**ESQUEMA DE PLANO DE AULA PRÁTICA**

**TAREFA 6** "Comunicação Apache com SSL"

Tópico no Moodle: Segurança na comunicação entre Clientes e Servidores Web com o protocolo criptográfico SSL, utilizando servidor web Apache e OpenSSL.

Tempo de Aula: 1 hora-aula

Data:

Grupo: <Aluno\_1>

 <Aluno\_2>

**ASSUNTO**

O **Hypertext Transfer Protocol (HTTP**) - *Protocolo de Transferência de*[*Hipertexto*](http://pt.wikipedia.org/wiki/Hipertexto) - é um[protocolo de comunicação](http://pt.wikipedia.org/wiki/Protocolo_de_comunica%C3%A7%C3%A3o) (na [camada de aplicação](http://pt.wikipedia.org/wiki/Camada_de_aplica%C3%A7%C3%A3o) segundo o [Modelo OSI](http://pt.wikipedia.org/wiki/Modelo_OSI)) utilizado para sistemas de informação de hipermedia distribuídos e colaborativos.[1](http://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol%22%20%5Cl%20%22cite_note-T._Berners-Lee_ekjhkjgi.C3.A7ugiougiouguguigt_all.2C_1996-1) Seu uso para a obtenção de recursos interligados levou ao estabelecimento da [World Wide Web](http://pt.wikipedia.org/wiki/World_Wide_Web).

Coordenado pela [World Wide Web Consortium](http://pt.wikipedia.org/wiki/World_Wide_Web_Consortium) e a [Internet Engineering Task Force](http://pt.wikipedia.org/wiki/Internet_Engineering_Task_Force), culminou na publicação de uma série de *[Requests for Comments](http://pt.wikipedia.org/wiki/Request_for_Comments%22%20%5Co%20%22Request%20for%20Comments)*; mais notavelmente o [RFC 2616](http://tools.ietf.org/html/rfc2616), de junho de 1999, que definiu o HTTP/1.1.

Normalmente, este protocolo utiliza a [porta](http://pt.wikipedia.org/wiki/Porta_%28inform%C3%A1tica%29) 80 e é usado para a comunicação de [sites web](http://pt.wikipedia.org/wiki/Site), comunicando na linguagem [HTML](http://pt.wikipedia.org/wiki/HTML).

O HyperText Transfer Protocol é um protocolo de aplicação responsável pelo tratamento de pedidos e respostas entre cliente e servidor na[World Wide Web](http://pt.wikipedia.org/wiki/World_Wide_Web). Ele surgiu da necessidade de distribuir informações pela [Internet](http://pt.wikipedia.org/wiki/Internet) e para que essa distribuição fosse possível foi necessário criar uma forma padronizada de comunicação entre os clientes e os servidores da Web e entendida por todos os computadores ligados à Internet. Com isso, o protocolo HTTP passou a ser utilizado para a comunicação entre computadores na Internet e a especificar como seriam realizadas as transações entre clientes e servidores, através do uso de regras básicas.

O problema que surge aqui é a questão da segurança na comunicação entre cliente (navegador) e servidor Web. HTTP com segurança é chamado HTTPS, que aparece bastante no momento de certas autenticações do usuário na máquina cliente, quando precisa fazer sua autenticação perante o servidor, ou quando o servidor envia seu certificado para o cliente no navegador, para esse ter certeza que está acessando o servidor que realmente deseja.

**HTTPS** (***HyperText Transfer Protocol Secure*** ), é uma implementação do protocolo [HTTP](http://pt.wikipedia.org/wiki/HTTP) sobre uma camada adicional de segurança que utiliza o protocolo [SSL](http://pt.wikipedia.org/wiki/SSL)/[TLS](http://pt.wikipedia.org/wiki/TLS). Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão [criptografada](http://pt.wikipedia.org/wiki/Criptografia) e que se verifique a autenticidade do [servidor](http://pt.wikipedia.org/wiki/Servidor) e do cliente por meio de certificados digitais. A porta [TCP](http://pt.wikipedia.org/wiki/TCP) usada por norma para o protocolo HTTPS é a 443.

O protocolo HTTPS é utilizado, em regra, quando se deseja evitar que a informação transmitida entre o [cliente](http://pt.wikipedia.org/wiki/Cliente) e o [servidor](http://pt.wikipedia.org/wiki/Servidor) seja visualizada por terceiros, como por exemplo no caso de compras *online*. A existência na barra de tarefas de um cadeado (que pode ficar do lado esquerdo ou direito, dependendo do navegador utilizado) demonstra a certificação de página segura ([SSL](http://pt.wikipedia.org/wiki/SSL)). A existência desse certificado indica o uso do protocolo HTTPS e que a comunicação entre o browser e o servidor se dará de forma segura. Para verificar a identidade do servidor é necessário abrir esse certificado com um duplo clique no cadeado para exibição do certificado. Conexões HTTPS são frequentemente usadas para transações de pagamentos na World Wide Web e para transações sensíveis em sistemas de informação corporativos. Porém, o HTTPS não deve ser confundido com o menos utilizado protocolo "Secure HTTP" (S-HTTP), especificado na [RFC 2660](http://tools.ietf.org/html/rfc2660). A ideia principal do HTTPS é criar um canal seguro sobre uma rede insegura. Isso garante uma proteção razoável de pessoas que realizam escutas ilegais (os chamados [eavesdroppers](http://pt.wikipedia.org/wiki/Eavesdropping%22%20%5Co%20%22Eavesdropping)) e de ataques de homem-no-meio ([man-in-the-middle](http://pt.wikipedia.org/wiki/Ataque_man-in-the-middle%22%20%5Co%20%22Ataque%20man-in-the-middle)), dado que a cifragem foi adequadamente utilizada e que o certificado do servidor é verificável e confiável.

As URLs HTTPS começam com "https://" e utilizam a porta 443 como padrão, enquanto as [URLs](http://pt.wikipedia.org/wiki/URL%22%20%5Co%20%22URL) [HTTP](http://pt.wikipedia.org/wiki/HTTP) começam com "http://" e utilizam a [porta](http://pt.wikipedia.org/wiki/Anexo%3ALista_de_portas_de_protocolos) 80 como padrão. HTTP é inseguro e sujeito a [ataques de homem-no-meio](http://pt.wikipedia.org/wiki/Ataque_man-in-the-middle) e [escutas ilegais](http://pt.wikipedia.org/wiki/Eavesdropping), que podem levar a atacantes ganharem acesso a contas de páginas na web e a informações sensíveis. O HTTPS foi projetado para proteger contra esses ataques e é considerado seguro contra eles.

### Configuração do Servidor

Para preparar uma página web de modo a aceitar conexões HTTPS, o administrador deve criar um [certificado de chave pública](http://pt.wikipedia.org/wiki/Certificado_digital) para o servidor web. Esse certificado deve ser assinado por uma [autoridade de certificação](http://pt.wikipedia.org/wiki/Autoridade_de_Certifica%C3%A7%C3%A3o) confiável para que o navegador aceite a conexão e não exiba avisos. A autoridade certifica que o proprietário do certificado é o operador do servidor web que o apresenta. Os navegadores web são geralmente distribuídos com uma lista de autoridades de certificação de assinatura para que elas possam verificar certificados assinados por eles.

 **BIBLIOGRAFIA BÁSICA E INSTALAÇÕES**

Instalando Apache 2.2.22 no Ubuntu: [**http://ftp.unicamp.br/pub/apache/**](http://ftp.unicamp.br/pub/apache/)

[**http://www.us.apache.org/dist/**](http://www.us.apache.org/dist/)

Instalando OpenSSL: <http://www.openssl.org/>

Apache com SSL, utiliza o OpenSSL. A licença é BSD-Style e pode ser utilizada para fins comerciais e não comerciais.

Esse sistema foi criado pelo desenvolvedor Ben Laury, participante direto dos projetos Apache e OpenSSL.

SSL

**OBJETIVOS DA AULA**

Conhecer como se configura HTTPS no servidor web Apache com o OpenSSL.
Compreender como se constrói e se instala um certificado de servidor com o OpenSSL, o qual é fornecido ao usuário através do navegador.

**SELEÇÃO DE CONTEÚDO – A PRÁTICA**

Um conteúdo mínimo, consistindo de um roteiro de aula, com Apache e OpenSSL, visando o objetivo acima, pode ser colocada como:

 **MOSTRE SEU EXPERIMENTO (Deposite a tarefa no Moodle).**

Abra terminal (Ctrl Alt+t) no Ubuntu. As senhas são aluno/network ou para super-usuário use ufliictsc. Comece.

É necessário que o usuário efetue o logon com o login de administrador, o “root”.

$sudo su (). Entre com a senha de usuário.

1° Passo: Caso, exista um servidor Proxy de Web, colocado na saída da sua rede local. Um servidor Proxy bem conhecido é o Squid (Linux), que não será aqui reportado.

Para configurar o **apt-get autenticado em servidor proxy (o apache e o SSL, serão instalados via apt-get no que segue)**, é preciso acrescentar as linhas no arquivo /etc/profile, usando o pico ficaria o seguinte comando:

# gedit /etc/profile

Acrescentando as seguintes linhas com as devidas alterações do endereço ip e porta da rede:

http\_proxy=”http://ip.do.servidor.proxy:porta”

ftp\_proxy=”http://ip.do.servidor.proxy:porta”

export http\_proxy ftp\_proxy

2° Passo:

É necessário fazer a atualização dos pacotes de instalações de serviços, o comando é o seguinte:

# apt-get update

3° Passo:

Feita essa atualização, será iniciada a instalação OpenSSL. O OpenSSL é uma implementação de código aberto dos protocolos SSL e TLS. A biblioteca implementa as funções básicas de criptografia e disponibiliza várias funções utilitárias. Para a instalação usamos o seguinte comando:

# apt-get install openssl

4° Passo:

Agora será instalado o Apache2 mais o complemento necessário para a ativação do SSL, o comando será o seguinte:

# apt-get install apache2 apache2.2-common

Se houver pedidos de confirmação durante este procedimento, o usuário deve confirmar com “s”.

Em seguida será feita a habilitação do módulo SSL no Apache com o seguinte comando:

# a2enmod ssl

5° Passo:

O Apache2 com SSL funciona através de um certificado reconhecido na web. Para gerar o certificado (CSR) será necessário a criação de uma chave, essa chave deverá conter no mínimo 08 caracteres, a criptografia usada nessa chave é a DES3, essa chave ficará no arquivo server.key. De dentro do diretório onde ela ficará armazenada, o “/etc/ssl/private”, o usuário deve digitar o comando abaixo para criação da chave:

# openssl genrsa -des3 -out server.key 1024

6° Passo:

Com a chave criada é importante acrescentar informações. Após digitar o comando abaixo, ele pedirá a chave e para preencher alguns dados, são os dados que serão visualizados referentes ao criador do certificado:

# openssl req -new -key server.key -out server.csr

Toda vez que o usuário reiniciar o Apache2 ele pedirá a chave. Isso faz com que caso haja um restart por algum motivo, o serviço não suba pois dependerá da presença humana para a digitação.

Os comandos abaixo fazem com que o Apache2 não peça a chave em caso de restart do serviço, ou seja, ele faz a autenticação automaticamente. O usuário deve entrar no diretório “/etc/ssl/private” e executar o comando abaixo:

# openssl rsa -in server.key -out server.key.insecure

Executado o comando acima, deverá feita a substituição do arquivo atual pelo criado. Antes desse passo o usuário deve fazer uma cópia de segurança do arquivo atual ou renomear o arquivo “server.key”. Com o comando abaixo será feita a renomeação e o arquivo permanecerá na mesma pasta:

# mv server.key server.bak

Após a renomeação do arquivo, será feita a renomeação do arquivo criado, que fará com que não seja mais pedida a chave quando houver restart do serviço. O comando será o seguinte:

# mv server.key.insecure server.key

7° Passo:

O certificado (auto-assinado) será criado com o comando abaixo:

# openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

8° Passo:

Será necessário efetuar a instalação da chave e do certificado anteriormente criados com os seguintes comandos:

# cp server.crt /etc/ssl/certs

# cp server.key /etc/ssl/private

No comando para instalação do server.key vai dar uma mensagem, isso porque foi criado no próprio diretório onde ele seria instalado, se tivesse sido criado em outro diretório não haveria a mensagem.

9° Passo:

Repare os diretórios do Apache2:

/etc/apache2/sites-available/ contém os sites disponíveis
 arquivos: default e default-ssl

/etc/apache2/sites-available/ contém os sites habilitados
 (um site pode não estar habilitado num momento)

Agora serão efetuadas algumas alterações de configuração no Apache editando o arquivo /etc/apache2/sites-available/default.

Usando o editor gedit (nano, vim, pico) o comando será o seguinte:

# gedit etc/apache2/sites-available/default (save)

---------------------------------------------------------------

**Arquivo default**

<VirtualHost \*:80>

DocumentRoot /var/www/teste/html

Options Indexes FollowSymLinks Multiviews

ErrorLog /var/www/teste/html/logs/error.log

CustomLog /var/www/teste/html/logs/access.log combined

</VirtualHost>

<VirtualHost \*:443>

DocumentRoot /var/www/teste/ssl

Options Indexes FollowSymLinks Multiviews

ErrorLog /var/www/teste/ssl/logs/error.log

CustomLog /var/www/teste/ssl/logs/access.log combined

SSLEngine on

SSLCertificateFile /etc/ssl/certs/server.crt

SSLCertificateKeyFile /etc/ssl/private/server.key

</VirtualHost>

-----------------------------------------------------------

# gedit etc/apache2/sites-available/default-ssl (save)

Arquivo default-ssl

-----------------------------------------------------------

<IfModule mod\_ssl.c>

<VirtualHost \_default\_:443>

 ServerAdmin webmaster@localhost

 DocumentRoot /var/www/teste/ssl

 <Directory />

 Options FollowSymLinks

 AllowOverride None

 </Directory>

 <Directory /var/www/teste/ssl>

 Options Indexes FollowSymLinks MultiViews

 AllowOverride None

 Order allow,deny

 allow from all

 </Directory>

 ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/

 <Directory "/usr/lib/cgi-bin">

 AllowOverride None

 Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch

 Order allow,deny

 Allow from all

 </Directory>

 ErrorLog ${APACHE\_LOG\_DIR}/error.log

 # Possible values include: debug, info, notice, warn, error, crit,

 # alert, emerg.

 LogLevel warn

 CustomLog ${APACHE\_LOG\_DIR}/ssl\_access.log combined

 # SSL Engine Switch:

 # Enable/Disable SSL for this virtual host.

# SSLEngine on

 # A self-signed (snakeoil) certificate can be created by installing

 # the ssl-cert package. See

 # /usr/share/doc/apache2.2-common/README.Debian.gz for more info.

 # If both key and certificate are stored in the same file, only the

 # SSLCertificateFile directive is needed.

# SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem

# SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

 # Server Certificate Chain:

 # Point SSLCertificateChainFile at a file containing the

 # concatenation of PEM encoded CA certificates which form the

 # certificate chain for the server certificate. Alternatively

 # the referenced file can be the same as SSLCertificateFile

 # when the CA certificates are directly appended to the server

 # certificate for convinience.

 #SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt

 # Certificate Authority (CA):

 # Set the CA certificate verification path where to find CA

 # certificates for client authentication or alternatively one

 # huge file containing all of them (file must be PEM encoded)

 # Note: Inside SSLCACertificatePath you need hash symlinks

 # to point to the certificate files. Use the provided

 # Makefile to update the hash symlinks after changes.

 #SSLCACertificatePath /etc/ssl/certs/

 #SSLCACertificateFile /etc/apache2/ssl.crt/ca-bundle.crt

 # Certificate Revocation Lists (CRL):

 # Set the CA revocation path where to find CA CRLs for client

 # authentication or alternatively one huge file containing all

 # of them (file must be PEM encoded)

 # Note: Inside SSLCARevocationPath you need hash symlinks

 # to point to the certificate files. Use the provided

 # Makefile to update the hash symlinks after changes.

 #SSLCARevocationPath /etc/apache2/ssl.crl/

 #SSLCARevocationFile /etc/apache2/ssl.crl/ca-bundle.crl

 # Client Authentication (Type):

 # Client certificate verification type and depth. Types are

 # none, optional, require and optional\_no\_ca. Depth is a

 # number which specifies how deeply to verify the certificate

 # issuer chain before deciding the certificate is not valid.

 #SSLVerifyClient require

 #SSLVerifyDepth 10

 # Access Control:

 # With SSLRequire you can do per-directory access control based

 # on arbitrary complex boolean expressions containing server

 # variable checks and other lookup directives. The syntax is a

 # mixture between C and Perl. See the mod\_ssl documentation

 # for more details.

 #<Location />

 #SSLRequire ( %{SSL\_CIPHER} !~ m/^(EXP|NULL)/ \

 # and %{SSL\_CLIENT\_S\_DN\_O} eq "Snake Oil, Ltd." \

 # and %{SSL\_CLIENT\_S\_DN\_OU} in {"Staff", "CA", "Dev"} \

 # and %{TIME\_WDAY} >= 1 and %{TIME\_WDAY} <= 5 \

 # and %{TIME\_HOUR} >= 8 and %{TIME\_HOUR} <= 20 ) \

 # or %{REMOTE\_ADDR} =~ m/^192\.76\.162\.[0-9]+$/

 #</Location>

 # SSL Engine Options:

 # Set various options for the SSL engine.

 # o FakeBasicAuth:

 # Translate the client X.509 into a Basic Authorisation. This means that

 # the standard Auth/DBMAuth methods can be used for access control. The

 # user name is the `one line' version of the client's X.509 certificate.

 # Note that no password is obtained from the user. Every entry in the user

 # file needs this password: `xxj31ZMTZzkVA'.

 # o ExportCertData:

 # This exports two additional environment variables: SSL\_CLIENT\_CERT and

 # SSL\_SERVER\_CERT. These contain the PEM-encoded certificates of the

 # server (always existing) and the client (only existing when client

 # authentication is used). This can be used to import the certificates

 # into CGI scripts.

 # o StdEnvVars:

 # This exports the standard SSL/TLS related `SSL\_\*' environment variables.

 # Per default this exportation is switched off for performance reasons,

 # because the extraction step is an expensive operation and is usually

 # useless for serving static content. So one usually enables the

 # exportation for CGI and SSI requests only.

 # o StrictRequire:

 # This denies access when "SSLRequireSSL" or "SSLRequire" applied even

 # under a "Satisfy any" situation, i.e. when it applies access is denied

 # and no other module can change it.

 # o OptRenegotiate:

 # This enables optimized SSL connection renegotiation handling when SSL

 # directives are used in per-directory context.

 SSLEngine on

# SSLCertificateFile /etc/apache2/apache.pem

 SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire

 SSLCertificateFile /etc/ssl/certs/server.crt

 SSLCertificatekeyFile /etc/ssl/private/server.key

 <FilesMatch "\.(cgi|shtml|phtml|php)$">

 SSLOptions +StdEnvVars

 </FilesMatch>

 <Directory /usr/lib/cgi-bin>

 SSLOptions +StdEnvVars

 </Directory>

 # SSL Protocol Adjustments:

 # The safe and default but still SSL/TLS standard compliant shutdown

 # approach is that mod\_ssl sends the close notify alert but doesn't wait for

 # the close notify alert from client. When you need a different shutdown

 # approach you can use one of the following variables:

 # o ssl-unclean-shutdown:

 # This forces an unclean shutdown when the connection is closed, i.e. no

 # SSL close notify alert is send or allowed to received. This violates

 # the SSL/TLS standard but is needed for some brain-dead browsers. Use

 # this when you receive I/O errors because of the standard approach where

 # mod\_ssl sends the close notify alert.

 # o ssl-accurate-shutdown:

 # This forces an accurate shutdown when the connection is closed, i.e. a

 # SSL close notify alert is send and mod\_ssl waits for the close notify

 # alert of the client. This is 100% SSL/TLS standard compliant, but in

 # practice often causes hanging connections with brain-dead browsers. Use

 # this only for browsers where you know that their SSL implementation

 # works correctly.

 # Notice: Most problems of broken clients are also related to the HTTP

 # keep-alive facility, so you usually additionally want to disable

 # keep-alive for those clients, too. Use variable "nokeepalive" for this.

 # Similarly, one has to force some clients to use HTTP/1.0 to workaround

 # their broken HTTP/1.1 implementation. Use variables "downgrade-1.0" and

 # "force-response-1.0" for this.

 BrowserMatch "MSIE [2-6]" \

 nokeepalive ssl-unclean-shutdown \

 downgrade-1.0 force-response-1.0

 # MSIE 7 and newer should be able to use keepalive

 BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

</VirtualHost>

</IfModule>

----------------------------------------------------------------

10° Passo:

O HTTPS escuta sobre a porta de número 443. É necessário adicionar a linha NameVirtualHost \*:443 ao /etc/apache2/ports.conf

Arquivo ports.conf (save)

----------------------------------------------------------------------

# If you just change the port or add more ports here, you will likely also

# have to change the VirtualHost statement in

# /etc/apache2/sites-enabled/000-default

# This is also true if you have upgraded from before 2.2.9-3 (i.e. from

# Debian etch). See /usr/share/doc/apache2.2-common/NEWS.Debian.gz and

# README.Debian.gz

# Apache listen in port 80

NameVirtualHost \*:80

Listen 80

# Apache-SSL listen in port 443

NameVirtualHost \*:443

<IfModule mod\_ssl.c>

 # If you add NameVirtualHost \*:443 here, you will also have to change

 # the VirtualHost statement in /etc/apache2/sites-available/default-ssl

 # to <VirtualHost \*:443>

 # Server Name Indication for SSL named virtual hosts is currently not

 # supported by MSIE on Windows XP.

 Listen 443

</IfModule>

<IfModule mod\_gnutls.c>

 Listen 443

</IfModule>

----------------------------------------------------------------

Outro arquivo que precisa ser acrescentado é apache2.conf

No diretório # cd /etc/apache2/apache2.conf
No arquivo apache2.conf devem ser acrescentadas as linhas:

# ServerName

ServerName localhost

-------------------------------------------------------------------------------------------------------------------------------

11° Passo:

Depois de instalado o certificado será feito o reinício do Apache2.

Para estabelecer as configurações acrescentadas nos arquivos do Apache:

# /etc/init.d/apache2 reload ou #service apache2 reload

Para ativar a execução do Apache:

# /etc/init.d/apache2 restart ou #service apache2 restart

Se quiser ver o status do serviço em dado momento:

# service apache2 status (deve acusar “Apache is running.”)

Se o usuário optou pela instalação com o pedido da chave criada no início do exemplo, a chave será pedida toda vez que houver reiniciar o serviço, mas se o usuário efetuou os passos para a autenticação automática, o Apache2 não pedirá a chave.

12° Passo:

Feitos todos os passos do exemplo, será feito um teste prático, o usuário deve criar uma pasta chamada “teste” dentro do “/var/www/”, e criar pastas para testar HTTP e HTTPS.

Dentro do arquivo o usuário deve alterar as linhas que possuam “/var/www/” para “/var/www/teste/” (só serão duas linhas), assim estará sendo indicando o caminho para a pasta da página inicial.

Os comandos para criar é:

# mkdir /var/www/teste/html/ (para o exemplo, deve conter o arquivo index.html e um diretório logs que contém o arquivo error.log apropriado para os logs de erros.

# mkdir /var/www/teste/ssl/ (para o exemplo, deve conter o arquivo index.html e um diretório logs que contém o arquivo error.log apropriado para os logs de erros.

Criar diretórios aqui com mkdir <nome\_diretorio>

Criar arquivos vazios com gedit

Dentro da pasta “/var/www/teste/html e /var/www/teste/ssl ” deve-se criar o arquivo “index.html” e no diretório ssl criar o arquivo “index.html” Para criar o arquivo dentro da pasta teste o usuário deve digitar:

# gedit index.html

Inserir os códigos abaixo, depois salvar e sair:

<HTML>

<HEAD> <TITLE> Teste do Apache2 sem e com SSL (HTTP-HTTPS) </TITLE></HEAD>

<BODY> Teste efetuado com sucesso...</BODY>

</HTML>

Em seguida, no browser, para testar os endereços será o seguinte:

<http://localhost> (testa HTTP) Apache sem segurança

<https://localhost/teste/> (testa HTTPS) Apache com segurança SSL.

Será pedido para aceitar o certificado, pois o certificado (auto-assinado) criado no exemplo não será reconhecido automaticamente pelo browser como seguro. Para haver esse reconhecimento o criador da chave deve contratar uma autoridade certificadora para gerar o certificado a partir da chave gerada. Essa autoridade de certificação é uma empresa autorizada a emitir, renovar e cancelar certificados digitais, após verificar a identidade e a legitimidade da parte solicitante através de uma autoridade de registro.

**CONCLUSÃO**

Postar no Moodle este arquivo, em nome da dupla. A sua presença na aula e sua nota da tarefa será dada no Moodle, após a realização na aula.