captura e codificação do fluxo, bem como uma breve elucidação dos tocadores – *players* – que habilitam a utilização de um *stream*. O segundo capítulo proporciona uma análise sobre a segurança da informação aplicada aos fluxos. São tratados assuntos como o *proxy*, o protocolo SSL e os certificados digitais. No terceiro capítulo, apresentamos como estudo de caso uma solução de segurança implementada com as ferramentas selecionadas. Versaremos sobre a configuração feita em cada uma delas, a execução de testes para verificar o seu funcionamento e os resultados obtidos.

1 STREAMS MULTIMÍDIA

Neste capítulo, serão apresentados os aspectos referentes aos conceitos sobre fluxos multimídia. Começaremos com seu histórico. Em seguida, versaremos sobre os tipos existentes. Para os aspectos técnicos, focaremos em técnicas de formatação do conteúdo, como se dá sua captura e codificação. Ao fim do capítulo, mostraremos alguns tocadores que efetuam sua reprodução.

1.1 Histórico

No início dos anos 20, *George O. Squier* desenvolveu a primeira rede de transmissão e distribuição de sinais de áudio a partir das linhas de energia elétrica existentes na época. Isso caracterizou o primeiro registro de uma transmissão de *streaming*, e foi batizada de *muzak* (STREAMING, s/d).

Com o passar dos anos, várias tentativas de se transmitir sinais de áudio em fluxo foram feitas. No início da década de 90, uma tecnologia de redes de comunicação chamada *Mbone* foi concebida. Ela trabalhava com acesso *multicast* sobre a internet e utilizava equipamentos específicos. Essa solução foi utilizada para a transmissão de um *show* da banda *Rolling Stones*, em 1994, considerado o primeiro uso significativo da tecnologia de *streaming* realizado sobre redes IP (MBONE, s/d).

Essas tentativas encontraram um de seus picos em 1995, com a criação da *RealAudio* pela *Real Networks* (GIROD, s/d). Uma solução completa foi elaborada pela empresa, o que incluía dos servidores que configuravam as transmissões aos tocadores. A primeira transmissão foi a de um jogo de beisebol entre os times *New York Yankees* e *Seattle Mariners*, naquele mesmo ano. Em 1997, ela criou a primeira solução de *streaming* de vídeo do mercado. Até o início dos anos 2000, a *Real Networks* dominou o mercado de *streaming*, com mais de 85% de participação. Entretanto, não conseguiu manter o sucesso, pois sua solução para criação do fluxo era paga (STREAMING, s/d).

Em 1996, a *Microsoft* entrou no mercado com o anúncio da tecnologia *NetShow* (GIROD, s/d). Essa solução evoluiu para a W*indows Media,* que contemplava a transmissão de *streaming* de áudio e vídeo. A partir dos anos 2000, a empresa passou a distribuí-la em todos os sistemas operacionais que disponibilizava de forma gratuita, o que ocasionou grande crescimento no uso (STREAMING, s/d). Em meados dos anos 2000, o mercado de streaming já estava dominado pela *Microsoft*. Entretanto, a interface de seus produtos não era personalizável, o que restringiu o seu alcance primariamente para computadores que possuíam o *Windows* instalado.

A *Apple* também investiu no mercado de *streaming.* Sua solução, chamada *QuickTime*, já existia à época da *RealAudio*, mas possuía foco no conteúdo multimídia feito para CD-ROMs (GIROD, s/d). Em 1999, na versão 4.0, foi introduzido o suporte a fluxos *(*QUICKTIME, s/d*)*. A empresa disponibiliza o produto de forma gratuita, tanto para produção quanto para reprodução. O *Quicktime* sofreu da mesma limitação do *Windows Media* e não podia ser personalizado para diferentes plataformas.

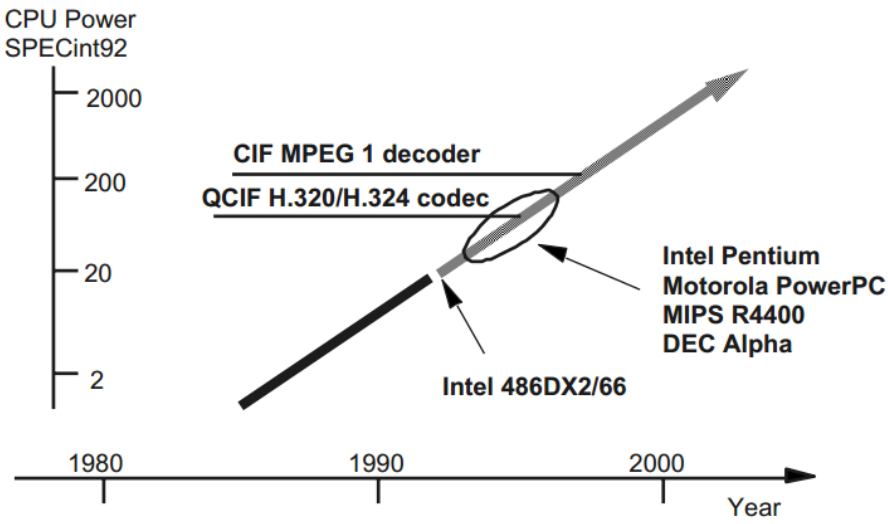
A empresa *Macromedia* desenvolveu o *Flash,* aplicação de criação de conteúdo animado e reprodução de conteúdo multimídia. Ela foi, em 2005, comprada pela empresa *Adobe*. Essa solução foi a primeira a permitir grande personalização, o que a tornou capaz de ser executada em todos os sistemas operacionais. Dessa forma, foi amplamente adotada pela indústria de entretenimento. A qualidade das transmissões era superior, em muitos casos, às soluções da *Microsoft* e da *Apple* (OZER, s/d). Em 2010, a solução era líder de mercado.

O W3C, nos dias atuais, trabalha na revisão da linguagem HTML5. Esta habilita a transmissão de *streams* de áudio e vídeo sem o uso de qualquer ferramenta adicional além do navegador. Como evolução da linguagem de criação de páginas mais utilizada na internet, o HTML, foi adotada pela *Apple* como solução padrão para a reprodução de conteúdo de fluxo em seus produtos mais recentes, o que impulsionou o uso da tecnologia ao redor do mundo (STREAMING, s/d).

Toda a história do *streaming* foi acompanhada por fatores técnicos de grande importância. Podemos, entre eles, citar o considerável aumento de poder

computacional. Na figura 1, é possível ver uma evolução do poder de processamento nos últimos 27 anos:

Figura 1 – Evolução de Processamento.



Fonte: GIROD (s/d).

Sem esses aperfeiçoamentos, não seria possível habilitar o ganho de qualidade em som e imagem observado nos fluxos atuais.

Além disso, é necessário mencionar o desenvolvimento das tecnologias de transmissão. Até o final dos anos 90, o meio de acesso à internet mais comum era o acesso discado, via modem. Esse tipo de conexão era lento e demandava inúmeros ajustes e limites para que o fluxo fosse transmitido de forma aceitável. A partir dos anos 2000, tecnologias de acesso à internet com maior velocidade, como o modem a cabo e a ADSL, possibilitaram ganhos expressivos na qualidade de transmissão, em função do aumento da banda disponível (AUSTERBERRY, 2005).

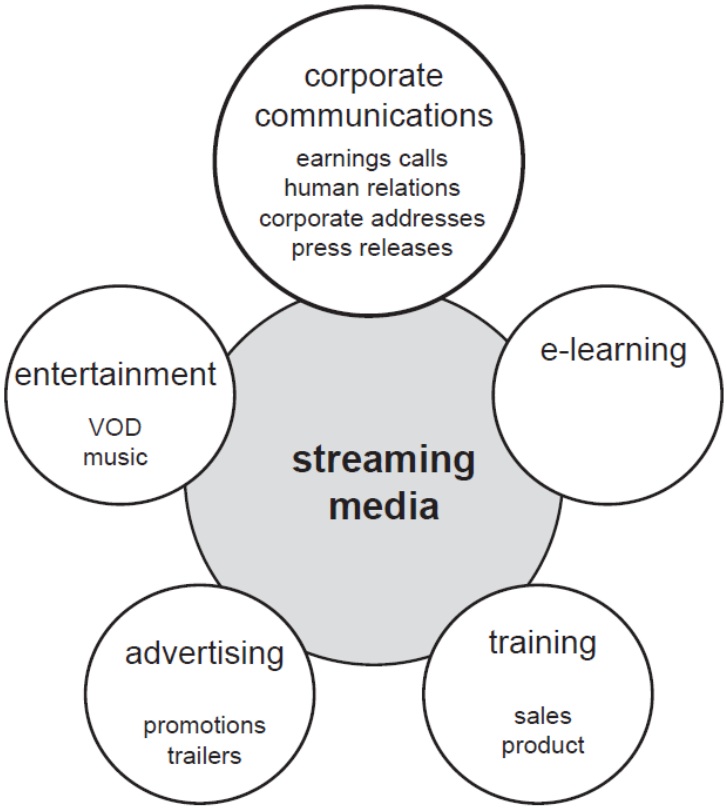
Esses avanços em processamento e transmissão, em conjunto com técnicas de codificação e compressão sofisticadas, foram os responsáveis pela atual qualidade de transmissão que é possível atestar nos dias de hoje.

A seguir, serão apresentadas as aplicações do uso de *streams* multimídia.

1.2 Aplicações

Há, no mercado, diversos usos para a tecnologia de *streaming.* Na figura 2 é possível ver uma figura com exemplos:

Figura 2 – Exemplos de uso de *streaming*.



Fonte: AUSTERBERRY (2005).

No ramo das comunicações de cunho corporativo, os fluxos podem ser úteis em reuniões, como conferências em que não seja possível estabelecer contato visual. Além disso, pode-se apresentar produtos por meio de transmissões em tempo real de feiras onde estão expostos. Ademais, um canal de comunicações pode ser criado com o uso de fluxos, para anúncios gerais internos, como vagas em determinado setor, metas e resultados da empresa, ou divulgar eventos como ações externas.

Para o mundo do entretenimento, são bastantes opções. Há a possibilidade de se transmitir músicas ou eventos sob demanda ou ao vivo, por meio de um canal de rádio ou TV que pode ser organizado em múltiplos gêneros musicais ou televisivos. O ouvinte ou telespectador seleciona ver ou ouvir apenas estilos que

gosta, a qualquer momento do dia. Da mesma forma, é possível transmitir conteúdo jornalístico ou religioso (AUSTERBERRY, 2005).

A área de publicidade e propaganda pode obter vantagens com o uso da tecnologia. Há pesquisas que mostram que os usuários que utilizam a tecnologia de *streaming* consomem mais quando estão online. Os produtos a serem ofertados podem variar. Exemplos são um comercial de um produto de limpeza, um *trailer* de algum filme famoso ou anúncios sobre novos trabalhos musicais de um artista (AUSTERBERRY, 2005).

As áreas de treinamento das empresas podem oferecer diversos cursos em áudio para os interessados, sobre os mais diversos assuntos. Com o auxílio de *streams* em vídeo, a experiência pode ser ainda mais imersiva (AUSTERBERRY, 2005).

1.3 Arquitetura

Há, basicamente, quatro elementos que compõem a estrutura de funcionamento de um serviço de *streaming* (AUSTERBERRY, 2005). São eles:

- Captura e Codificação
- Serviço
- Distribuição e Entrega
- Tocador de mídia

1.3.1 *Captura e Codificação*

A etapa de captura e codificação consiste em obter o áudio ou vídeo a partir do microfone, da mesa de som ou da câmera, gravá-lo em um arquivo de computador, comprimi-lo e codificá-lo. A captura é feita por meio de dispositivos físicos específicos, como placas instaladas em computadores que possuem diversas

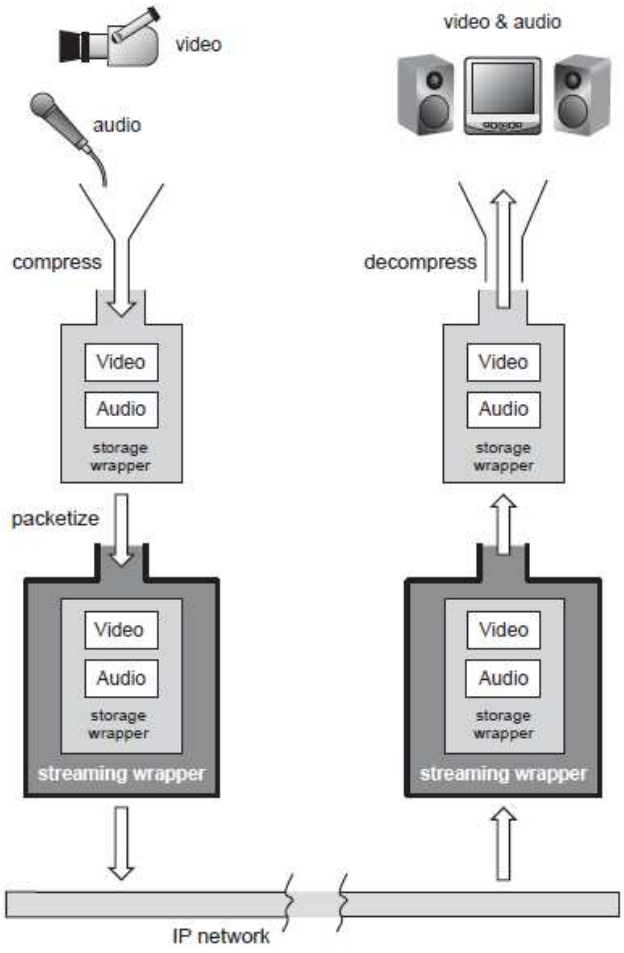
entradas para áudio e vídeo. Conecta-se os dispositivos de entrada a elas e efetua-se a gravação por meio do *software* adequado (AUSTERBERRY, 2005).

Em seguida, comprime-se o arquivo por meio do uso de compressores-decompressores, ou *codecs*. Estes reduzem o tamanho do arquivo com o objetivo de não haja perda considerável de qualidade. Além disso, esse passo é necessário para adequar o arquivo ao tamanho da banda disponível para transmissão.

Ao final, insere-se uma camada de identificação do fluxo. Em geral, são gravados metadados que carregam informações como a duração, número de partes e qualidade. Essa camada é chamada de codificação, ou *wrapping* (AUSTERBERRY, 2005).

Na figura 3 é possível ver os passos descritos naquele elemento da arquitetura:

Figura 3 – Captura e codificação.



Fonte: AUSTERBERRY (2005).

1.3.2 *Serviço*

O arquivo gerado é armazenado em um servidor de conteúdo, um computador que armazena arquivos de mídia. Ele é responsável pela distribuição do conteúdo pela rede IP. Essa distribuição é regulada, a fim de que o fluxo esteja alinhado às restrições da banda disponível. Além disso, ele controla os pedidos de alteração na reprodução para trás ou para frente, ou seja, *fast forwarding* – FF – e *rewinding* – RWD, bem como pausas (AUSTERBERRY, 2005).

1.3.3 *Distribuição e Entrega*

Para que o *stream* alcance o usuário, é necessário um canal de entrega e distribuição. Como regra geral, é necessária apenas a conectividade entre o servidor de conteúdo e o tocador de mídia. Entretanto, a internet não foi desenvolvida especificamente para esse tipo de transmissão. Logo, problemas como perda de qualidade e pausas inesperadas podem ser comuns. Para solucionar isso, são necessários algoritmos específicos para correção de erros e regulagem do fluxo.

Em casos de transmissões com grande demanda, é necessária a criação de uma rede específica para a distribuição, o que implica novos servidores e infraestrutura de rede dedicada. Pode-se, ainda, utilizar a rede chamada *Content Delivery Network* – CDN. Ela foi baseada na criação de uma camada de abstração sobre a internet que está preparada para a transmissão de conteúdo multimídia em fluxo (AUSTERBERRY, 2005).

Além das características acima, o aumento na banda de dados das conexões, conforme mencionado, contribuiu para melhorar a qualidade da transmissão e potencializar o serviço de entrega e distribuição.

1.3.4 *Tocador de Mídia*

A última parte da arquitetura de transmissão de *streams* representa a interface do usuário com a tecnologia. O tocador de mídia – ou *player* –, em geral, é

um aplicativo específico para reprodução de áudio e vídeo, seja em forma de *stream*, seja para reprodução de arquivos multimídia, como o mp3, *wave* ou formatos de vídeo. Várias empresas disponibilizam esses produtos, tais como a *Microsoft*, com o *Windows Media Player*, a *Apple*, com o *Quicktime*, e a *Nullsoft*, com o *Winamp*. Na figura 4 é possível ver a interface do *Winamp*:

Figura 4 – Interface do *Winamp*.

Fonte: o autor.

Além dos tocadores disponíveis para PCs, há também os desenvolvidos para dispositivos portáteis, como celulares, *tablets* e mp3 *players*. Outros dispositivos também podem fazer o uso de *streaming*, como os *set-top-boxes*, aparelhos de recepção de TV digital.

1.4 Protocolo RTP

Para melhor elucidação dos conceitos de *streaming*, apresentaremos o protocolo RTP e protocolos auxiliares.

O servidor de um *streaming* é, basicamente, um servidor *Web*. Todavia, deve possuir algumas particularidades, tais como (AUSTERBERRY, 2005):

- Controle de fluxo em tempo real;
- Chaveamento de fluxo inteligente;
- Navegação interativa.

Para que esses requisitos sejam satisfeitos, o *Internet Engineering Task Force* – IETF criou alguns protocolos para transmissão do fluxo, pois o protocolo HTTP não é capaz de satisfazer esses requisitos por conta própria. São eles:

- *Real-time Transport Protocol* – RTP;
- *Real-time Control Protocol* – RTCP;
- *Real-time Streaming Protocol* – RTSP.

O RTP é o protocolo responsável pelo transporte de dados em tempo real do *stream* multimídia de forma fim-a-fim, tanto para aplicações *unicast* quanto *multicast* (DUARTE, s/d). É nele que serão trafegados os pacotes de áudio e vídeo. Suas funções são, além do transporte, fornecer carimbos de tempo, criar números de sequência e determinar o tipo de codificação utilizada no *stream*. Ele trabalha com portas de número par e, em geral, utiliza o protocolo UDP, pois este privilegia a continuidade do fluxo (REAL-TIME, s/d).

Para cada transmissão é criada uma sessão. Ela consiste no endereço IP de destino do servidor e um par de portas, uma para o RTP e outra para o RTCP, que será tratado adiante. A negociação de portas é feita com o uso do RTSP. De acordo com a RFC 3550, a porta do RTCP será a porta ímpar seguinte à porta escolhida para o RTP (REAL-TIME, s/d).

O protocolo RTP foi concebido para transportar qualquer formato de áudio ou vídeo. As informações que determinam seu formato estão definidas em especificações chamadas *profiles*. Estas determinam o valor do campo *Payload Type* – PT, incluso no cabeçalho RTP, que é o tipo do pacote de áudio a ser trafegado, como, por exemplo, PCM, LPC ou MPA, formatos sem compressão. Além desses, outros *profiles* já foram estabelecidos, tais como o *Secure Real-time Transport Protocol* – SRTP, que provê criptografia ao *stream*. Ele está definido na RFC 3711.

Na figura 5, é possível ver o cabeçalho do protocolo RTP:

Figura 5 – Cabeçalho RTP.

RTP packet header

bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	P	X	CC	M	PT	Sequence Number
32	Timestamp						
64	SSRC identifier						
96	CSRC identifiers ...						
96+32×CC	Profile-specific extension header ID						Extension header length
128+32×CC	Extension header ...						

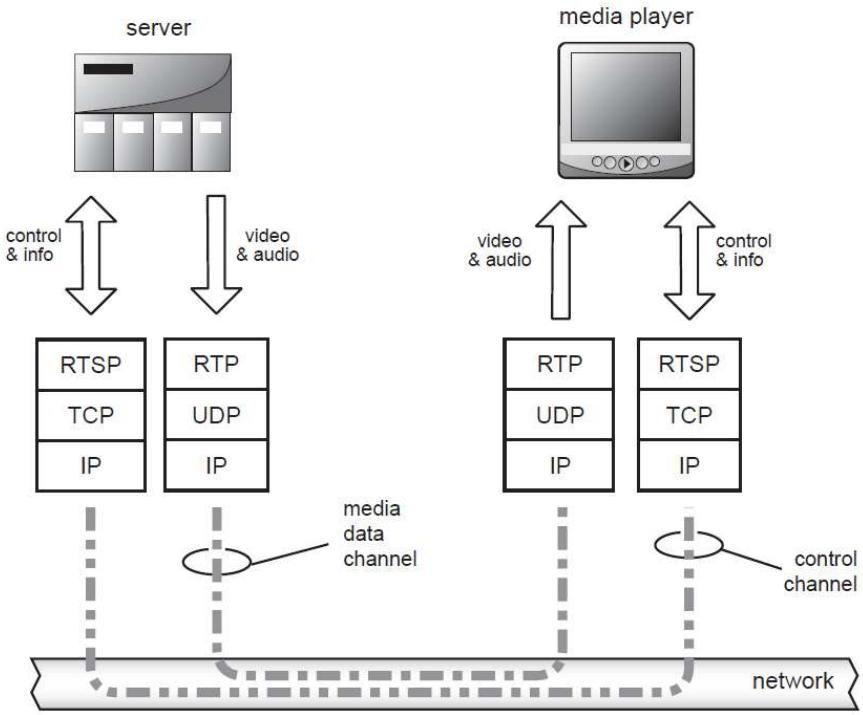
Fonte: REAL-TIME (s/d).

O protocolo RTCP é responsável pelos atributos relativos à qualidade de serviço – QoS e controle do fluxo. Ele coleta estatísticas sobre a qualidade da transmissão e remete ao servidor e demais clientes, caso solicitado. Isso é necessário para adequar o fluxo. Caso a banda disponível seja sobrecarregada, esses valores podem ser renegociados com o servidor e uma nova taxa de transmissão será escolhida, o que acarretará em queda de qualidade. Caso contrário, a mesma negociação poderá elevar a taxa de transmissão e aumentar a qualidade do serviço (REAL-TIME, s/d).

O protocolo RTSP é um protocolo *stateful* – orientado a estados – e foi concebido especialmente para controlar a transmissão de fluxos. Ele habilita funções como o FF, RWD e PAUSE, para que o usuário possa navegar pelo fluxo. Além disso, pode trabalhar com o RTCP e realizar o controle do fluxo mediante troca de informações estatísticas entre o *player* e o servidor. Como no HTTP, ele mantém uma conexão estabelecida e utiliza o TCP para troca de dados (SCHULZRINNE, s/d).

Pode-se ver, na figura 6, um diagrama com o funcionamento dos protocolos explicados:

Figura 6 – Funcionamento dos protocolos RTP de *streaming*.



Fonte: AUSTERBERRY (2005).

1.5 Protocolo Shoutcast/Icecast

O protocolo a ser utilizado neste trabalho para a transmissão dos fluxos é o *Icecast*, que é baseado no protocolo *shoutcast*, concebido pela *Nullsoft*. Seu funcionamento está largamente baseado no HTTP.

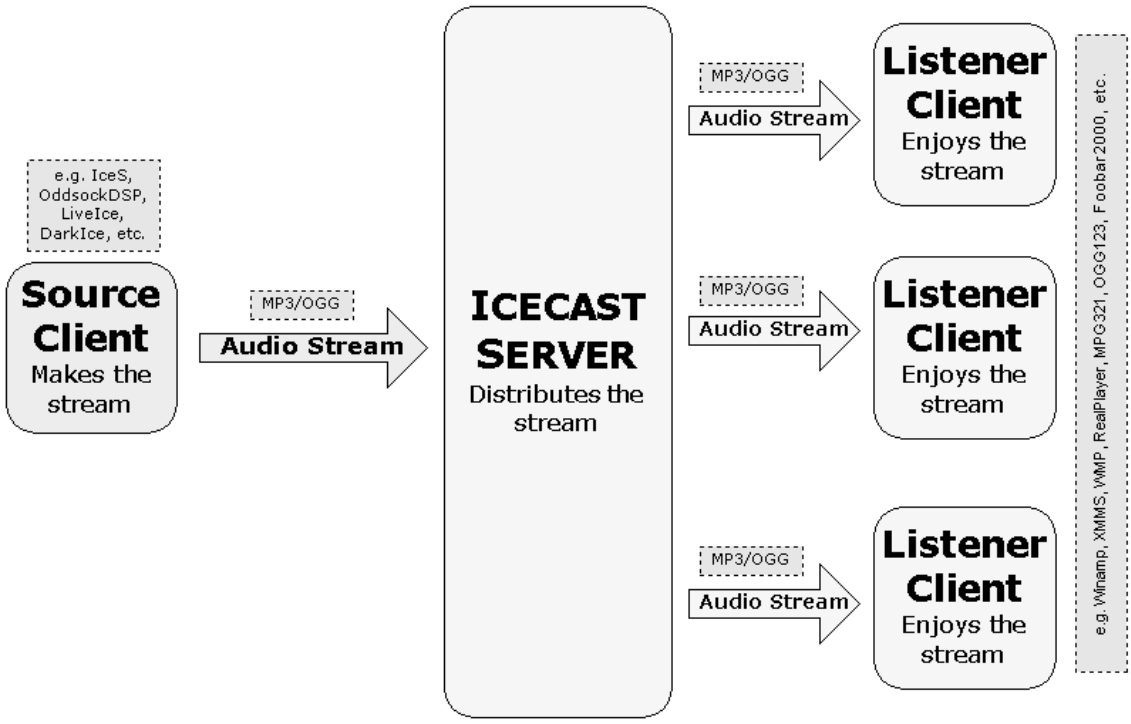
A arquitetura de funcionamento é dividida em três partes (JAY, s/d):

- Fonte – em geral, uma aplicação que gerencia uma câmera, microfone ou uma lista de reprodução de arquivos de mídia;
- Servidor – elemento responsável por despachar o que a fonte entrega e convertê-lo em fluxo para o cliente;

- Cliente – usado para escutar ou visualizar o conteúdo multimídia a partir do servidor.

Na figura 7, é possível ver o esquema de funcionamento do protocolo *Shoutcast/Icecast*:

FIGURA 7 – Funcionamento do *Icecast.*



Fonte: ICECAST (s/d).

Para que o servidor permita conexões requisitadas pelo cliente, ele precisa de uma fonte. Quando essa conexão é estabelecida, o servidor realiza o trabalho de passar os dados em fluxo da fonte para o cliente. A sequência de passos entre o servidor e o cliente pode ser vista a seguir (JAY, s/d):

- A fonte solicita uma conexão com a porta de serviço do servidor;
- A fonte envia a senha de conexão definida pelo servidor. Se estiver correta, o servidor responderá com uma mensagem informando que foi

autorizada a conexão e está pronto para receber os dados. Caso seja incorreta, o servidor sinaliza com uma mensagem de senha inválida;

- Se a fonte receber o OK, deve, então começar a enviar informações sobre o *stream* para o servidor.

Abaixo podemos ver um exemplo de código de cabeçalho enviado da fonte para o servidor. Nesse exemplo, é possível observar dados a respeito do fluxo em si (JAY, s/d):

- icy-name:Unnamed Server\r\n – nome da estação de rádio;
- icy-genre:Unknown Genre\r\n – gênero dos fluxos, tal como *rock*, *jazz*, *blues*, etc.;
- icy-pub:1\r\n – define se o servidor vai publicado em algum serviço de diretório;
- icy-br:56\r\n – define a taxa de *bits* do fluxo;
- icy-url:http://www.shoutcast.com\r\n – define a *homepage* do fluxo;
- icy-irc:%23shoutcast\r\n – define informações de contato IRC;
- icy-icq:0\r\n – define informações de contato ICQ;
- icy-aim:N%2FA\r\n – define informações de contato AIM;
- content-type: mime/type\r\n – define o tipo de dados a serem trafegados, como o HTTP;
- icy-reset: 1\r\n – determina se o servidor deve limpar o *buffer;*
- \r\n - fim do cabeçalho.

A comunicação cliente servidor é feita de forma similar a de um *browser* com uma página. O cliente se conecta ao servidor e envia informações sobre os recursos que suporta. O servidor, por sua vez, responde com a seguinte sequência de cabeçalho (JAY, s/d):

- ICY 200 OK\r\n – determina o sucesso da conexão com o servidor;
- icy-notice1:
This stream requires Winamp
 – notificação de uso do tocador de mídia;
- icy-notice2:SHOUTcast Distributed Network Audio Server/posix v1.x.x
 – enuncia ao cliente qual o tipo de servidor e sua versão;
- icy-name:Unnamed Server\r\n – nome do servidor;
- icy-genre:Unknown Genre\r\n – o gênero do fluxo que o servidor entrega;
- icy-url:http://www.shoutcast.com\r\n – define a *homepage* do fluxo;
- Content-Type:audio/mpeg\r\n – tipo de conteúdo do fluxo que será enviado;
- icy-pub:1\r\n – define se o servidor está em algum serviço de diretório ou não;
- icy-br:56\r\n – taxa de bits do servidor;
- \r\n – fim do cabeçalho.

A partir do estabelecimento da conexão acima, o fluxo começa a ser transmitido.

Com os conceitos acerca do funcionamento de *streams* multimídia em mente, podemos avançar e tratar sobre como se dará uma possível solução para garantir a segurança das transmissões. A seguir, veremos os conceitos que servem como base para a segurança em *streams*.

2 SEGURANÇA EM STREAMS

O presente capítulo busca enunciar todos os conceitos de segurança envolvidos na solução proposta neste trabalho. Ela possui aspectos usuais no mercado, relacionados ao uso de *proxies*, criptografia, *hash*, assinaturas e certificados digitais, além de protocolos específicos como o SSL e o HTTPS.

2.1 Proxy

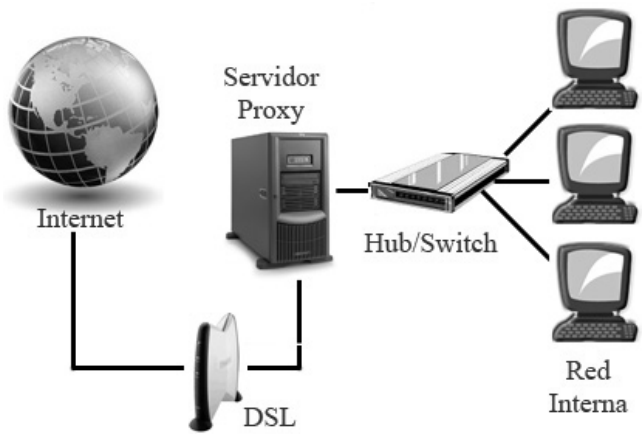
Um *proxy* pode ser definido como uma solução de segurança de redes que trata a conexão que sai de uma rede. Seu papel é capturar o pacote que trafega com algum destino externo, examiná-lo e decidir, a partir de definições preestabelecidas, tais como as determinações de uma Política de Segurança, se ele pode ser encaminhado, ou seja, direcionado para um servidor externo à rede.

De forma geral, o *proxy* atua onde um *firewall* atuaria, mas de forma complementar, entre a rede interna e a rede externa. Além disso, pode-se examinar pacotes que entram na rede, que são, em condições normais, resultados de solicitações internas (NAKAMURA, 2007).

Para o estabelecimento da comunicação, são criadas duas conexões (NAKAMURA, 2007). A primeira é a conexão feita entre o cliente e o *proxy*. A segunda é a feita entre o *proxy* e o servidor externo, caso haja permissão. Esse proxy é também definido como do tipo *forward* (APACHE, s/d).

Na figura 8 é possível ver uma figura que ilustra o funcionamento de um *forward proxy*:

Figura 8 – Servidor Proxy entre uma rede e a Internet.



Fonte: TEOTÔNIO (2010).

As principais características de um *proxy* são (NAKAMURA, 2007):

- Não permite conexões diretas entre *hosts* internos e externos;
- Permite o anonimato a usuários internos;
- Aceita autenticação de usuário;
- Avalia o conteúdo – *payload* – do pacote, o que sugere uma melhor verificação do que está sendo trafegado;
- *Caching* – armazena conteúdo que é comumente acessado para maior velocidade de entrega aos clientes, o que elimina a necessidade de acesso externo.

Entretanto, há desvantagens, tais como (NAKAMURA, 2007):

- É mais lento do que um filtro de pacotes comum, como um *firewall*;
- Pode exigir configurações específicas para cada aplicação.

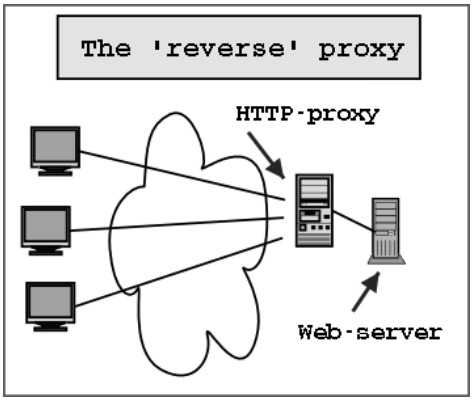
Um segundo tipo de *proxy* é o chamado de *open.* Este está baseado no conceito de acesso livre. Qualquer usuário pode obter acesso a esse servidor e seu

papel é redirecionar o cliente a qualquer outro endereço válido na Internet, o que possibilita o anonimato (PROXY, s/d).

O terceiro tipo de *proxy* é o *reverse,* ou reverso. Seu funcionamento consiste na solicitação de um acesso do cliente ao servidor *proxy*, que, por sua vez, recebe a requisição e decide, de forma autônoma, para onde redirecioná-la (APACHE, s/d).

Em geral, os reais responsáveis pelo tratamento da requisição que chega ao *proxy* são servidores *Web*. Neste trabalho, o servidor de mídia será quem receberá as requisições. Assim que a resposta do servidor *Web* for recebida, o *reverse proxy* responderá ao cliente como se a resposta fosse criada por ele próprio, o que adiciona uma camada de segurança à comunicação. Na figura 9 é possível ver uma figura que exemplifica seu funcionamento:

Figura 9 – funcionamento de um *proxy* reverso.



Fonte: VAZ (2011).

Entre as características de um *proxy* reverso, pode-se citar (PROXY, s/d):

- Criptografa as conexões solicitadas aos servidores *Web*, por meio de algoritmos criptográficos como o SSL;
- Realiza balanceamento de carga. Caso haja vários servidores *Web* que possam receber as requisições, pode-se equilibrar a distribuição de

requisições e direcionar os novos pedidos aos servidores menos atarefados;

- *Caching* – armazena conteúdo que é comumente acessado para maior velocidade de entrega aos clientes, o que elimina a necessidade de acesso externo;
- Adiciona uma camada extra de segurança à rede interna.

O *proxy* reverso é a ferramenta que será utilizada para a solução de segurança apresentada para os fluxos. Seu papel será o de receber as requisições de transmissão em formato seguro, decriptografá-las, autenticá-las e encaminhar ao servidor de fluxo.

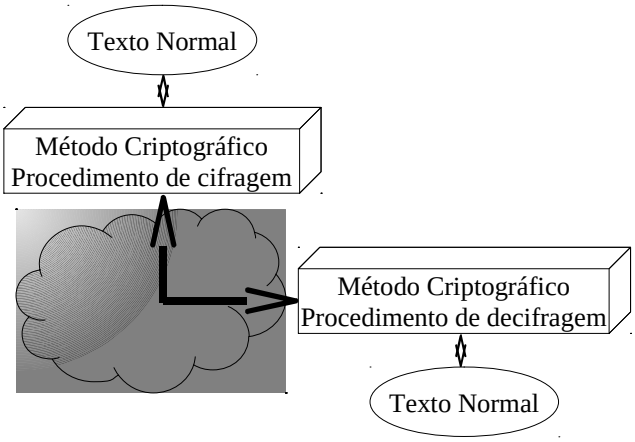
A seguir, serão apresentados os conceitos básicos que envolvem a camada de segurança dos *streams* apresentada neste trabalho.

2.2 Criptografia

A criptografia é a ferramenta usada para garantir confidencialidade e autenticação para o SSL, algoritmo de segurança deste trabalho que será explicado em detalhes mais adiante.

A confidencialidade é assegurada quando se modifica o texto original do texto às claras por meio de um algoritmo criptográfico que gera outro texto – chamado de texto cifrado, ilegível para quem não possui acesso ao segredo do método criptográfico utilizado. O texto cifrado é, então, transmitido, como pode ser visto na Figura 10. No destino, o procedimento de decifragem é realizado com o conhecimento do segredo do método criptográfico. Isso permite que o destinatário obtenha o texto original (GPR, s/d).

Figura 10 – Esquema básico do método criptográfico.



Fonte: KUROSE (2010).

Nos algoritmos criptográficos mais recentes, a segurança está associada à chave e não ao algoritmo (ARTHUR, s/d). Caso um adversário deseje quebrar o código criptográfico – ou seja, deduzir a senha utilizada – deve-se garantir que os recursos computacionais necessários para essa quebra sejam tão grandes que ela seja inviável (SCHNEIER, s/d).

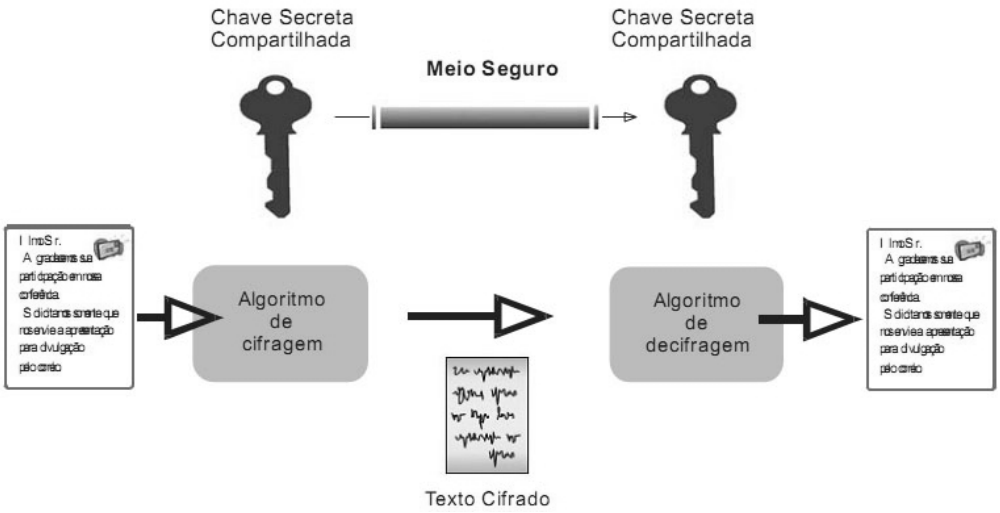
Há dois tipos básicos de algoritmos criptográficos (UOL, s/d):

- Chave secreta ou simétricos – usados para garantir a confidencialidade;
- Chave pública ou assimétricos – usados para garantir a autenticação.

2.2.1 *Criptografia Simétrica*

Os métodos criptográficos que usam a mesma chave para cifragem e decifragem são classificados como simétricos, ou baseados em chave secreta (GPR, s/d). Seu modo de funcionamento está ilustrado na Figura 11:

Figura 11 – Criptografia simétrica.



Fonte: MENDONÇA (s/d).

A criptografia simétrica pode ser de dois tipos (KUROSE, 2010):

- Cifra de bloco – cifra um bloco de tamanho fixo de bits a cada vez. É muito utilizado em protocolos como o SSL, que será tratado adiante.
- Cifra de fluxo – manipula os bits do texto normal operando-os, geralmente, com a função XOR – ou exclusivo, com bits gerados a partir de um processo pseudoaleatório.

O uso de criptografia simétrica apresenta algumas vantagens, tais como bom desempenho no processo de cifragem e decifragem. Entretanto, um problema importante no uso desse tipo de criptografia é a distribuição da chave secreta. Deve haver um meio de enviá-la de forma segura para a outra parte da comunicação, sob pena de um terceiro apoderar-se dela e conseguir se comunicar como se fosse uma das partes legítimas. Entretanto, há soluções, tais como a criptografia que será explicada a seguir.

2.2.2 *Criptografia Assimétrica*

A criptografia assimétrica é obtida a partir da utilização de um par de chaves diferentes, uma para cifragem e outra para decifragem. Elas são produzidas de tal forma que seja impossível, em termos computacionais, a obtenção de uma delas conhecendo-se apenas a outra (DIFFIE, 1976).

Respeitada essa condição, uma dessas chaves pode-se tornar pública. Assim, quem quiser enviar uma mensagem para outra pessoa deve obter a chave pública do destinatário e cifrar a mensagem com ela. Somente o destinatário, que será o exclusivo detentor da outra chave, conhecida por chave privada, será capaz de reverter o processo, e ela possibilitará a recuperação do texto em claro (SOARES, 1995).

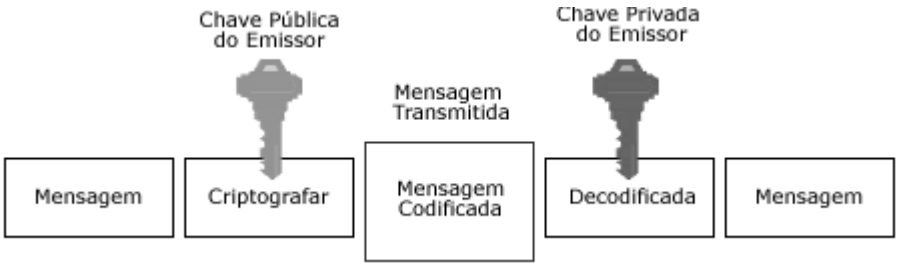
Os algoritmos assimétricos resolvem o grande problema dos simétricos mencionado no item anterior, ou seja, a troca de chaves por meio de um canal inseguro, pois só é necessária a publicação da chave pública para o receptor de mensagens seguras. Entretanto, para uma comunicação bilateral, será necessário gerar quatro chaves diferentes, ou seja, duas para um lado da comunicação, uma privada e outra pública, e outras duas para o outro lado (GPR, s/d).

Embora existam várias técnicas para geração das chaves, a maioria dos algoritmos assimétricos utiliza grandes números primos (KUROSE, 2010). A segurança, nesse caso, está na dificuldade em fatorar grandes números nos seus fatores primos (GPR, s/d).

Quanto maior o número de *bits* utilizados, mais seguros serão os algoritmos. Atualmente, é comum o uso de chaves de 1024 e 2048 bits. O grande problema em aumentar demais o tamanho da chave é o fato de tornar o processamento mais lento, o que pode gerar perda considerável de desempenho (KUROSE, 2010).

Na figura 12, pode-se visualizar o funcionamento da criptografia assimétrica:

Figura 12 – Criptografia assimétrica.



Fonte: RECKZIEGEL (2006).

As técnicas de criptografia simétrica e assimétrica podem ser utilizadas em conjunto, o que potencializa os aspectos de segurança. Esse assunto será tratado a seguir.

2.2.3 *Chave de Sessão*

Os algoritmos simétricos, em geral, possuem bom desempenho. Entretanto, a gerência de chaves pode se tornar complicada, pois a segurança do algoritmo é bastante dependente da guarda e envio da chave entre uma parte da comunicação e a outra. Os algoritmos assimétricos são, por sua vez, consideravelmente mais lentos que os simétricos. Porém, nesses a gerência de chaves é simplificada, pois o uso da chave pública para cifragem elimina o principal problema da gerência de chaves, que é a distribuição da chave secreta (KUROSE, 2010).

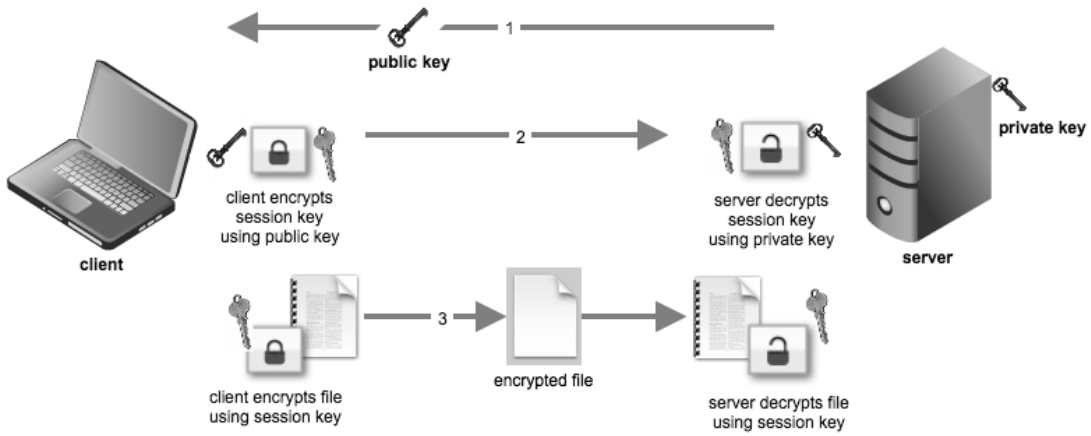
Desse modo, os algoritmos de chave pública são frequentemente usados em conjunto com os de chave simétrica. Isso torna possível somar as vantagens dos dois tipos, ou seja, efetuar a troca de chaves secretas por meio da criptografia assimétrica e, em seguida, trocar informações criptografadas com o uso da chave secreta compartilhada.

Nessa união, o remetente escolhe uma chave chamada de chave de sessão. Ele cifra essa chave com a chave pública do destinatário e a envia. O remetente deve ser o único capaz de decifrá-la com sua chave privada. A partir dessa etapa, ambos os participantes da comunicação passam a compartilhar uma

chave e algoritmos de chave simétrica podem ser usados para cifrar os dados trocados durante a sessão de comunicação.

Pode-se visualizar o funcionamento de uma chave de sessão com a figura 13:

Figura 13 – Funcionamento da chave de sessão.



Fonte: JOHN (2012).

A seguir, trataremos sobre como pode-se garantir a integridade do fluxo a ser transmitido.

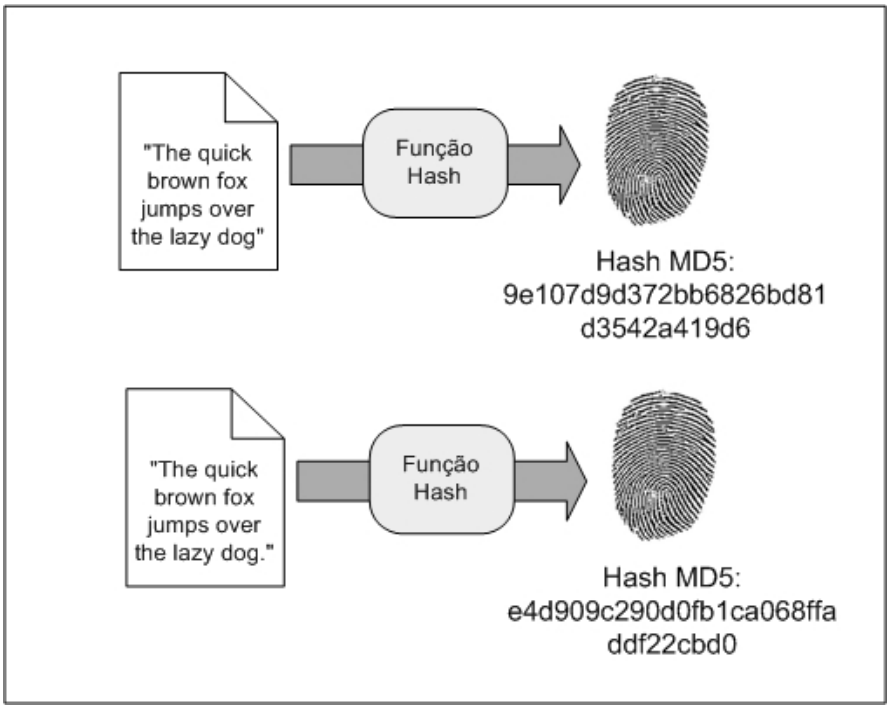
2.3 Função Hash

Funções *hash* trabalham com o texto em claro. Esse sofre uma transformação ao passar pela função e o resultado será uma mensagem de saída de tamanho fixo. Essa função pode ser usada para verificar a integridade da mensagem, pois, caso haja alguma alteração na mensagem original, o valor *hash* calculado a partir da mensagem alterada será completamente diferente do valor calculado a partir da mensagem original (KUROSE, 2010). Há diversas funções *hash*. As mais utilizadas são o MD5, algoritmo que já foi vastamente explorado, e as variações do SHA, que proporcionam melhor segurança nos resultados (BELLARE, 1996).

Entretanto, podem ocorrer problemas com o resultado dessas funções. Uma colisão ocorre quando se consegue obter, para entradas diferentes, resultados iguais. Caso ocorra, a verificação de integridade é invalidada (KUROSE, 2010).

Na figura 14 é possível visualizar um esquema com a função *hash*. Nela, a simples adição de um ponto final à frase já acarreta na total alteração do código *hash* resultante:

Figura 14 – Função *hash*.



Fonte: CARVALHO (2008).

2.4 Assinatura Digital

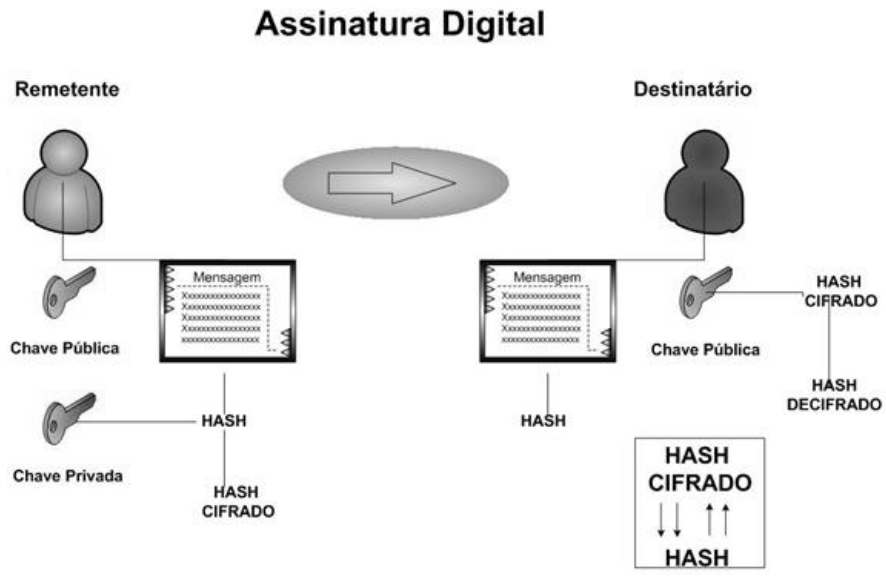
As assinaturas digitais são equivalentes, no mundo digital, às assinaturas manuais. Seu propósito é alcançar a autenticidade e, por meio da técnica escolhida, a integridade de um documento. A geração é feita com o uso da chave privada do remetente na mensagem, o que pode assegurar a autenticidade, caso permaneça em sigilo (KUROSE, 2010).

Para aliar os aspectos de integridade e autenticidade por meio do uso de assinaturas digitais, deve-se seguir alguns passos, que envolvem do momento da assinatura até seu efetivo reconhecimento.

Em princípio, gera-se o valor *hash* do documento a ser assinado e cifra-se o resultado com a chave privada do remetente. Esse passo torna mais rápido o processo, pois, se fosse necessário cifrar todo o documento, tal ação acarretaria em perda de desempenho. Se desejado, para acrescentar confidencialidade, pode-se cifrar o documento e o valor *hash* cifrado com a chave privada do remetente e, em seguida, cifrar o resultado da primeira cifragem com a chave pública do destinatário (CRISTIAN, 2007).

Após o recebimento, o destinatário decifra a mensagem recebida com sua chave privada, obtendo a mensagem e um valor *hash* cifrado com a chave privada do remetente. Ele calcula o valor *hash* da mensagem recebida, por meio do uso do mesmo algoritmo utilizado no remetente. Adiante, utiliza a chave pública do remetente no *hash* recebido. Por fim, compara o *hash* calculado com aquele obtido com a chave pública do remetente. Se não houver modificações durante o envio da mensagem, a assinatura será validada mediante a igualdade na comparação. A Figura 15 ilustra o processo de assinatura digital:

Figura 15 – Assinatura digital.



Fonte: CRISTIAN (2007).

2.5 Certificado Digital

Um dos problemas que envolvem a criptografia assimétrica é a verificação de integridade e origem de uma chave pública. Ao realizar-se uma pesquisa de uma chave numa base pública de certificados, torna-se importante saber a sua real origem. Se ela for autêntica, pode-se cifrar a mensagem sem maiores problemas. Caso não seja, corre-se o risco de alteração da mensagem enviada mediante interceptação por alguém não autorizado, o que pode trazer prejuízos.

Entre as várias formas de se verificar a origem da chave, a mais comum é o uso de certificados digitais. Estes são documentos eletrônicos que possuem informações a respeito do dono da chave pública, tais como seu número de série, nome, endereço, telefone, data de emissão do certificado e prazo de validade. Seu objetivo é atestar o dono da chave pública.

O uso de certificados digitais é melhor gerenciado se atrelado a uma estrutura de manutenção, comumente chamada de Infraestrutura de Chaves Públicas – ICP. Esta consiste em partes que serão descritas a seguir.

2.5.1 *Infraestrutura de Chaves Públicas – ICP*

O certificado é assinado por uma entidade confiável para o dono da chave, denominada Autoridade Certificadora – AC. Uma AC é uma entidade, pública ou privada, responsável por emitir, distribuir, renovar, revogar e gerenciar os certificados. Além disso, tem a responsabilidade de verificar se o titular do certificado possui a chave privada que corresponde à chave pública que faz parte do certificado. Como ações principais, realiza a criação e assinatura digital do certificado do assinante, onde o certificado emitido pela AC representa a declaração da identidade do titular, que possui um par único de chaves, ou seja, uma pública e outra privada (ITI, s/d).

Cabe à AC, ainda, emitir Listas de Certificados Revogados – LCR – e manter registros de suas operações, sempre em consonância com as práticas

definidas na Declaração de Práticas de Certificação – DPC, bem como estabelecer e fazer cumprir, pelas Autoridades de Registro – ARs a ela vinculadas, as políticas de segurança necessárias para garantir a autenticidade da identificação realizada (ITI, s/d).

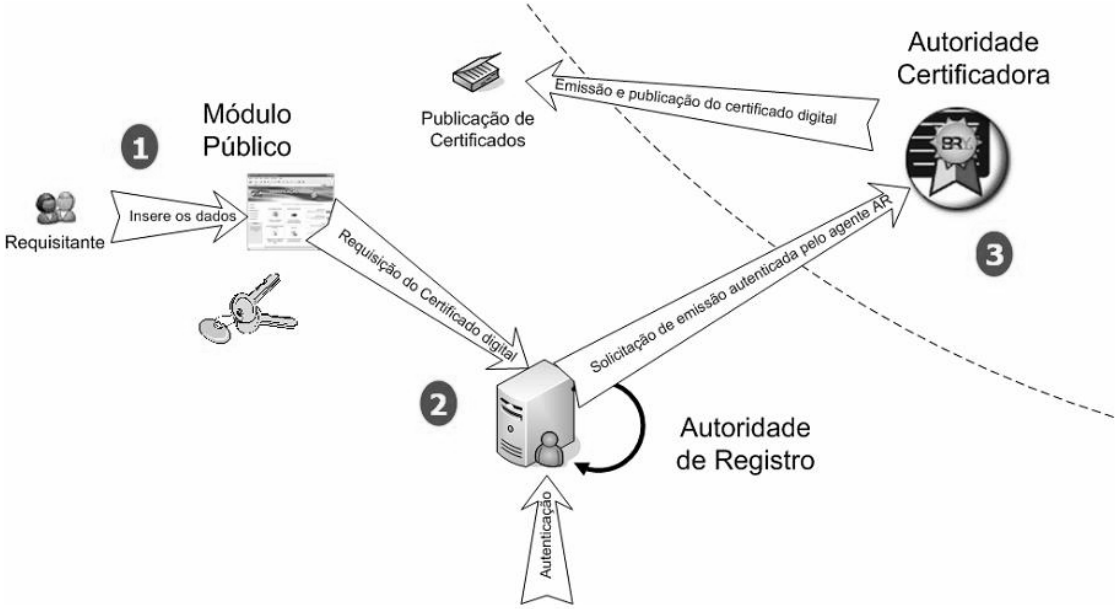
Outra parte da ICP é a Autoridade de Registro – AR. Uma AR é responsável pela interface entre o usuário e a AC. Sempre com vínculo a uma AC, a AR tem por objetivo receber, validar e encaminhar solicitações de emissão ou revogação de certificados digitais e identificar, de forma presencial, seus solicitantes. A AR deve, ainda, manter registros de todas as suas operações (ITI, s/d).

Uma parte que, por enquanto, permanece opcional, mas que é muito importante à estrutura de uma ICP é a Autoridade Certificadora do Tempo – ACT. É nela que os usuários de serviços de Carimbo do Tempo confiam para emitir carimbos de tempo. Um carimbo de tempo pode ser utilizado para a verificação de certificados expirados, por exemplo. A ACT tem como atribuição a responsabilidade pelo fornecimento de carimbos de tempo. Estes carimbos são o conjunto de atributos fornecidos pela parte confiável do tempo que, associado a uma assinatura digital, confere provar a sua existência em determinado período (ITI, s/d).

Como exemplo, podemos citar um documento que é produzido e seu conteúdo é criptografado. Em seguida, ele recebe os atributos ano, mês, dia, hora, minuto e segundo, atestado na forma da assinatura realizada com certificado digital servindo assim para comprovar sua autenticidade. A ACT atesta não apenas a questão temporal de uma transação, mas também seu conteúdo (ITI, s/d).

Na figura 16, podemos ver como uma ICP funciona no fornecimento de certificados digitais:

Figura 16 – Infraestrutura de Chaves Públicas.



Fonte: NUNES (2007).

Há vários tipos de certificados utilizados pelas ACs. Um dos mais comuns é o que está baseado no padrão X.509, que será visto a seguir.

2.5.2 *O padrão X.509*

O SSL, algoritmo de segurança utilizado neste trabalho, faz uso de certificados do padrão X.509. Criado pelo ITU-T e bastante difundido no mercado, nasceu do padrão X.500, que disciplina a construção e administração de serviços de diretório. O X.509 consiste, entre outras características, num padrão sobre o que um certificado digital deve fornecer para que seja confiável e seguro. São definidos campos obrigatórios e extensões, partes adicionais que podem ser alteradas de acordo com as necessidades do usuário.

Um certificado X.509 deve conter os seguintes campos (CERTIFICADO, s/d):

- Versão – contém a versão do certificado X.509, atualmente na versão 3;

- Número serial – todo certificado deve possuir um. Não é globalmente único, mas único no âmbito de uma AC. As LCRs o usam para apontar quais certificados se encontram revogados;
- Tipo de algoritmo – Contem um identificador do algoritmo criptográfico usado pela AC para assinar o certificado juntamente com o tipo de função de *hash* criptográfica usada no certificado;
- Nome do titular – Nome da entidade para o qual o certificado foi emitido;
- Nome do emitente – Autoridade Certificadora que emitiu/assinou o certificado;
- Período de validade – Mostra o período de validade do certificado no formato "Não antes" e "Não depois" (Ex. "Não antes de 05/03/2006 - 14:35:02" "Não depois de 05/03/2007 - 14:03:20");
- Informações de chave pública da entidade;
- Algoritmo de chave pública;
- Chave pública;
- Assinatura da AC – A garantia que a AC provê sobre a veracidade das informações contidas neste certificado de acordo com as políticas da AC;
- Identificador da chave do titular – É uma extensão do X.509 que possui um identificador numérico para a chave pública contida neste certificado, especialmente útil para que programas de computador possam se referir a ela;
- Identificador da chave do emitente – A mesma ideia mencionada anteriormente, só que se referindo a chave pública da AC que emitiu o certificado;
- Atributos ou extensões – A vasta maioria dos certificados X.509 possui campos chamados extensões – OID, que proveem algumas informações extras, como cadastros adicionais do titular e do emitente, especificações de propósito do certificado, e assim por diante.

Todos os conceitos explicados neste capítulo servem como base para que seja possível entender como o principal mecanismo de segurança apresentado neste trabalho funciona. Chamado de *Secure Sockets Layer* – SSL, será explicado no tópico a seguir.

2.6 Secure Sockets Layer – SSL/TLS

O SSL, algoritmo criptográfico que foi substituído pelo *Transport Layer Security* – TLS, é um padrão criptográfico bastante difundido pelos sistemas de segurança de redes. Entre seus objetivos estão a prevenção de interceptação da comunicação e modificação de informações por partes não autorizadas.

O SSL disponibiliza uma camada que se encontra acima da camada de transporte e abaixo da camada de aplicação da arquitetura TCP/IP. Essa camada extra é responsável por receber as informações da camada de aplicação, encapsulá-las de forma segura e transmitir o resultado para a camada de transporte. Isso evidencia que, como vantagem, o SSL pode receber informações de qualquer protocolo de aplicação, tais como o HTTP ou FTP, e torná-las seguras por meio do uso de criptografia, certificados digitais e assinaturas digitais (SECURE, s/d).

Seu funcionamento está baseado na arquitetura cliente-servidor. Além disso, em geral, a indicação de que um serviço usa o SSL é indicada por meio de uma porta TCP, que costuma ser a de número 443.

A seguir, enumeram-se os passos descritos para o estabelecimento da conexão, chamada de *handshake* (SECURE, s/d):

1) O cliente envia ao servidor informações iniciais como o a versão do SSL, informações para cifragem, número randômico, entre outros. A autenticação do cliente pelo servidor é opcional;
2) O servidor envia, em resposta, informações iniciais de suporte aos parâmetros do cliente, bem como um número randômico. Além disso, é

enviado seu certificado, para comprovação de identidade. Caso o cliente necessite ser autenticado, o servidor requisitará o certificado do cliente neste passo;

3) O cliente utiliza as informações obtidas pelo servidor para efetuar a autenticação. Informações como a validade do certificado e a cadeia de certificação são verificadas. Caso algo esteja em desacordo, o cliente será avisado e a tentativa de conexão poderá ser cancelada;

4) Caso a autenticação do passo anterior seja bem sucedida, o cliente fará a geração da *pre-master secret*. Essa chave é cifrada por meio da chave pública obtida com o certificado digital do servidor e, em seguida, é enviada para o servidor, usando algum algoritmo criptográfico assimétrico como o RSA ou *Diffie-Hellman*. Caso haja a necessidade de se autenticar o cliente, este também será responsável por escolher alguma informação e assiná-la com sua chave privada. Essa assinatura será enviada junto com o certificado digital do cliente e a *pre-master secret* cifrada.

5) O servidor, então, tentará autenticar o cliente por meio do certificado e assinatura enviados, caso seja necessário. Ele fará uso de mecanismos como a verificação de validade do certificado e sua cadeia de certificação. Se a autenticação falhar, o servidor encerrará o processo de *handshake*. Feita a autenticação com sucesso, o servidor utilizará sua chave privada para decifrar a mensagem que contém a *pre-master secret*. Essa chave servirá como base para criação da *master secret,* que será gerada por ambas as partes, segundo um algoritmo preestabelecido.

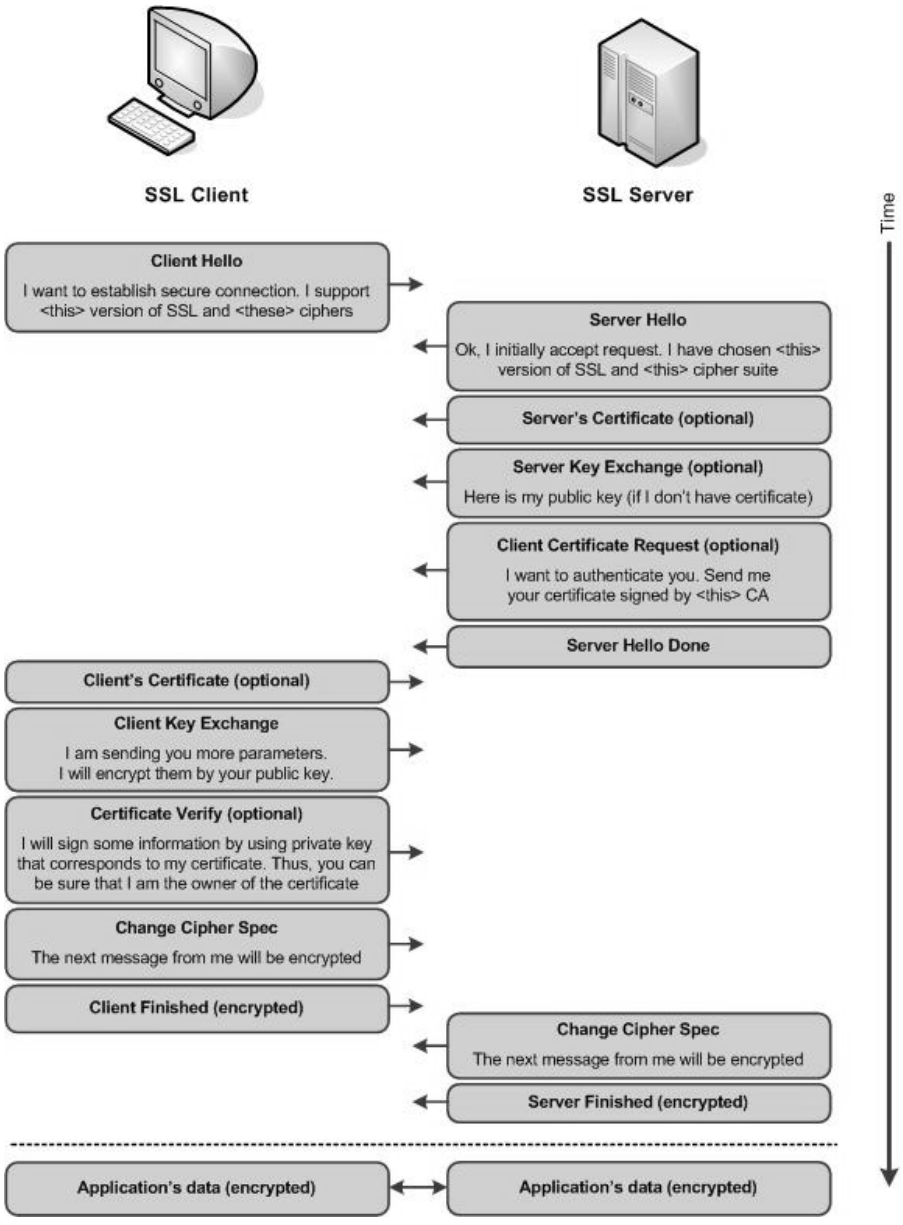
6) A *master secret* será utilizada para a criação da chave de sessão, também criada por ambas as partes. Essa chave simétrica é a que será utilizada para a transmissão de informações entre cliente e servidor.

7) Uma mensagem do cliente é enviada ao servidor para determinar que a comunicação será feita mediante o uso da chave de sessão gerada. Com essa chave, ele envia uma mensagem já criptografada dizendo

que sua parte de negociação de conexão está encerrada. Da mesma forma, o servidor envia uma mensagem criptografada para atestar que sua parte também encerrou.

Na figura 17 é possível ver um esquema simplificado com o funcionamento do SSL:

Figura 17 – Esquema de funcionamento do SSL.



Fonte: GUIMARÃES (s/d).

A conexão estabelecida pode ser renegociada a qualquer momento, seja em parâmetros criptográficos, seja em parâmetros de desempenho.

O SSL será utilizado em conjunto com o protocolo HTTP, que caracterizará o protocolo a seguir.

2.7 Hypertext Transfer Protocol Secure – HTTPS

O HTTPS consiste no uso do protocolo HTTP, comumente utilizado para a transmissão de dados na internet, com o SSL como transporte. O HTTP não oferece comunicação segura por padrão. O resultado da combinação é uma conexão que protege contra diversos tipos de ataques, tais como o *man-in-the-middle*, famoso ataque que permite a um terceiro manipular a informação segura trafegada entre as partes legítimas. Em geral, o protocolo utiliza a porta 443 para comunicação.

No contexto deste trabalho, o HTTPS será o protocolo utilizado pelo tocador de mídia para a conexão com o servidor de mídia. Assim que a solicitação de conexão chegar ao *proxy*, este deverá tratá-la para assegurar que há a devida proteção da comunicação.

Com os assuntos teóricos explicados, pode-se, no capítulo a seguir, partir para a proposta de solução de segurança desenvolvida, que integra o que foi visto ao longo deste capítulo e do anterior e um conjunto de soluções *open source* disponíveis no mercado.

3 PROPOSTA DE SEGURANÇA

Neste capítulo será mostrado como pode ser feita uma solução de segurança da informação para um fluxo de *streams*. A estrutura está embasada no *proxy* reverso, representado pela ferramenta *Nginx*, que receberá a requisição SSL. Ao recebê-la, deverá exigir o certificado do cliente, característica opcional do SSL, e autenticá-lo junto ao certificado da AC para a verificação de informações como a validade do certificado. Da mesma forma, a operação padrão do SSL de autenticação do servidor junto ao cliente será realizada.

Caso obtenha sucesso no passo anterior, a segurança será reforçada pelo uso da autenticação simples. A solicitação, após o *handshake* SSL, será encaminhada a uma página PHP, que será responsável por realizar a autenticação, mediante o uso de credenciais como usuário e senha. Esses serão conferidos por meio de um cadastro armazenado no banco de dados *MySQL*.

Após o sucesso da etapa anterior, uma nova página disponibilizará o acesso a um *link* que direcionará o usuário para o servidor de mídia, representado pelo *Icecast*, responsável pela geração e transmissão do fluxo de *streaming* para o cliente.

A seguir, apresentaremos as ferramentas e a configuração delas, bem como a sua integração.

3.1 Icecast

O servidor de mídia *Icecast* é um projeto *open source* desenvolvido pela empresa *Xiph.org*. Foi lançado no início de 1999 e permite a transmissão em fluxo de vários formatos, tais como o ogg *Vorbis*, mp3, além da transmissão de vídeos usando o codec *Theora* (ICECAST, s/d).

Seu modelo se utiliza de fontes externas para a transmissão dos fluxos. Essas fontes são responsáveis pela geração dos sinais de áudio ou vídeo, como, por exemplo, um aplicativo que captura os sinais de uma câmera ou microfone, ou

um concentrador de arquivos multimídia. O projeto disponibiliza o aplicativo *Ices* como fonte externa. Ele consiste num concentrador que trabalha com fluxos ao vivo ou *playlists* – listas que contém os arquivos a serem reproduzidos no fluxo.

O *Icecast*, neste trabalho, é o servidor responsável por obter os arquivos de mídia e transformá-los em fluxo. Há a possibilidade de se configurar algumas opções por meio da sua interface gráfica. Entretanto, sua interface é, em princípio, utilizada para acompanhar as configurações feitas.

3.2 Nginx

O *Nginx* é uma ferramenta criada por Igor *Sysoev*. Ele é um servidor *proxy* e um de seus principais usos é como *proxy* reverso para Servidores *Web*. Além disso, pode funcionar como *proxy* de correio, balanceador de carga e empacotador – *wrapper* – de conexões HTTPS. Seu uso é regulado por uma licença baseada na BSD (NGINX, s/d).

Essa ferramenta foi inicialmente concebida para servir o tráfego de grandes domínios russos, tais como o *Yandex*, *Mail.ru* e *Rambler*. Hoje, trabalha com o tráfego de sites como o *Netflix* e *Wordpress*. Segundo o *Netcraft*, o *Nginx* serviu cerca de 12,81% dos sítios mais requisitados da internet em fevereiro de 2013 (NGINX, s/d).

Para os objetivos deste trabalho, o *Nginx* realizará toda a negociação – *handshake* – SSL com o cliente, o que evitará que a requisição do *streaming* chegue diretamente ao servidor de mídia, adicionando uma camada de segurança. Ao mesmo tempo, assegurará que a conexão esteja devidamente autenticada e verificada antes de chegar a seu destino final por meio do uso de certificados digitais e autenticação simples. Ademais, o *Nginx* será o servidor responsável em receber requisições HTTPS de tocadores de mídia, verificar se estão de acordo com os padrões de formação do protocolo e encaminhá-las para seu destino. Além disso, o servidor é o responsável por hospedar as páginas PHP que farão a autenticação simples. A seguir, estão algumas características da ferramenta (NGINX, s/d):

- Serviço de páginas estáticas e de índice – *index*, auto indexação e *cache* de descrição *open file*;
- Serviço de *proxy* acelerado com *cache*; balanceamento de carga e tolerância a falhas;
- Suporte à aceleração com *caching* de servidores *FastCGI*, uwsgi, SCGI e *memcached*;
- Arquitetura modular. Os filtros incluem o uso do *gzip*, respostas truncadas, XSLT, SSI e transformação de imagens;
- Suporte SNI para SSL e TLS.

3.3 OpenCA

O *OpenCA* é uma solução criada pela *OpenCA LABS* que oferece uma aplicação para implementação de uma AC. Seu uso é *open source*. Como base, utiliza-se de várias aplicações comuns no mercado, tais como o PERL, *OpenLDAP*, *OpenSSL*, *Apache*, MySQL, entre outros.

A ferramenta é bem completa, pois, além da criação da AC, permite a criação e gerenciamento de uma AR, entre outros.

Neste trabalho, o *OpenCA* é responsável por gerar os certificados de cliente para que sejam autenticados junto ao servidor *Nginx* ao utilizar o SSL. Além disso, deve disponibilizar o certificado da AC para que o *Nginx* possa verificar o certificado do cliente.

Na figura 18 é possível ver uma tela com o funcionamento do *OpenCA*:

FIGURA 18 – Tela do *OpenCA*.

Fonte: o autor.

3.4 Configuração

As configurações foram obtidas a partir dos arquivos padrão ou de exemplos obtidos na internet. Como as soluções utilizadas possuem considerável documentação, foi possível obter várias referências que colaboraram para a efetiva construção da solução.

A configuração do *Nginx* consistiu na edição do arquivo *nginx.conf* e *default,* ambos encontrados na instalação padrão da ferramenta no *Ubuntu*. O primeiro é responsável pelas configurações globais do servidor. O último, por sua vez, permite realizar a configuração do *host* virtual que responderá pelos protocolos HTTP e HTTPS. Nessas configurações, define-se informações como o número da porta para conexão, qual a versão e cifra criptográfica a ser usada pelo SSL, o diretório padrão das páginas, a autenticação do cliente, localização dos certificados do servidor e da AC, e assim por diante.

O *Icecast* possui um arquivo de configuração, chamado *icecast.xml*. Nele, define-se informações como o número máximo de conexões ao servidor de mídia, credenciais de acesso à interface gráfica de configuração e acompanhamento, credenciais de acesso para que a fonte de mídia se conecte ao servidor, a porta de conexão, entre outros.

A fonte de mídia, representada pelo *Ices*, também possui arquivo de configuração e foi chamado de *ices.conf*. Nele, é possível configurar os dados a respeito do *stream* gerado. Neste trabalho, foi utilizado um fluxo gerado a partir de uma *playlist* – lista estática de arquivos de mídia –, chamada *playlist.txt*, que será reproduzida dinamicamente, sem precisar manter a ordem dos arquivos de mídia colocada no arquivo.

A geração dos certificados digitais do cliente utilizados foi feita com o uso do *OpenCA*. Esta ferramenta habilita todo o processo de gerenciamento de certificados realizado por uma AC. Além da instalação do pacote principal, deve-se instalar o pacote *OpenCA Tools*. O arquivo de configuração é o *config.xml*. Ele define como se deve comportar o servidor, credenciais de acesso, porta de conexão, emails para contato, entre outros. Para efeito de testes práticos dos conceitos aprendidos ao longo do curso, o certificado do servidor SSL foi auto-assinado.

A solução de autenticação simples foi realizada com o uso da linguagem de script PHP em conjunto com o banco de dados *MySQL*. O uso do PHP no servidor *Nginx* requer a instalação de um módulo chamado *FastCGI*. A senha é armazenada no banco de dados na forma de *hash* no padrão SHA-1. A autenticação consiste em solicitar ao usuário, após a validação de seu certificado, usuário e senha para adição de mais uma camada de segurança à solução. Caso haja extravio ou roubo do certificado, o invasor deve ter, ainda, essas credenciais para completar o acesso ao *stream*.

A seguir, veremos como ocorreram os testes com a solução instalada, configurada e em operação.

3.5 Testes

O ambiente onde foram feitos os testes é composto por cinco máquinas lógicas. A primeira, representada pelo *Nginx*, será o *proxy* reverso e efetuará a conexão SSL, com autenticação do servidor e do cliente.

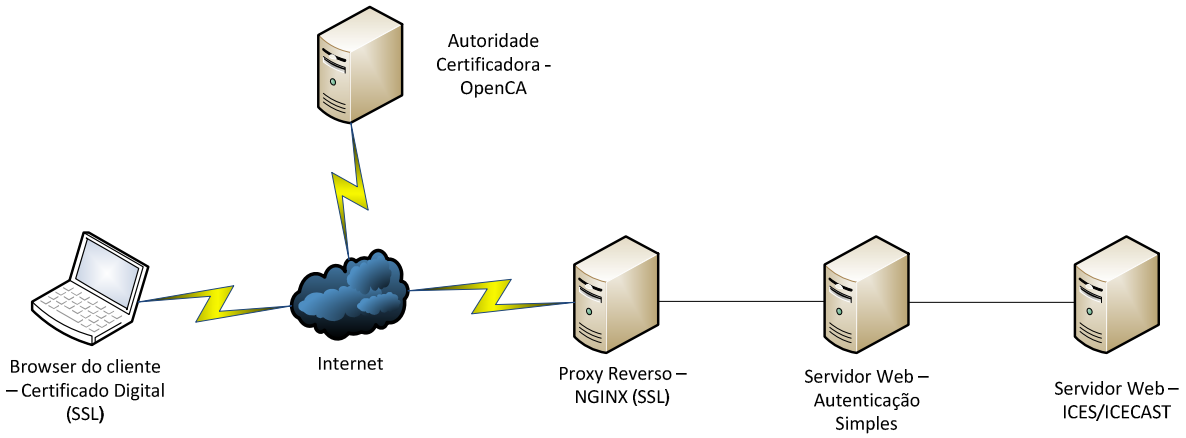
A segunda representa o servidor *Web* que efetuará a autenticação simples. Esse serviço é também provido pelo *Nginx*, que hospeda as páginas em PHP e o banco de dados *MySQL*.

A terceira máquina é o servidor *Icecast*, que carregará a lista de reprodução de arquivos mp3 e os transformará em fluxo para entrega ao cliente. As três máquinas estão baseadas na plataforma *Ubuntu Linux* 11.04, instalado no ambiente virtualizado *Oracle VirtualBox*.

A quarta hospeda o *OpenCA,* autoridade certificadora que será utilizada para gerar os certificados e roda no sistema operacional *CentOS* 5.9, também instalado no *VirtualBox*. A quinta, por sua vez, é a máquina do cliente que deseja acessar o *stream* por meio do *browser*.

A figura 19 apresenta um esquema com a solução empregada:

Figura 19 – Esquema de funcionamento da solução de segurança.



Fonte: o autor.

3.5.1 *Autenticação SSL do cliente*

O objetivo deste teste é verificar o funcionamento do processo de autenticação SSL para que o servidor ateste o certificado do cliente, permitindo o

seu acesso. Conforme explicado anteriormente, esse passo não é padrão para um *handshake* SSL.

Para sua execução, foram gerados dois certificados de cliente fictícios. Os certificados, de nome “Fabricio Cardoso” e “Pedro Almeida”, foram emitidos e assinados pela AC criada com o *OpenCA,* chamada “CA do Ximbica”. O certificado da “CA do Ximbica” foi configurado no *Nginx* para que ele possa efetivamente reconhecer os certificados que serão apresentados pelo cliente.

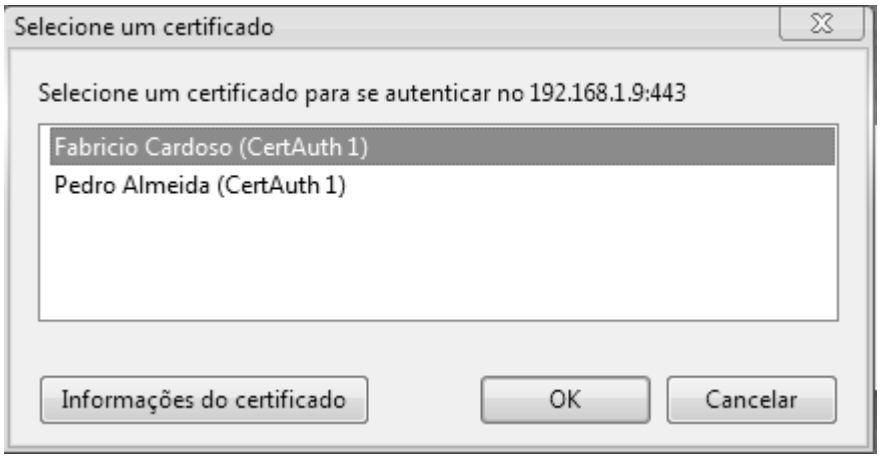
Os certificados de cliente foram devidamente instalados no formato PKCS #12. A ferramenta *OpenCA* possibilita a exportação nesse formato, o que facilitou o procedimento. Ao importá-los na máquina do cliente, foi solicitada a senha para acesso à chave privada. A instalação do certificado pode ser vista na figura 20:

FIGURA 20 – Solicitação de Certificado.

Fonte: o autor.

Acessou-se a página do servidor por meio do *browser* utilizando o protocolo HTTPS, o endereço IP 192.168.1.9, endereço local, e, opcionalmente, a porta 443. Neste teste, foram utilizados dois navegadores: o *Google Chrome* e o *Mozilla Firefox*. O *Internet Explorer* apresentou problemas na autenticação do cliente, pois o servidor não conseguiu efetuar a autenticação. Com o acesso inicial realizado, foi exibida a tela de seleção de certificados, que pode ser vista na figura 21:

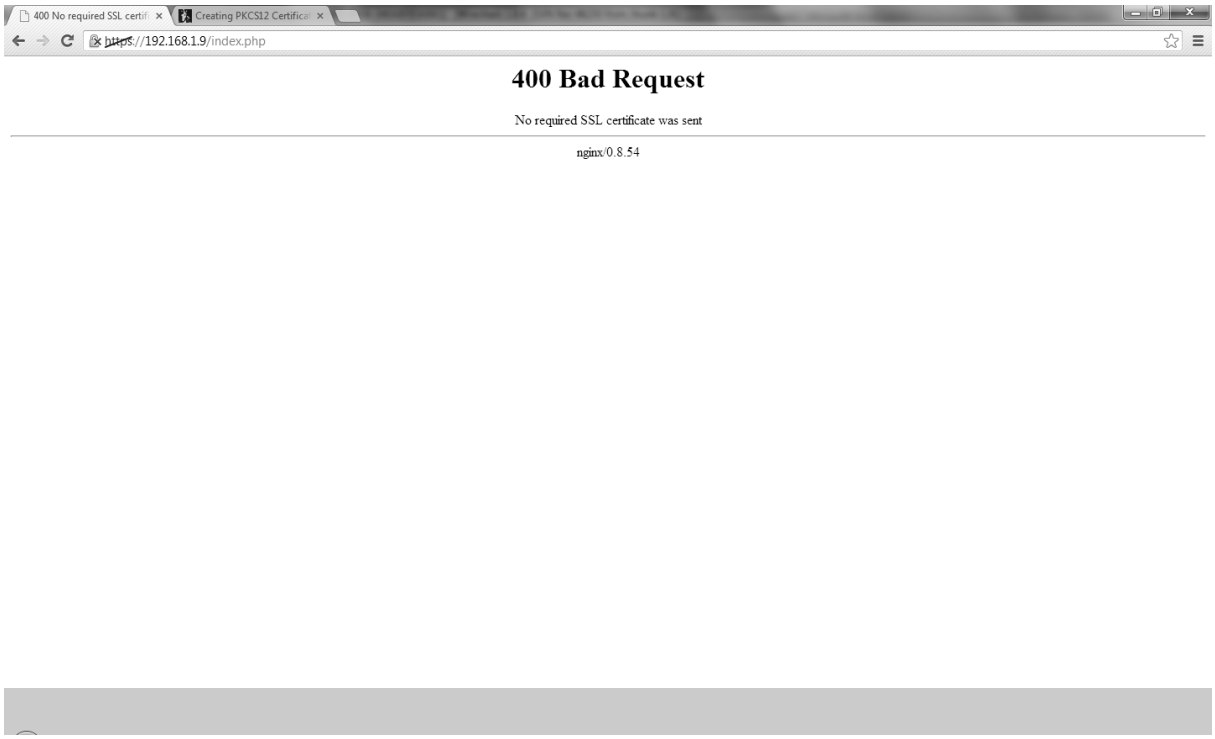
FIGURA 21 – Seleção de Certificado.



Fonte: o autor.

Um teste inicial consistiu em não apresentar nenhum certificado, clicando no botão *Cancelar* da figura 21. A tela exibida após a ação foi a da figura 22, que bloqueou o acesso caso não fosse apresentado algum certificado:

FIGURA 22 – Tela de bloqueio de acesso do *Nginx*.



Fonte: o autor.

Na segunda parte do teste, selecionamos um certificado. Com isso, o servidor iniciou os procedimentos do *handshake* SSL para autenticação do cliente. Utilizou-se o *Wireshark* para capturar os pacotes de rede e acompanhar o *handshake,* conforme a figura 23:

FIGURA 23 – Extrato do *Wireshark* mostrando o *handshake* SSL com autenticação de cliente e servidor.

No.	Source	Destination	Length	Info
22	192.168.1.2	192.168.1.9	220	Client Hello
23	192.168.1.2	192.168.1.9	220	Client Hello
26	192.168.1.9	192.168.1.2	1514	Server Hello
27	192.168.1.9	192.168.1.2	1514	Certificate
29	192.168.1.9	192.168.1.2	398	Server Key Exchange, Certificate Request, Server Hello Done
32	192.168.1.9	192.168.1.2	1514	Server Hello
33	192.168.1.9	192.168.1.2	1514	Certificate
35	192.168.1.9	192.168.1.2	398	Server Key Exchange, Certificate Request, Server Hello Done
45	192.168.1.2	192.168.1.9	220	Client Hello
47	192.168.1.9	192.168.1.2	1514	Server Hello
48	192.168.1.9	192.168.1.2	1514	Certificate
50	192.168.1.9	192.168.1.2	398	Server Key Exchange, Certificate Request, Server Hello Done
52	192.168.1.2	192.168.1.9	1241	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
56	192.168.1.9	192.168.1.2	1004	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
74	192.168.1.2	192.168.1.9	1128	Client Hello
77	192.168.1.9	192.168.1.2	199	Server Hello, Change Cipher Spec, Encrypted Handshake Message
78	192.168.1.2	192.168.1.9	113	Change Cipher Spec, Encrypted Handshake Message
79	192.168.1.2	192.168.1.9	512	Application Data, Application Data
81	192.168.1.9	192.168.1.2	571	Application Data
82	192.168.1.2	192.168.1.9	464	Application Data, Application Data
83	192.168.1.9	192.168.1.2	475	Application Data

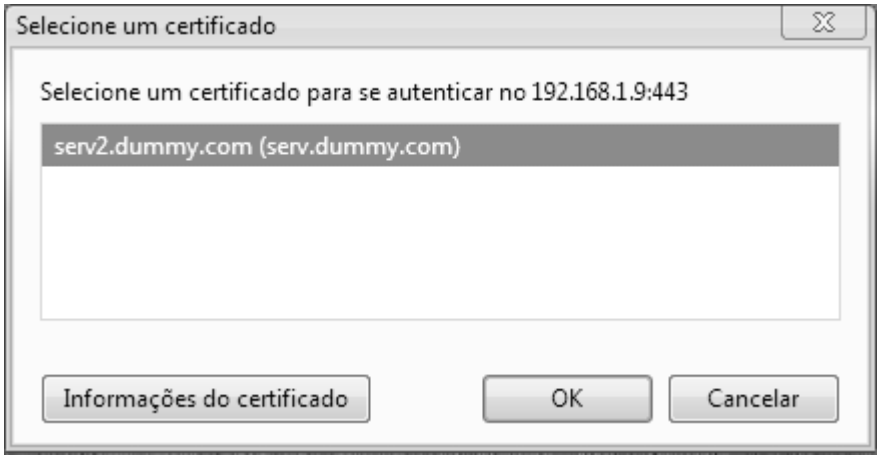
Fonte: o autor.

O primeiro sinal de *handshake* começou com o *Client Hello*. O servidor apresentou o seu certificado para que o cliente o autenticasse. Em seguida, o servidor exigiu a apresentação de um certificado do cliente. Após a devida autenticação do servidor pelo cliente, o cliente enviou para o servidor o certificado selecionado. Com isso, uma nova sessão foi criada pelo servidor, mediante a bem sucedida análise de certificado do cliente junto às informações da AC existentes no certificado de AC no *Nginx*. Ao final da comunicação, houve início a transmissão segura pelo canal.

Tentou-se, em seguida, efetuar testes de autenticação do cliente fazendo uso de um certificado emitido por uma AC que o *Nginx* não reconhece. O resultado foi o de que o servidor simplesmente ignorou certificados que não tinham ACs reconhecidas. Isso pode ser visto na figura 24, em que foi retirado o certificado da AC “CA do Ximbica” das configurações do servidor e houve a substituição por outro,

da AC fictícia “serv.dummy.com”, e importou-se um certificado emitido por ela, com nome “serv2.dummy.com”:

FIGURA 24 – CA substituída e certificado correspondente.



Fonte: o autor.

Os certificados emitidos pela “CA do Ximbica” simplesmente sumiram da lista, o que caracteriza uma boa construção do *Nginx* quanto à verificação de certificados.

Com essa etapa concluída, partiu-se para a segunda etapa de testes: os de autenticação SSL do servidor pelo cliente.

3.5.2 *Autenticação SSL do servidor*

Este teste foi elaborado para verificar como se dá a autenticação do servidor pelo cliente. Essa operação é padrão do protocolo SSL. Utilizou-se, de forma proposital, um certificado emitido e assinado pelo próprio servidor *Web* que hospeda o *Nginx*. Isso foi possível com o auxílio da ferramenta *OpenSSL*, que gerou e assinou o certificado. O *browser*, tendo o *Google Chrome* como exemplo, ao se deparar com um certificado auto-assinado, emitiu uma tela de aviso alertando para o fato de o certificado do servidor *Nginx* não ter sido devidamente verificado. A tela pode ser vista na figura 25:

FIGURA 25 – Tela de aviso de falha de verificação de certificado.

Fonte: o autor.

Clicou-se na opção “Continuar mesmo assim” para forçar a aceitação do certificado do servidor pelo *browser*. Com isso, a autenticação foi feita com sucesso.

Essa ação não é recomendada e foi tomada apenas para observar a aplicação prática de certificados auto-assinados, pois o certificado pode ter sido emitido por entidades não confiáveis, o que compromete toda a troca de informações pelo canal seguro.

Será apresentada, adiante, a última fase de testes deste trabalho, que trata da autenticação simples.

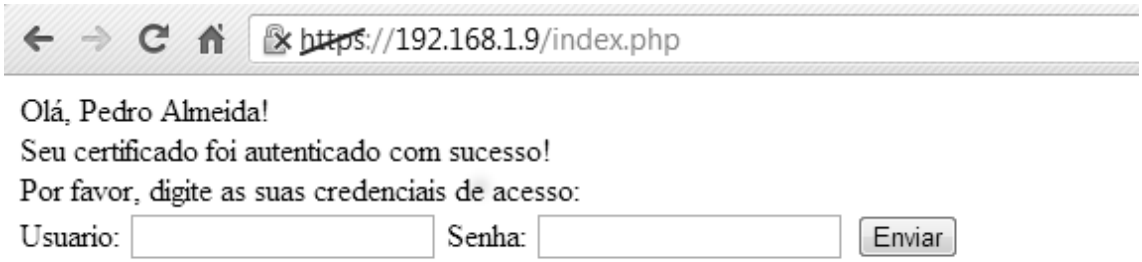
3.5.3 *Autenticação Simples*

Este último teste consiste em verificar a funcionalidade da autenticação simples proposta, mediante a inserção de credenciais de acesso que serão verificadas por meio de uma consulta ao banco de dados *MySQL*, que guarda os

hashes das senhas. Se o usuário for bem sucedido, a aplicação deverá permitir o acesso ao *link* do *stream*. Caso contrário, será necessário utilizar novas credenciais.

Logo após o passo descrito no item 3.5.1, há um direcionamento para uma página de *script* PHP, onde são solicitados usuário e senha. Na figura 26 está a página de autenticação:

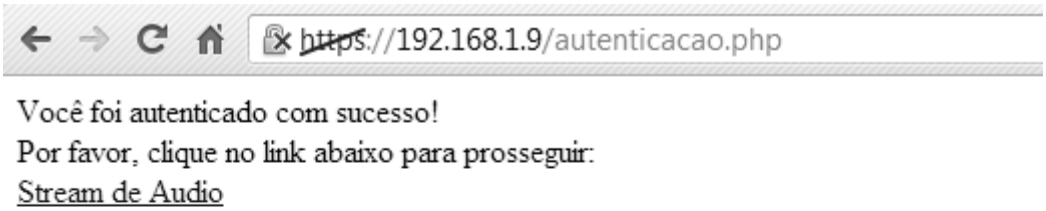
FIGURA 26 – Página de autenticação simples.



Fonte: o autor.

Feita com sucesso, a autenticação liberou o acesso à página com o *link* para o fluxo, conforme a figura 27:

FIGURA 27 – *Link* de acesso ao *stream*.



Fonte: o autor.

Foi testado um perfil que não existe e, como consequência, não houve sucesso. A tela da figura 28 foi, então, exibida:

FIGURA 28 – Tela de falha na autenticação simples.

Fonte: o autor.

A seguir, serão mostradas quais as conclusões que puderam ser obtidas ao longo deste trabalho.

CONCLUSÃO

No ramo dos *streams*, há diversas formas de se aplicar a segurança da informação. Neste trabalho, a ideia principal reside no fato de que um canal de comunicação pode se tornar seguro de forma transparente para o usuário. Para ser bem sucedido no acesso ao *stream*, basta que obtenha certificado válido junto ao serviço e credenciais adequadas.

No cenário atual, pode-se ampliar o estudo para várias vertentes. Uma evolução, por exemplo, seria utilizar a solução para filtrar os tipos de acesso com o uso das credenciais de usuário e senha. Ou seja, poder-se-ia permitir o acesso de determinada pessoa a determinado *stream* de maior qualidade com a autenticação simples, de acordo com o perfil de um assinante, por exemplo. Isso se torna possível por meio da alteração do *script* PHP desenvolvido. Ademais, esse *script* pode sofrer alterações como o uso de *salt* quando a senha for cadastrada. Isso pode conferir maior aleatoriedade a ela, o que eleva sua segurança.

Da mesma forma, o uso de certificados digitais pessoais pode realizar a filtragem descrita no parágrafo anterior, sempre de acordo com níveis de acesso predeterminados.

Além das sugestões mencionadas, existem outras melhorias que podem ser feitas. No servidor *Nginx*, sugere-se que sejam utilizados certificados de servidor assinados por uma AC confiável e que seja reconhecida pelo ICP Brasil, autoridade de caráter nacional em certificados digitais.

Ao longo do desenvolvimento da solução proposta, foi observado que, enquanto meramente acadêmica, pode ser utilizada de forma comercial, mediante ajuste fino nas configurações e maior personalização.

Como sugestão de solução alternativa, há a opção de se utilizar algoritmos criptográficos diretamente no *stream,* o que dispensa a necessidade de se trabalhar com o SSL. O trabalho pode ser feito bloco a bloco ou em fluxo, com criptografia simétrica ou assimétrica, ou ambas. Entretanto, tal solução possivelmente demandará conhecimentos aprofundados de programação.

Por fim, verificou-se que a solução possui caráter genérico e pode ser utilizada para diversas aplicações, tais como a autenticação de clientes em *web services*, páginas de acesso restrito, entre outras.

REFERÊNCIAS

APACHE. **Forward and Reverse Proxies**. Disponível em <http://httpd.apache.org/docs/2.0/mod/mod_proxy.html#forwardreverse>. Acesso em: 13 jan. 2013.

ARTHUR, L. **VPN - Virtual Private NETWORK.** Disponível em: < http://www.slideshare.net/luiz_arthur/seguranca-da-informao-vpn/> Acesso em: 14 dez. 2012.

AUSTERBERRY, D. **The Technology of Video and Audio Streaming**. Burlington: El Sevier, 2005, p. 133-243.

BELLARE, M.; CANETTI, R.; KRAWCZYK, H. **Keying Hash Functions for Message Authentication**. p. 3, 1996.

CARVALHO, H. E. T. **PKI – Infraestrutura de Chaves Públicas**. Disponível em: < http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2008_2/hugo/index.html>. Acesso em 29 nov. 2012, il.

CERTIFICADO Digital. Wikipédia: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Certificado_digital>. Acesso em: 15 jan. 2013.

CRISTIAN, 2007. **Assinatura Digital**. Disponível em: <http://blog.cristiantm.com.br/2007/07/25/criptografia-para-leigos-parte-3/>. Acesso em: 10 de out. 2012.

DIFFIE, B. W; HELLMAN, M. E. **News directions in cryptography. IEEE Transactions on Information Theory**. v. 22, p. 644 – 654, 1976.

DUARTE, O. C. M. B. **RTP**. Disponível em: <http://www.gta.ufrj.br/grad/06_1/rtp/>. Acesso em: 16 nov. 2012.

GIROD, B. **Video Over Networks**. Disponível em: <http://www.stanford.edu/class/ee398b/handouts/lectures/08-VideoOverNetworks.pdf>. Acesso em: 15 nov. 2012.

GPR, 2000. **Virtual Private Network**. Disponível em: <http://www.gpr.com.br/download/vpn.pdf/>. Acesso em: 13 dez. 2012.

GUIMARÃES, M. S. **SSL no Apache (https)**. Disponível em: < http://www.csirt.pop-mg.rnp.br/docs/https/https.htm>. Acesso em: 30 nov. 2012, il.

HOUSLEY, R. **Internet X.509 Public Key Infrastructure**. Disponível em: <http://www.ietf.org/rfc/rfc2459.txt>. Acesso em: 20 jan. 2013.

ICECAST. **Icecast Streaming Server**. Disponível em: <http://www.icecast.org/>. Acesso em: 02 fev. 2012.

ICECAST. **Stream Structure per Mount Point**. Disponível em:

< http://www.icecast.org/docs.php>. Acesso em: 21 jan. 2013, il.

ITI. **Certificado Digital**. Disponível em: <http://www.iti.gov.br/certificacao-digital/certificado-digital>. Acesso em: 24 jan. 2013.

JAY, M. **The SHOUTcast Streaming Standard [Technical]**. Disponível em: <http://forums.radiotoolbox.com/viewtopic.php?t=74>. Acesso em: 03 fev. 2013.

JOHN, V. **Managed File Transfer and Network Solutions**. Disponível em: < http://www.jscape.com/blog/bid/84422/Secure-File-Transfers-Symmetric-vs-Asymmetric-Key-Encryption>. Acesso em: 01 dez. 2012, il.

KUROSE, J F.; ROSS, K W. **Redes de Computadores e a Internet - Uma Abordagem Top-Down**. 3 ed. São Paulo: Pearson, 2010. 656 p.

MBONE. Wikipédia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Mbone>. Acesso em: 10 nov. 2012.

MENDONÇA, G. G. **Protocolos de Segurança IP – IPSec**. Disponível em: < http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2009_2/gabriel/implementacao.html#bits>. Acesso em: 02 dez. 2012, il.

NAKAMURA, E. T.; GEUS, P. L. **Segurança de Redes em Ambientes Corporativos**. São Paulo: Novatec, 2007, p. 237-241.

NGINX. **Nginx Professional Services**. Disponível em: < http://nginx.org/en/docs/>. Acesso em: 03 jan. 2013.

NUNES, D. S. **PKI – Public Key Infrastructure**. Disponível em: <http://www.gta.ufrj.br/grad/07_2/delio/Resumo.html>. Acesso em: 26 nov. 2012, il.

OZER, J. **What is Streaming?** Disponível em: <http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-Streaming-74052.aspx>. Acesso em: 19 nov. 2012.

PROXY Server. Wikipédia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Proxy_server>. Acesso em: 04 jan. 2013.

QUICKTIME. Wikipédia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/QuickTime>. Acesso em: 10 nov. 2012.

REAL Time Streaming Protocol. Wikipédia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/RTSP>. Acesso em: 26 nov. 2012.

REAL-TIME Transport Protocol. Wikipédia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Real-time_Transport_Protocol>. Acesso em: 25 nov. 2012.

RECKZIEGEL, M. **Criptografia em PHP e Web Services**. Disponível em: < http://imasters.com.br/artigo/4802/web-services/criptografia-em-php-e-webservices/>. Acesso em: 18 nov. 2012, il.

SCHNEIER, B. **Applied Cryptography**. 2 ed. John Wiley & Sons, 1996. 784p.

SCHULZRINNE, H. **RTP: A Transport Protocol for Real-Time Applications**. Disponível em: <http://www.ietf.org/rfc/rfc3550.txt>. Acesso em: 25 nov. 2012.

SECURE Sockets Layer. Wikipédia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Secure_Sockets_Layer>. Acesso em: 15 jan. 2013.

SOARES, G.; LEMOS, G.; COLCHET, S. **Redes de Computadores - LANs, MANs e WANs às Redes ATM**. 2 ed. Rio de Janeiro: Campus, 1995. 705 p.

STREAMING Media. Wikipédia: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Streaming_media>. Acesso em: 19 nov. 2012.

TEOTÔNIO, I. D. **Configurando Proxy manual nos clientes e bloqueando Conexões do Internet Explorer**. Disponível em: < http://techlivre.blogspot.com.br/2010/02/configurando-proxy-manual-e-bloqueando.html>. Acesso em: 15 dez. 2012, il.

VAZ, N. **Proxy Reverso com Apache**. Disponível em: <http://linuxnatyworking.wordpress.com/tag/reverse-proxy/>. Acesso em: 20 nov. 2012, il.

APÊNDICE A – Arquivo /etc/nginx/nginx.conf

```
user www-data;
worker_processes 4;
pid /var/run/nginx.pid;

events {
	worker_connections 768;
	# multi_accept on;
}

http {

	##
	# Basic Settings
	##

	sendfile on;
	tcp_nopush on;
	tcp_nodelay on;
	keepalive_timeout 65;
	types_hash_max_size 2048;
	# server_tokens off;

	# server_names_hash_bucket_size 64;
	# server_name_in_redirect off;

	include /etc/nginx/mime.types;
	default_type application/octet-stream;

	##
	# Logging Settings
	##

	access_log /var/log/nginx/access.log;
	error_log /var/log/nginx/error.log;

	##
	# Gzip Settings
	##

	gzip on;
	gzip_disable "msie6";

	# gzip_vary on;
	# gzip_proxied any;
	# gzip_comp_level 6;
	# gzip_buffers 16 8k;
	# gzip_http_version 1.1;
	# gzip_types text/plain text/css application/json application/x-javascript
text/xml application/xml application/xml+rss text/javascript;

	##
	# Virtual Host Configs
	##

	include /etc/nginx/conf.d/*.conf;
	include /etc/nginx/sites-enabled/*;
}


#mail {
#	# See sample authentication script at:
#	# http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
#	# auth_http localhost/auth.php;
#	# pop3_capabilities "TOP" "USER";
#	# imap_capabilities "IMAP4rev1" "UIDPLUS";
```

```
#
#      server {
#             listen     localhost:110;
#             protocol   pop3;
#             proxy      on;
#      }
#
#      server {
#             listen     localhost:143;
#             protocol   imap;
#             proxy      on;
#      }
#}
```

APÊNDICE B – Arquivo /etc/nginx/sites-available/default

```
# You may add here your
# server {
#	...
# }
# statements for each of your virtual hosts to this file

##
# You should look at the following URL's in order to grasp a solid understanding
# of Nginx configuration files in order to fully unleash the power of Nginx.
# http://wiki.nginx.org/Pitfalls
# http://wiki.nginx.org/QuickStart
# http://wiki.nginx.org/Configuration
#
# Generally, you will want to move this file somewhere, and start with a clean
# file but keep this around for reference. Or just disable in sites-enabled.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##

server {
	#listen   80; ## listen for ipv4; this line is default and implied
	#listen   [::]:80 default ipv6only=on; ## listen for ipv6

	root /var/www/html;
	index index.html index.htm;

	# Make site accessible from http://localhost/
	server_name 192.168.1.9;

	location / {
		# First attempt to serve request as file, then
		# as directory, then fall back to index.html
		try_files $uri $uri/ /index.html;
	}

	location /doc {
		root /usr/share;
		autoindex on;
		allow 127.0.0.1;
		deny all;
	}

	location /images {
		root /usr/share;
		autoindex off;
	}

	#error_page 404 /404.html;

	# redirect server error pages to the static page /50x.html
	#
	#error_page 500 502 503 504 /50x.html;
	#location = /50x.html {
	#	root /usr/share/nginx/www;
	#}

	# proxy the PHP scripts to Apache listening on 127.0.0.1:80
	#
	#location ~ \.php$ {
	#	proxy_pass http://127.0.0.1;
	#}

	# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
	#
```

```
        location ~ \.php$ {
                fastcgi_pass 127.0.0.1:9000;
                fastcgi_index index.php;
                include fastcgi_params;
                  fastcgi_param   SCRIPT_FILENAME $document_root$fastcgi_script_name;
        }

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #       deny all;
        #}
}


# another virtual host using mix of IP-, name-, and port-based configuration
#
#server {
#       listen 8000;
#       listen somename:8080;
#       server_name somename alias another.alias;
#       root html;
#       index index.html index.htm;
#
#       location / {
#               try_files $uri $uri/ /index.html;
#       }
#}


# HTTPS server
#
server {
        listen 443;
        server_name 192.168.1.9;

        root /var/www/html;
        index index.html index.htm;

        ssl on;
        ssl_certificate server.crt;
        ssl_certificate_key server.key;
         ssl_client_certificate openca.crt;
         ssl_verify_client on;
         ssl_verify_depth 1;

        ssl_session_timeout 5m;

        ssl_protocols SSLv3 TLSv1;
        ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv3:+EXP;
        ssl_prefer_server_ciphers on;

        location / {
                try_files $uri $uri/ /index.html;
        }

         location ~ \.php$ {
                 fastcgi_pass 127.0.0.1:9000;
                 fastcgi_index index.php;
                 include fastcgi_params;
                 fastcgi_param   SCRIPT_FILENAME $document_root$fastcgi_script_name;
                 fastcgi_param   VERIFIED $ssl_client_verify;
                 fastcgi_param   DN $ssl_client_s_dn;
                 fastcgi_param   CERT $ssl_client_cert;
         }
}
```

APÊNDICE C – Arquivo /usr/local/etc/icecast.xml

```
<icecast>
    <limits>
        <clients>100</clients>
        <sources>2</sources>
        <threadpool>5</threadpool>
        <queue-size>524288</queue-size>
        <client-timeout>30</client-timeout>
        <header-timeout>15</header-timeout>
        <source-timeout>10</source-timeout>
        <!-- If enabled, this will provide a burst of data when a client
             first connects, thereby significantly reducing the startup
             time for listeners that do substantial buffering. However,
             it also significantly increases latency between the source
             client and listening client.  For low-latency setups, you
             might want to disable this. -->
        <burst-on-connect>1</burst-on-connect>
        <!-- same as burst-on-connect, but this allows for being more
             specific on how much to burst. Most people won't need to
             change from the default 64k. Applies to all mountpoints  -->
        <burst-size>65535</burst-size>
    </limits>

    <authentication>
        <!-- Sources log in with username 'source' -->
        <source-password>hackme</source-password>
        <!-- Relays log in username 'relay' -->
        <relay-password>hackme</relay-password>

        <!-- Admin logs in with the username given below -->
        <admin-user>admin</admin-user>
        <admin-password>hackme</admin-password>
    </authentication>

    <!-- set the mountpoint for a shoutcast source to use, the default if not
         specified is /stream but you can change it here if an alternative is
         wanted or an extension is required
    <shoutcast-mount>/live.nsv</shoutcast-mount>
    -->

    <!-- Uncomment this if you want directory listings -->
    <!--
    <directory>
        <yp-url-timeout>15</yp-url-timeout>
        <yp-url>http://dir.xiph.org/cgi-bin/yp-cgi</yp-url>
    </directory>
     -->

    <!-- This is the hostname other people will use to connect to your server.
    It affects mainly the urls generated by Icecast for playlists and yp
    listings. -->
    <hostname>192.168.1.9</hostname>

    <!-- You may have multiple <listener> elements -->
    <listen-socket>
        <port>8000</port>
        <!-- <bind-address>127.0.0.1</bind-address> -->
        <!-- <shoutcast-mount>/stream</shoutcast-mount> -->
    </listen-socket>
    <!--
    <listen-socket>
        <port>8001</port>
    </listen-socket>
    -->
```

```
    <!--<master-server>127.0.0.1</master-server>-->
    <!--<master-server-port>8001</master-server-port>-->
    <!--<master-update-interval>120</master-update-interval>-->
    <!--<master-password>hackme</master-password>-->

    <!-- setting this makes all relays on-demand unless overridden, this is
         useful for master relays which do not have <relay> definitions here.
         The default is 0 -->
    <!--<relays-on-demand>1</relays-on-demand>-->

    <!--
    <relay>
        <server>127.0.0.1</server>
        <port>8001</port>
        <mount>/example.ogg</mount>
        <local-mount>/different.ogg</local-mount>
        <on-demand>0</on-demand>

        <relay-shoutcast-metadata>0</relay-shoutcast-metadata>
    </relay>
    -->

    <!-- Only define a <mount> section if you want to use advanced options,
         like alternative usernames or passwords
    <mount>
        <mount-name>/example-complex.ogg</mount-name>

        <username>othersource</username>
        <password>hackmemore</password>

        <max-listeners>1</max-listeners>
        <dump-file>/tmp/dump-example1.ogg</dump-file>
        <burst-size>65536</burst-size>
        <fallback-mount>/example2.ogg</fallback-mount>
        <fallback-override>1</fallback-override>
        <fallback-when-full>1</fallback-when-full>
        <intro>/example_intro.ogg</intro>
        <hidden>1</hidden>
        <no-yp>1</no-yp>
        <authentication type="htpasswd">
                <option name="filename" value="myauth"/>
                <option name="allow_duplicate_users" value="0"/>
        </authentication>
        <on-connect>/home/icecast/bin/stream-start</on-connect>
        <on-disconnect>/home/icecast/bin/stream-stop</on-disconnect>
    </mount>

    <mount>
        <mount-name>/auth_example.ogg</mount-name>
        <authentication type="url">
            <option name="mount_add"
value="http://myauthserver.net/notify_mount.php"/>
            <option name="mount_remove"
value="http://myauthserver.net/notify_mount.php"/>
            <option name="listener_add"
value="http://myauthserver.net/notify_listener.php"/>
            <option name="listener_remove"
value="http://myauthserver.net/notify_listener.php"/>
        </authentication>
    </mount>

    -->

    <fileserve>1</fileserve>

    <paths>
             <!-- basedir is only used if chroot is enabled -->
        <basedir>/usr/local/share/icecast</basedir>
```

```
        <!-- Note that if <chroot> is turned on below, these paths must both
             be relative to the new root, not the original root -->
        <logdir>/log</logdir>
        <webroot>/web</webroot>
        <adminroot>/admin</adminroot>
        <!-- <pidfile>/usr/local/share/icecast/icecast.pid</pidfile> -->

        <!-- Aliases: treat requests for 'source' path as being for 'dest' path
             May be made specific to a port or bound address using the "port"
             and "bind-address" attributes.
          -->
        <!--
        <alias source="/foo" dest="/bar"/>
          -->
        <!-- Aliases: can also be used for simple redirections as well,
             this example will redirect all requests for http://server:port/ to
             the status page
          -->
        <alias source="/" dest="/status.xsl"/>
    </paths>

    <logging>
        <accesslog>access.log</accesslog>
        <errorlog>error.log</errorlog>
        <!-- <playlistlog>playlist.log</playlistlog> -->
       <loglevel>3</loglevel> <!-- 4 Debug, 3 Info, 2 Warn, 1 Error -->
       <logsize>10000</logsize> <!-- Max size of a logfile -->
        <!-- If logarchive is enabled (1), then when logsize is reached
             the logfile will be moved to [error|access|playlist].log.DATESTAMP,
             otherwise it will be moved to [error|access|playlist].log.old.
             Default is non-archive mode (i.e. overwrite)
        -->
        <!-- <logarchive>1</logarchive> -->
    </logging>

    <security>
        <chroot>1</chroot>
        <changeowner>
            <user>fabricio</user>
            <group></group>
        </changeowner>
    </security>
</icecast>
```

APÊNDICE D – Arquivo /usr/local/etc/ices.conf

```
<?xml version="1.0"?>
<ices:Configuration xmlns:ices="http://www.icecast.org/projects/ices">
  <Playlist>
    <!-- This is the filename used as a playlist when using the builtin
        playlist handler. -->
    <File>playlist.txt</File>
    <!-- Set this to 0 if you don't want to randomize your playlist, and to
        1 if you do. -->
    <Randomize>1</Randomize>
    <!-- One of builtin, perl, or python. -->
    <Type>builtin</Type>
    <!-- Module name to pass to the playlist handler if using  perl or python.
        If you use the builtin playlist handler then this is ignored -->
    <Module>ices</Module>
    <!-- Set this to the number of seconds to crossfade between tracks.
         Leave out or set to zero to disable crossfading (the default).
    <Crossfade>5</Crossfade>
    -->
  </Playlist>

  <Execution>
    <!-- Set this to 1 if you want ices to launch in the background as a
         daemon -->
    <Background>0</Background>
    <!-- Set this to 1 if you want to see more verbose output from ices -->
    <Verbose>1</Verbose>
    <!-- This directory specifies where ices should put the logfile, cue file
        and pid file (if daemonizing). Don't use /tmp if you have l33t h4x0rz
         on your server. -->
    <BaseDirectory>/tmp</BaseDirectory>
  </Execution>

  <Stream>
    <Server>
      <!-- Hostname or ip of the icecast server you want to connect to -->
      <Hostname>localhost</Hostname>
      <!-- Port of the same -->
      <Port>8000</Port>
      <!-- Encoder password on the icecast server -->
      <Password>hackme</Password>
      <!-- Header protocol to use when communicating with the server.
           Shoutcast servers need "icy", icecast 1.x needs "xaudiocast", and
          icecast 2.x needs "http". -->
      <Protocol>http</Protocol>
    </Server>

    <!-- The name of the mountpoint on the icecast server -->
    <Mountpoint>/ices</Mountpoint>
    <!-- The name of the dumpfile on the server for your stream. DO NOT set
        this unless you know what you're doing.
    <Dumpfile>ices.dump</Dumpfile>
    -->
    <!-- The name of you stream, not the name of the song! -->
    <Name>Radio fuderenga do Ximbica</Name>
    <!-- Genre of your stream, be it rock or pop or whatever -->
    <Genre>Variados</Genre>
    <!-- Longer description of your stream -->
    <Description>Stream de hits totalmente excelente</Description>
    <!-- URL to a page describing your stream -->
    <URL>http://localhost/</URL>
    <!-- 0 if you don't want the icecast server to publish your stream on
        the yp server, 1 if you do -->
    <Public>0</Public>
```

```
    <!-- Stream bitrate, used to specify bitrate if reencoding, otherwise
        just used for display on YP and on the server. Try to keep it
        accurate -->
    <Bitrate>128</Bitrate>
    <!-- If this is set to 1, and ices is compiled with liblame support,
        ices will reencode the stream on the fly to the stream bitrate. -->
    <Reencode>0</Reencode>
    <!-- Number of channels to reencode to, 1 for mono or 2 for stereo -->
    <!-- Sampe rate to reencode to in Hz. Leave out for LAME's best choice
    <Samplerate>44100</Samplerate>
    -->
    <Channels>2</Channels>
  </Stream>
</ices:Configuration>
```

APÊNDICE E – Arquivo /usr/local/etc/openca/config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openca>
    <software_config>
        <!--
            ############################################################
                                  USAGE WARNING
            ############################################################

            If yo change this file then you must change all files in
            etc which has the suffix .template. Please do this with
            the script openca-configure.

            Example:
                    template: servers/ca.conf.template
                    openca-configure config.xml servers/ca.conf.template
servers/ca.conf

            If you don't do this then you have an inconsistent
            OpenCA installation. So this warning is serious.

            You can update all templates with a simple bash script.
            configure_etc.sh is such a script and demonstrates the
            usage of openca-configure.

            2003-Mar-12, Michael Bell <michael.bell@web.de>
        -->
        <prefix>@</prefix>
        <suffix>@</suffix>

        <!-- =============== -->
        <!-- general options -->
        <!-- =============== -->

        <option>
            <name>default_language</name>
            <value>en_GB</value>
        </option>
        <option>
            <name>default_charset</name>
            <value>utf-8</value>
        </option>
        <option>
            <!--
                 cert_chars says if you want utf8 or latin1
                 strings to be allowed in the fields of reqs
                 and certs. Can have values "UTF8" or "LATIN1".
                 If left empty or absent altogether, defaults to
                 LATIN1 strings.
              -->
            <name>cert_chars</name>
            <value>UTF8</value>
        </option>
       <option>
           <name>default_web_username</name>
           <value>admin</value>
       </option>
       <option>
           <name>default_web_password</name>
           <value>+afG3zQTJYIuPqJkz+OeXvjHOqQ</value>
       </option>
       <option>
            <!-- Name of your Organization (e.g, University of ... ) -->
            <name>organization</name>
            <value>CA do Ximbica</value>
```

```
</option>
<option>
    <!--
         Once cert_chars is UTF8 you may use
         strings in national languages here.
      -->
    <name>ca_organization</name>
    <value>CA do Ximbica</value>
</option>
<option>
    <!--
         Once cert_chars is UTF8 you may use
         strings in national languages here.
      -->
    <name>ca_locality</name>
    <value>Brasilia</value>
</option>
<option>
    <!--
         Once cert_chars is UTF8 you may use
         strings in national languages here.
      -->
    <name>ca_state</name>
    <value>DF</value>
</option>
<option>
    <!--
        please enter the ISO country code here
        DE, IT, PL, UK, US ...
        this country code is ALWAYS two characters long
      -->
    <name>ca_country</name>
    <value>BR</value>
</option>
<option>
    <name>sendmail</name>
    <value>/usr/lib/sendmail -n -t </value>
</option>
<option>
    <name>send_mail_automatic</name>
    <value>no</value>
</option>
<option>
    <!-- Use this for the Web Interface email link -->
    <name>support_mail_address</name>
    <value>support@pki.openca.org</value>
</option>
<option>
    <!-- Use this for the EMAIL sending address -->
    <name>service_mail_account</name>
    <value>admin@pki.openca.org</value>
</option>
<option>
    <name>policy_link</name>
    <value>https://centosvm/pki/pub/policy.html</value>
</option>

<!-- ======================== -->
<!-- web server configuration -->
<!-- ======================== -->
<option>
    <name>httpd_protocol</name>
    <value>https</value>
</option>
<option>
    <name>httpd_host</name>
    <value>centosvm</value>
</option>
```

```
        <option>
            <!-- please include the colon if you specify a port        -->
            <!-- please remember this is dependend from httpd_protocol -->
            <name>httpd_port</name>
            <value>:443</value>
        </option>
        <!--
        <option>
            <name>menu_logo_left</name>
            <value>
              - Here you can put references to the logo, you can use
                   any html reference you want but please keep in mind that:
                   no <> are allowed, use instead &lt; and &gt; rispectively.

                example:
                   &lt;img src="https://xyz.org/mylogo.jpg" alt="XYZ Logo"/&gt;
             -
            </value>
        </option>
        <option>
            <name>menu_logo_right</name>
                &lt;a href="__HTDOCS_PREFIX__/thanks.html"&gt;
                   &lt;img src="__HTDOCS_PREFIX__/images/openca-logo.png"
alt="OpenCA Logo"/&gt;
                &lt;/a&gt;
            <value></value>
        </option>
        -->
        <option>
            <!--
                You can add more CDPs here. Please enter one CDP per line.
                This is the content of an OpenSSL configuration section.
                Example:
URI.1=http://pki.openca.org/pub/crl/cacrl.crl
URI.2=ldap://pki.openca.org/cn=CA,o=OpenCA,c=US
URI.3=http://mirror.pki.openca.org/pub/crl/cacrl.crl
URI.4=ldap://mirror.pki.openca.org/cn=CA,o=OpenCA,c=US
             -->
            <name>CRLDistributionPoints</name>
            <value>
URI.1=http://centosvm/pki/pub/crl/cacrl.crl
            </value>
        </option>
        <option>
            <!--
                You can add Authority Info Access URIs here.
                Please enter one URI per line.
                This is the content of an OpenSSL configuration section.
            -->
            <name>authInfoAccess</name>
            <value>
authorityInfoAccess=caIssuers;URI:http://centosvm/pki/pub/cacert/cacert.crt,OCSP;UR
I:http://centosvm:2560/,prqpServer;URI:http://centosvm:830/
           </value>
        </option>
        <option>
            <name>NS_CRLDistributionPoint</name>
            <value>http://centosvm/pki/pub/crl/cacrl.crl</value>
        </option>

        <!-- ========================= -->
        <!-- ldap server configuration -->
        <!-- ========================= -->
        <option>
            <name>ldap_protocol</name>
            <value>3</value>
        </option>
        <option>
```

```
    <name>ldap_host</name>
    <value>centosvm</value>
</option>
<option>
    <name>ldap_port</name>
    <value>389</value>
</option>
<option>
    <name>ldaproot</name>
    <value></value>
</option>
<option>
    <name>ldaprootpwd</name>
    <value></value>
</option>
<option>
    <name>useLDAP</name>
    <value>no</value>
</option>
<option>
    <name>update_ldap_automatic</name>
    <value>no</value>
</option>

<!-- ====================== -->
<!-- database configuration -->
<!-- ====================== -->
<option>
    <name>dbmodule</name>
    <!-- you can use DB or DBI -->
    <value>DBI</value>
</option>
<option>
    <name>db_type</name>
    <value>mysql</value>
</option>
<option>
    <name>db_name</name>
    <value>openca</value>
</option>
<option>
    <name>db_host</name>
    <value>localhost</value>
</option>
<option>
    <name>db_port</name>
    <value>3306</value>
</option>
<option>
    <name>db_user</name>
    <value>openca</value>
</option>
<option>
    <name>db_passwd</name>
    <value>openca</value>
</option>
<option>
    <name>db_namespace</name>
    <!--
       a namespace is prefix in front of every table
       Example: table user1
                ==>
                select * from user1.certificate;
       This is not required for MySQL, PostgreSQL and IBM DB2.
       Nevertheless all supported database can use such namespaces
       and it is the default behaviour of Oracle. Oracle uses as
       namespace usually the name of the database.
      -->
```

```
    <value></value>
</option>

<!-- ==================== -->
<!-- module configuration -->
<!-- ==================== -->
<option>
    <name>module_shift</name>
    <!-- 8 bits are enough for IDs from 0 to 255    -->
    <!-- please remember that 0 is the ID of the CA -->
    <value>8</value>
</option>
<option>
    <name>ra_module_id</name>
    <value>1</value>
</option>
<option>
    <name>ldap_module_id</name>
    <value>2</value>
</option>
<option>
    <name>node_module_id</name>
    <value>3</value>
</option>
<option>
    <name>pub_module_id</name>
    <value>32</value>
</option>
<option>
    <name>scep_module_id</name>
    <value>33</value>
</option>
<option>
    <name>batch_module_id</name>
    <value>128</value>
</option>

<!-- ============================== -->
<!-- configuration of relative paths -->
<!-- ============================== -->

<option>
    <name>batch_htdocs_url_prefix</name>
    <value>/pki/batch</value>
</option>
<option>
    <name>batch_cgi_url_prefix</name>
    <value>/cgi-bin/pki/batch</value>
</option>
<option>
    <name>ca_htdocs_url_prefix</name>
    <value>/pki/ca</value>
</option>
<option>
    <name>ca_cgi_url_prefix</name>
    <value>/cgi-bin/pki/ca</value>
</option>
<option>
    <name>node_htdocs_url_prefix</name>
    <value>/pki/node</value>
</option>
<option>
    <name>node_cgi_url_prefix</name>
    <value>/cgi-bin/pki/node</value>
</option>
<option>
    <name>ra_htdocs_url_prefix</name>
    <value>/pki/ra</value>
```

```
</option>
<option>
    <name>ra_cgi_url_prefix</name>
    <value>/cgi-bin/pki/ra</value>
</option>
<option>
    <name>ldap_htdocs_url_prefix</name>
    <value>/pki/ldap</value>
</option>
<option>
    <name>ldap_cgi_url_prefix</name>
    <value>/cgi-bin/pki/ldap</value>
</option>
<option>
    <name>pub_htdocs_url_prefix</name>
    <value>/pki/pub</value>
</option>
<option>
    <name>pub_cgi_url_prefix</name>
    <value>/cgi-bin/pki/pub</value>
</option>
<option>
    <name>scep_cgi_url_prefix</name>
    <value>/cgi-bin/pki/scep</value>
</option>

<!-- ==================================
     configuration of Node Update paths
     ================================== -->

<option>
    <name>perl_module_prefix</name>
    <value>/usr/local/lib/openca/perl_modules</value>
</option>
<option>
    <name>find_command</name>
    <value>/usr/bin/find %d -name %n</value>
</option>
<option>
    <name>web_get_command</name>
    <value>/usr/bin/wget -O - %h</value>
</option>

<!-- ==================================
     configuration of Node Update paths
     ================================== -->

<option>
    <name>batch_htdocs_fs_prefix</name>
    <value>/var/www/html/pki/batch</value>
</option>
<option>
    <name>batch_cgi_fs_prefix</name>
    <value>/var/www/cgi-bin/pki/batch</value>
</option>
<option>
    <name>ca_htdocs_fs_prefix</name>
    <value>/var/www/html/pki/ca</value>
</option>
<option>
    <name>ca_cgi_fs_prefix</name>
    <value>/var/www/cgi-bin/pki/ca</value>
</option>
<option>
    <name>node_htdocs_fs_prefix</name>
    <value>/var/www/html/pki/node</value>
</option>
<option>
```

```
    <name>node_cgi_fs_prefix</name>
    <value>/var/www/cgi-bin/pki/node</value>
 </option>
 <option>
    <name>ra_htdocs_fs_prefix</name>
    <value>/var/www/html/pki/ra</value>
 </option>
 <option>
    <name>ra_cgi_fs_prefix</name>
    <value>/var/www/cgi-bin/pki/ra</value>
 </option>
 <option>
    <name>ldap_htdocs_fs_prefix</name>
    <value>/var/www/html/pki/ldap</value>
 </option>
 <option>
    <name>ldap_cgi_fs_prefix</name>
    <value>/var/www/cgi-bin/pki/ldap</value>
 </option>
 <option>
    <name>pub_htdocs_fs_prefix</name>
    <value>/var/www/html/pki/pub</value>
 </option>
 <option>
    <name>pub_cgi_fs_prefix</name>
    <value>/var/www/cgi-bin/pki/pub</value>
 </option>
 <option>
    <name>scep_cgi_fs_prefix</name>
    <value>/var/www/cgi-bin/pki/scep</value>
 </option>

 <!-- ====================== -->
 <!-- configuration of SCEP -->
 <!-- ====================== -->

 <option>
    <name>SCEP_RA_CERT</name>
    <value></value>
 </option>
 <option>
    <name>SCEP_RA_KEY</name>
    <value></value>
 </option>
 <option>
    <name>SCEP_RA_PASSWD</name>
    <value></value>
 </option>

 <!-- ====================== -->
 <!-- general configuration -->
 <!-- ====================== -->

 <option>
   <name>USE_LOAS</name>
   <value>yes</value>
</option>
<option>
    <name>prefix</name>
    <value>/usr/local</value>
 </option>
 <option>
    <name>bindir</name>
    <value>/usr/local/bin</value>
 </option>
 <option>
    <name>etc_prefix</name>
    <value>/usr/local/etc/openca</value>
```

```
</option>
<option>
    <name>lib_prefix</name>
    <value>/usr/local/lib/openca</value>
</option>
<option>
    <name>var_prefix</name>
    <value>/usr/local/var/openca</value>
</option>
<option>
    <name>batch_prefix</name>
    <value>batch</value>
</option>
<option>
    <name>ca_prefix</name>
    <value>ca</value>
</option>
<option>
    <name>ldap_prefix</name>
    <value>ldap</value>
</option>
<option>
    <name>node_prefix</name>
    <value>node</value>
</option>
<option>
    <name>pub_prefix</name>
    <value>pub</value>
</option>
<option>
    <name>ra_prefix</name>
    <value>ra</value>
</option>
<option>
    <name>scep_prefix</name>
    <value>scep</value>
</option>


<!-- ========================== -->
<!-- dataexchange configuration -->
<!-- ========================== -->

<!-- there are several templates available today                    -->
<!--   0. no dataexchange configure - the default                   -->
<!--        this makes only sense for an all in one box             -->
<!--        it is strongly recommended to use this only for testing -->
<!--   1. the node acts as CA only                                  -->
<!--        the node exports to one or several RAs only             -->
<!--        the node can export to LDAP too                         -->
<!--   2. the node acts as RA only                                  -->
<!--        the node exchange data with a CA and public/scep        -->
<!--        the node can act as LDAP too                            -->
<!--        the node can export to LDAP too                         -->
<!--   3. the node acts as public/scep only                         -->
<!--        the node exchange data with a RA                        -->
<!--   4. the node acts as LDAP only                                -->
<!--        the node receives data from CA or RA                    -->
<!--   5. the node acts as public/scep and RA                       -->
<!--        the node echanges data with a CA only                   -->
<!--        no support for dataexchange with additional LDAP        -->
<!--   6. the node acts as RA and CA                                -->
<!--        the node exchange data with public/scep                 -->
<!--        the node can export to LDAP too                         -->
<!--                                                                -->
<!--   LDAP is only relevant if it is the only protocol on the node -->
```

```
<!--   0. no dataexchange configure - the default                    -->
    <option>
      <name>enroll_ca_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crl_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_ca_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>download_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>download_crl_states</name>
      <value></value>
    </option>
    <option>
      <name>download_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_csr_states</name>
      <value></value>
    </option>

<!--   1. the node acts as CA only                                   -->
<!--
```

```
    <option>
      <name>enroll_ca_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_crl_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_crr_states</name>
      <value>ARCHIVED DELETED APPROVED</value>
    </option>
    <option>
      <name>enroll_csr_states</name>
      <value>ARCHIVED DELETED</value>
    </option>
    <option>
      <name>enroll_mail_states</name>
      <value>CRINS DEFAULT</value>
    </option>
    <option>
      <name>receive_crr_states</name>
      <value>APPROVED</value>
    </option>
    <option>
      <name>receive_csr_states</name>
      <value>APPROVED</value>
    </option>
    <option>
      <name>download_ca_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>download_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>download_crl_states</name>
      <value></value>
    </option>
    <option>
      <name>download_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_csr_states</name>
      <value></value>
    </option>
  -->

<!--   2. the node acts as RA only                                    -->
<!--
```

```
    <option>
      <name>enroll_ca_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_crl_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_crr_states</name>
      <value>ARCHIVED DELETED APPROVED SIGNED PENDING NEW</value>
    </option>
    <option>
      <name>enroll_csr_states</name>
      <value>ARCHIVED DELETED</value>
    </option>
    <option>
      <name>enroll_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_crr_states</name>
      <value>PENDING NEW</value>
    </option>
    <option>
      <name>receive_csr_states</name>
      <value>PENDING RENEW NEW</value>
    </option>
    <option>
      <name>download_ca_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crl_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crr_states</name>
      <value>ARCHIVED DELETED APPROVED</value>
    </option>
    <option>
      <name>download_csr_states</name>
      <value>ARCHIVED DELETED</value>
    </option>
    <option>
      <name>download_mail_states</name>
      <value>CRINS DEFAULT</value>
    </option>
    <option>
      <name>upload_crr_states</name>
      <value>APPROVED</value>
    </option>
    <option>
      <name>upload_csr_states</name>
      <value>APPROVED</value>
    </option>
  -->

<!--   3. the node acts as public/scep only                          -->
<!--
```

```
    <option>
      <name>enroll_ca_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crl_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_ca_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crl_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crr_states</name>
      <value>ARCHIVED DELETED APPROVED SIGNED PENDING RENEW NEW</value>
    </option>
    <option>
      <name>download_csr_states</name>
      <value>ARCHIVED DELETED</value>
    </option>
    <option>
      <name>download_mail_states</name>
      <value>CRINS DEFAULT</value>
    </option>
    <option>
      <name>upload_crr_states</name>
      <value>NEW</value>
    </option>
    <option>
      <name>upload_csr_states</name>
      <value>RENEW NEW</value>
    </option>
  -->

<!--   4. the node acts as LDAP only                                  -->
<!--
```

```
    <option>
      <name>enroll_ca_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crl_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_ca_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crl_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crr_states</name>
      <value>ARCHIVED DELETED APPROVED SIGNED PENDING RENEW NEW</value>
    </option>
    <option>
      <name>download_csr_states</name>
      <value>ARCHIVED DELETED</value>
    </option>
    <option>
      <name>download_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_csr_states</name>
      <value></value>
    </option>
  -->

<!--   5. the node acts as public/scep and RA                        -->
<!--
```

```
    <option>
      <name>enroll_ca_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crl_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>enroll_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_ca_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crl_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>download_crr_states</name>
      <value>ARCHIVED DELETED APPROVED</value>
    </option>
    <option>
      <name>download_csr_states</name>
      <value>ARCHIVED DELETED</value>
    </option>
    <option>
      <name>download_mail_states</name>
      <value>CRINS DEFAULT</value>
    </option>
    <option>
      <name>upload_crr_states</name>
      <value>APPROVED</value>
    </option>
    <option>
      <name>upload_csr_states</name>
      <value>APPROVED</value>
    </option>
  -->

<!--   6. the node acts as RA and CA                                  -->
<!--
```

```
    <option>
      <name>enroll_ca_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_certificate_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_crl_states</name>
      <value>VALID</value>
    </option>
    <option>
      <name>enroll_crr_states</name>
      <value>ARCHIVED DELETED APPROVED SIGNED PENDING NEW</value>
    </option>
    <option>
      <name>enroll_csr_states</name>
      <value>ARCHIVED DELETED</value>
    </option>
    <option>
      <name>enroll_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>receive_crr_states</name>
      <value>PENDING NEW</value>
    </option>
    <option>
      <name>receive_csr_states</name>
      <value>PENDING RENEW NEW</value>
    </option>
    <option>
      <name>download_ca_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>download_certificate_states</name>
      <value></value>
    </option>
    <option>
      <name>download_crl_states</name>
      <value></value>
    </option>
    <option>
      <name>download_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_csr_states</name>
      <value></value>
    </option>
    <option>
      <name>download_mail_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_crr_states</name>
      <value></value>
    </option>
    <option>
      <name>upload_csr_states</name>
      <value></value>
    </option>
  -->

<!-- these are the devices for the default dataexchange -->
<option>
```

```
          <name>dataexchange_device_up</name>
          <value>/dev/fd0</value>
        </option>
        <option>
          <name>dataexchange_device_down</name>
          <value>/dev/fd0</value>
        </option>
        <option>
          <name>dataexchange_device_local</name>
          <value>/tmp/openca_local</value>
        </option>

    </software_config>
</openca>
```

APÊNDICE F – Arquivo /var/www/html/index.php

```
<html>
<title>
Pagina de Autenticacao - v1.0
</title>
<body>
<?php

//echo $_SERVER['DN'];

$common_name_cert = (string)$_SERVER['DN'];

if( ($CN = strpos($common_name_cert, 'CN=')) !== FALSE )
   $nome_cert = substr($common_name_cert, $CN + 3);

?>
OlÃ¡,
<?php

printf('%s',$nome_cert);

?>!<br>

<?php

if ($_SERVER['VERIFIED'] = 'SUCCESS')
    printf('Seu certificado foi autenticado com sucesso!');

?><br>
Por favor, digite as suas credenciais de acesso:

<form action="autenticacao.php" method="post">
Usuario: <input type="text" name="usuario">
Senha: <input type="password" name="senha">
<input type="submit">

<?php
//$salt = "corra para as montanhas";

?>


</form>
</body>
</html>
```

APÊNDICE G – Arquivo /var/www/html/autenticacao.php

```
<html>
<body>

<?php $senha_cripto = sha1($_POST["senha"]);

$usuario = $_POST["usuario"];

$con_banco = mysql_connect('localhost','root','asdqwe');
if (!$con_banco)
  {
  die('Problemas ao conectar com o banco: ' . mysql_error());
  }

mysql_select_db("monografia", $con_banco);

$result = mysql_query("SELECT count(*) as validacao FROM autenticacao where
usuario='$usuario' and senha='$senha_cripto'");

$row = mysql_fetch_array($result);

//echo $row['validacao'];

mysql_close($con_banco);

if ($row['validacao'] == 1) {
   echo 'VocÃª foi autenticado com sucesso! <br> Por favor, clique no link
abaixo para prosseguir:<br>';
   echo '<a href="http://192.168.1.9:8000/ices">Stream de Audio</a><br>';
}

else {
   echo 'AutenticaÃ§Ã£o falhou!<br>';
   echo '<INPUT Type="button" VALUE="Voltar" onClick="history.go(-1);return
true;">';
}
?>

</body>
</html>
```