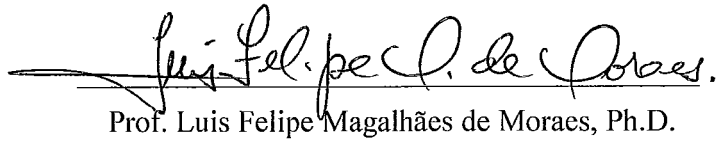
PROPOSTA E IMPLEMENTAÇÃO DE UM MODO SEGURO PARA HTTP, COM NÍVEL SELETIVO DE SEGURANÇA, SEM ALTERAÇÕES EM SERVIDORES E NAVEGADORES

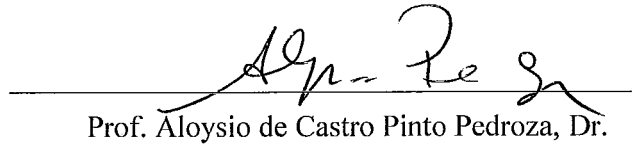
Marcelo Costa Pinto e Santos

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

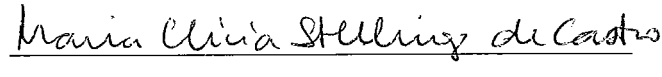
Aprovada por:



Prof. Luis Felipe Magalhães de Moraes, Ph.D.



Prof. Aloysio de Castro Pinto Pedroza, Dr.



Profª. Maria Clícia Stelling de Castro, D.Sc

RIO DE JANEIRO, RJ – BRASIL

OUTUBRO DE 2001

SANTOS, MARCELO COSTA PINTO E

Proposta e Implementação de um Modo Seguro para HTTP, com Nível Seletivo de Segurança, sem Alterações em Servidores e Navegadores [Rio de Janeiro] 2001

VIII, 128 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2001)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Segurança em redes de computadores
2. Criptografia

I.COPPE/UFRJ II.Título (série)

Agradecimentos

Alguém disse certa vez que, quando você realmente deseja alguma coisa, o universo conspira para que você a consiga. Eu acredito nisto e, no meu caso, o universo foi representado pela minha família, meus amados esposa e filhos, que por mais de uma vez, tiveram de abrir mão de minha companhia e mesmo de momentos de lazer no decorrer destes últimos dois anos. De meus pais que por tantas vezes tiveram de escutar, mesmo sem entender totalmente, eu temo, meus assuntos sobre COPPE, mestrado, segurança, etc... Meu orientador e meus colegas do laboratório Ravel, que tanto me ajudaram com o projeto e nas dificuldades na implementação em linguagem C, que eu não utilizava há mais de 10 anos. Aos meus a colegas professores do Colégio Técnico Universitário da Universidade Federal de Juiz de Fora (MG), que certamente tiveram suas obrigações ampliadas em função de meu afastamento para o mestrado. E, finalmente, mas não menos importante, a CAPES/UFJF/MEC, que proveram os recursos financeiros indispensáveis para a consecução de meus objetivos, na forma da bolsa do PICDT de que usufrui durante meu curso. Obrigado a todos vocês.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PROPOSTA E IMPLEMENTAÇÃO DE UM MODO SEGURO PARA HTTP, COM NÍVEL SELETIVO DE SEGURANÇA, SEM ALTERAÇÕES EM SERVIDORES E NAVEGADORES

Marcelo Costa Pinto e Santos

Outubro / 2001

Orientador: Luís Felipe Magalhães de Moraes

Programa: Engenharia de Sistemas e Computação

Apresentamos um processo alternativo para implementação de segurança para a WWW, utilizando os softwares tradicionais da camada aplicação, tanto servidores quanto clientes, de forma independente destes, que chamamos *WEBSEC*. O processo prevê a existência de um servidor que se comunica com o servidor HTTP tradicional, criptografa os dados e encaminha-os à um *plugin* que decifra as informações e as exibe na janela do navegador.

É previsto também o envio de informações no sentido inverso, quando o *plugin* criptografa dados preenchidos em um formulário e encaminha ao servidor WebSec que, por sua vez, decifra os dados e os passa ao servidor HTTP tradicional.

Desta forma podemos alterar as características da criptografia arbitrariamente (tamanho da chave, algoritmo,...), de forma independente de qualquer fornecedor de softwares e, ainda assim, continuar utilizando softwares amplamente conhecidos como o Internet Explorer, Netscape Navigator ou o Apache (servidor HTTP).

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PROPOSAL AND IMPLEMENTATION OF A SECURE WAY TO HTTP, WITH SELECTIVE SECURITY LEVEL, WITHOUT CHANGE IN SERVERS AND BROWSERS

Marcelo Costa Pinto e Santos

October / 2001

Advisor: Luís Felipe Magalhães de Moraes

Department: Systems Engeneering and Computer Science

We show an alternative process to implement security at the WWW, using traditional softwares, for clients and servers, in a way independent from them. We will call the process WEBSEC. The process creates a server that communicates with the traditional HTTP server, cryptograph the data and send to a plugin who decryptograph the information and show it in the navigator window.

We can send information in the bakway. The plugin cryptograph the data entered in a form and send them to the WebSec server, who decryptograph the data and pass to the traditional WebServer.

So, we can change the whole characteristics of the cryptography (length of the key, algorithim,...), in a way independent of any software suplier and still use well known softwares like Internet Explorer, Netscape Navigator or the Apache web server.

Índice

1 Introdução

O *Yankee Group,* instituto de pesquisas norte americano especializado em telecomunicações estima que, ao final do ano de 2001, existirão 14,1 milhões de brasileiros utilizando a Internet. Ao final de 2006 o mesmo instituto prevê o crescimento do número de usuários para 40,3 milhões. Portanto, a internet é uma realidade, e as perspectivas de crescimento sugerem que, cada vez mais, a "grande rede" entrará em nossas casas tornando-se, a exemplo da televisão e do telefone, um recurso rotineiro ao cidadão comum.

Ainda referênciando dados do *Yankee Group*, um estudo de Taylor Nelson Sofres nos informa que 15% dos usuários entrevistados em 36 países diferentes, fizeram compras no comércio tradicional impulsionados por informações encontradas na rede. O que faz uma pessoa preferir sair da comodidade de sua residência, onde poderia adquirir um produto sem levantar-se da cadeira, e ir até um estabelecimento comercial tradicional para realizar a tarefa? Segundo o mesmo estudo, cerca de 40% das pessoas que dizem não fazer compras pela Internet, alegam receio de divulgação indevida de seus dados pessoais, ou seja, não o fazem por acreditarem que a segurança dos dados enviados é insuficiente.

Com o crescimento das redes de banda larga, a miríade de serviços que podem ser oferecidos pela rede se multiplicará, provavelmente passando a incluir, nos próximos anos, TV por demanda, onde o telespectador não será acorrentado às programações das emissoras, podendo escolher o que ver a qualquer hora do dia, cursos onde alunos e professores se reunirão virtualmente pela INTERNET proliferarão, reduzindo a necessidade de migração interna para formação acadêmica, "telemedicina" poderá ser praticada, possibilitando que especialistas possam dar suporte a postos médicos de difícil acesso, além de inúmeras outras aplicações, que certamente surgirão, e não conseguimos nem sequer imaginar. Todas estas possibilidades, demandarão um meio de comunicação com níveis de segurança superiores aos que dispomos atualmente.

Dentre as aplicações que atualmente se proliferam na rede, certamente a mais popular é a malha de páginas de hipertexto, regida pelo protocolo *Hyper Text Tranfer* Protocol - HTTP. Mecanismos de segurança como o *Secure Socket Layer* - SSL e o

Secure http - SHTTP, abordados mais detalhadamente posteriormente neste trabalho, são cada vez mais utilizados mas, graças a grande concentração atual do mercado de software nas mãos de poucas empresas, todas norte americanas, ficamos totalmente dependentes destas corporações quanto a métodos de cifragem de dados utilizados, tamanhos de chave, e todos os demais parâmetros que influenciam no nível de segurança da comunicação.

Considere também que, em caso de conflito internacional, um dos campos de batalha será o mundo virtual, onde exércitos de especialistas em computação cumprirão missões de espionagem e desinformação, sem sair de seus quartéis, através de seus computadores. Existem precedentes, não confirmados, porém prováveis, de que órgãos governamentais norte americanos tenham solicitado a empresas de computação daquele país que deixassem formas secretas de burlar algoritmos de criptografia, como uma *porta dos fundos,* para facilitar a espionagem. Quem nos garante que a mesma prática não foi utilizada atualmente, de forma ainda mais velada?

Desenvolver um conjunto completo de aplicativos para a camada aplicação é uma forma de vencermos as dificuldades descritas. Esta não é a abordagem que propomos. Nossa idéia é o desenvolvimento de um "anexo" a estes softwares, tanto do lado do cliente, na forma de um *plugin*, como do lado do servidor, que implemente segurança de forma totalmente independente dos primeiros. Desta forma, teremos total domínio sobre o código e os parâmetros da criptografia utilizada, sem termos de desenvolver um conjunto de programas completamente novo que, além dos custos óbvios de desenvolvimento e manutenção, envolve gastos com treinamento de usuários que pode ser minimizado caso utilizemos os já existentes no mercado, amplamente conhecidos e utilizados em todo o mundo.

Os desempenhos obtidos com os protótipos desenvolvidos foram plenamente satisfatórios se comparados com as tecnologias atualmente em utilização pela comunidade mostrando, como discutiremos aportunamente, até mesmo vantagens sobre algumas plataformas.

O trabalho pressupõe que o leitor possua um conhecimento básico em redes de computadores, funcionamento de redes TCP/IP e arquitetura geral da Internet, em nível de curso de graduação, sendo adequado aos leitores interessados em iniciar estudos na área de segurança em redes, principalmente no capítulo 2, que traça um perfil dos recursos de segurança atualmente em utilização na internet, e de alguns ataques

conhecidos no meio *hacker*. O capítulo 2 constrói um vocabulário utilizado no decorrer do trabalho, e pode ser saltado, caso o leitor possua conhecimentos prévios na área de segurança lógica de dados, ou utilizado separadamente do restante do texto, como uma introdução ao estudo do assunto.

O capítulo 3 traz uma visão geral da idéia apresentada e algumas motivações para o desenvolvimento deste trabalho. No capítulo 4 apresentamos uma visão detalhada do projeto, discutindo-se as principais decisões de plataformas a utilizar, algoritmos e arquiteturas.

Um protótipo do sistema foi implementado, na forma descrita no capítulo 5. Alguns testes foram feitos, nas duas plataformas cliente mais utilizadas atualmente, e os resultados encontram-se descritos no capítulo 6, juntamente com algumas conclusões e sugestões para trabalhos futuros.

2 Tópicos Envolvendo Segurança em Sistemas de Informação

Ao final dos anos 90 vivemos uma situação inusitada. Técnicos apregoavam catástrofes em decorrência de uma falha de projeto em grande parte dos softwares em uso no mundo. Era o "*bug*" do milênio ou Y2K [9] . Previam-se usinas nucleares explodindo, aviões colidindo em aeroportos, falhas no fornecimento de serviços básicos, controle de tráfego, etc. Expurgados os exageros, este episódio serviu para demonstrar-nos quanto a sociedade da informação é dependente da integridade de seus sistemas de informação.

Se mudarmos o panorama e pensarmos que no lugar do "*bug*" do milênio encontra-se um terrorista, ou um oficial de um exército inimigo, chegamos a situação descrita em [23], onde os cenários de guerra passam ao mundo virtual, ficando no mundo real somente seus efeitos destruidores.

Dramaticidade à parte, segurança nas redes de computadores são o maior entrave ao desenvolvimento do comércio eletrônico e o maior obstáculo para que a sociedade atual possa usufruir de todas as facilidades e ganhos de produtividade prometidos pela tecnologia da informação.

Podemos dividir a segurança às redes de computadores em duas categorias principais: **Segurança Física** e **Segurança Lógica,** descritas a seguir.

2.1 Segurança Física X Segurança Lógica

A segurança física envolve controle do acesso ao hardware cuja proteção é desejada. Apesar de não ser o objeto principal de nosso estudo, sua importância não deve ser subestimada. Segundo [4] 73% dos ataques a corporações são provocados pelos funcionários ou ex-funcionários, ou seja, pessoas que têm acesso físico aos recursos do sistema.

Medidas para o aumento da segurança física envolvem desde o controle de acesso aos ambientes onde o processamento ocorre ou os dados são armazenados, até o monitoramento das irradiações eletromagnéticas emitidas pelos equipamentos por meio das quais pode-se ter acesso a dados secretos sem a devida autorização. Por exemplo,

existem instalações militares americanas onde todo o cabeamento de rede passa por um tubo cuja pressão do ar é mantida artificialmente superior à do meio ambiente de forma que, monitorando-se esta pressão, pode-se detectar violações na tubulação que configurem quebra na segurança.

Apesar de reconhecermos a importância do assunto, este trabalho tem seu foco na segurança lógica, onde invasores tentam, de máquinas remotas ao lugar invadido, acessar recursos computacionais sem a devida autorização de seus donos, utilizando-se para isto de redes de telecomunicação como a INTERNET.

2.2 Política de Segurança

A segurança tem sido freqüentemente negligenciada por gerentes de redes por várias razões, dentre as quais ressaltamos:

- A necessidade de controlarmos o acesso aos recursos computacionais disponíveis em uma rede, vai de encontro à própria origem das redes, cujo objetivo principal foi, e ainda é, o de disponibilizar facilidades aos usuários da forma mais ampla possível, sem restrições quanto à localização física dos recursos;
- Rotinas de identificação de usuários e controles de acesso consomem recursos da rede diminuindo sua eficiência ou, sob outro ponto de vista, exigindo maiores capacidades de processamento para que uma tarefa seja efetuada com a mesma eficiência;
- Recursos de segurança normalmente exigem que os usuários decorem senhas e aprendam a operá-los, indo de encontro à tão desejada facilidade de uso, grande responsável pela popularização do uso dos computadores nos últimos anos.

No entanto, muitas empresas têm descoberto do modo "mais difícil", que investimentos em segurança são necessários e, muitas vezes, essenciais ao funcionamento das organizações, principalmente as que utilizam a internet em seus negócios.

Faz-se necessária, portanto, a definição de uma **política de segurança** que determine quão valiosa é uma determinada informação para a organização, derivando daí as restrições ao seu acesso.

Entre estas restrições destacamos a determinação de **quem** pode ler, alterar, copiar determinado dado e quais destes eventos devem ser registrados em arquivos de

log para fins de auditoria posterior. A palavra "quem" esconde, além das diversas categorias de funcionários da organização, clientes e fornecedores que muitas vezes necessitam acesso privilegiado, público em geral que pode, dependendo do ramo de atuação da empresas, conter clientes em potencial, etc.

> *"Determinar uma política de segurança pode ser complexo porque uma política racional requer que a organização acesse o valor da informação. A política precisa ser aplicada a informações que encontrasse armazenada em computadores assim como a que trafega pela rede"*
>
> Comer [5]

Uma política de segurança deve envolver a organização como um todo, determinando, de forma clara e sem ambigüidades, responsabilidades dos componentes da empresa no que diz respeito à segurança dos dados. Deve ser de amplo conhecimento dos envolvidos considerando-se inclusive a possibilidade de incluir tais itens como cláusulas contratuais.

2.3 Segurança na Internet

O assunto segurança se complica quando a rede a proteger está conectada à Internet. O universo de possíveis agressores aumenta algumas ordens de grandeza, e sua segurança passa a depender da segurança das demais redes interconectadas.

Uma conexão à Internet se faz devido à necessidade de troca de dados entre redes, portanto, mesmo que sua rede conte com os melhores e mais atualizados recursos de segurança, você provavelmente terá de manter alguma relação de confiança com outras máquinas, em outras redes. Se esta outra rede for invadida, a sua própria estará vulnerável.

A maioria dos invasores deseja, por razões obvias, resguardar suas identidades e, para tal, invade sub-repticiamente alguns sites, mantendo a invasão fora do conhecimento dos administradores da rede, de forma a terem uma "base" para outros ataques. Quando um ataque chega a causar danos e vem a público, descobre-se que partiu de sites de universidades ou outras organizações respeitáveis que tiveram suas redes invadidas e utilizadas como um "trampolim" para novas invasões.

Existem relatos em [1], de invasores que, após entrar em uma rede, melhoram seus recursos de segurança para que a vítima não seja novamente invadida por outro

hacker, mantendo exclusividade na usurpação dos recursos. É como se na sub-cultura *hacker,* o indivíduo tomasse posse do lugar no ciber-espaço e o defendesse de outros grileiros.

Portanto, segurança é um assunto coletivo quando tratamos da internet. A presunção de que sua rede não possui dados sigilosos e, por isto não será alvo de ataques é equivocada e, caso você se despreocupe da segurança, será uma ameaça à toda a comunidade internet.

A seguir abordamos alguns conceitos e definições pertinentes, e alguns ataques amplamente utilizados pelos *hackers.*

2.4 Definição de Ataque

É consenso na sociedade atual que uma pessoa que tente utilizar recursos computacionais sem a autorização dos donos legítimos é um atacante e, apesar das legislações de alguns países ainda não refletirem este sentimento, devem ser coibidos de alguma forma.

O que dizer de alguém que teste sua rede quanto a possíveis vulnerabilidades? Como se experimentasse a tranca da porta de sua casa e, caso ela não fosse boa o bastante ele voltaria mais tarde para assaltá-la. Obviamente tal atividade deve também ser considerada ofensiva e coibida por lei.

Muitas vezes não é tão óbvia a determinação de "quando" um indivíduo está buscando por vulnerabilidades em sua rede ou está simplesmente verificando se você provê um determinado serviço, de forma legítima e sem más intenções. Estes limites ainda estão por definir e, engenheiros e juristas têm de cooperar para estabelecer normas e padrões para a sociedade.

Deve-se cuidar para que não se confunda **ataque** com **invasão**. Invasão compreende a possibilidade de efetivamente utilizar-se de recursos computacionais sem a devida autorização de seus possuidores legais. Ataque normalmente é o meio pelo que se consegue a invasão mas, em alguns casos, ataques podem ser desferidos visando imobilizar um determinado equipamento, sem necessariamente invadí-lo. Por exemplo, temos os ataques de negação de serviço, oportunamente abordados na seção 2.6.3.

2.5 Conceitos de Segurança

Seguem alguns conceitos importantes sobre segurança, que serão utilizados no decorrer deste trabalho:

Autenticação: certificação de que a mensagem realmente foi enviada por quem diz que a envia;

Integridade: certificação de que a mensagem não foi alterada durante seu trajeto pela rede;

Privacidade: certificação de que o conteúdo da mensagem não tenha sido revelado durante seu trajeto pela rede;

Não repudiação: possibilidade do destinatário de uma mensagem provar que a recebeu de uma determinada origem, mesmo que este negue ter enviado a mensagem.

2.6 Alguns Ataques Conhecidos

2.6.1 Analisadores de Protocolo (*Sniffers*)

São programas para estações do trabalho ou hardwares específicos, que colocam a interface de rede em modo promíscuo, ou seja, lendo todos os pacotes que passam pela rede e não somente os endereçados a ela.

São provavelmente as ferramentas mais óbvias para invasão de privacidade e espionagem eletrônica. Funcionam como um "grampo" digital.

As versões que rodam em estações de trabalho podem ser detectadas pelo aumento na carga de trabalho da estação ou na diminuição do espaço livre em disco, decorrente dos grandes arquivos gerados pela espionagem. Analisadores de protocolo tentam minimizar estes arquivos monitorando somente algumas conexões, ou selecionando as informações a armazenar de alguma forma.

Analisadores de protocolo com hardware próprio podem ser detectados testando a rede por endereços Ethernet desconhecidos.

2.6.2 Programas de Varredura

Uma das principais ferramentas dos invasores são programas que automaticamente percorrem toda a rede, ou uma determinada faixa de endereços IP, testando máquinas remotas quanto a:

- Que sistema operacional executa;
- Que serviços disponibilizam;
- Exige-se a identificação do usuário para todos os serviços disponibilizados?

Disparado o programa de varredura, o invasor pode dedicar-se a outras tarefas e, depois de algumas horas de processamento, consultar um relatório organizado, com as vulnerabilidades descobertas. Os programas de varredura são ferramentas importantes para invasores, mas também são utilizados com melhores intenções, por gerentes de segurança interessados em descobrir e extirpar falhas de segurança em suas redes.

A maioria dos programas de varredura de distribuição livre é construída para o sistema operacional UNIX, em forma de código fonte e exigem que o usuário tenha acesso a um compilador de linguagem C e arquivos que implementam os protocolos de comunicação da internet (TCP/IP). Se não for desejado que programas de varredura sejam executados, deve-se cuidar a quem são dados tais acessos. Um dos programas de varredura mais conhecidos tem nome sugestivo: SATAN (*Security Administrator's Tool For Analysing Networks - http://www.fish.com/satan/*) exige que o usuário tenha acesso à senha do superusuário para executa-lo.

Digno de nota também é o NESSUS (http://www.nessus.com), um programa gratuito desenvolvido por um jovem francês que incorpora ataques muito recentes. A título de exemplo reproduzimos a seguir um trecho do relatório gerado pelo NESSUS.

Timide.main.org 21

É possível tirar do ar o servidor remoto de FTP...enviando uma senha muito longa... Um invasor pode executar um comando qualquer... em uma estação remota utilizando este método... Solução: contate seu revendedor para uma atualização.

2.6.3 Ataques de Negação de Serviço (DoS "Denial of Service")

Um ataque de negação de serviço torna inoperante um sistema, sem necessariamente invadí-lo. É possível que um servidor, ou toda uma rede seja colocada fora de operação, inundando-a com algum tipo de solicitação ou tráfego maliciosos que a incapacite temporariamente de atender a solicitações legítimas.

Com muita facilidade pode-se, por exemplo, enviar uma quantidade arbitrária de mensagens de correio eletrônico para alguém de a forma tornar inviável a comunicação via e-mail com determinada pessoa (*mail bonb*).

No início do ano 2000 sites muito conhecidos sofreram ataques deste tipo, realizados por meio de uma ferramenta denominada *Smurf*. O atacante invade diversas redes, preferivelmente com conexões rápidas com a Internet, e instala nestas máquinas, sem serem detectados, programas que ficam aguardando um comando remoto para disparar uma quantidade muito grande de pacotes ICMP (*Internet Control Messages Protocol*) para uma outra rede, esta sim, a vítima real do ataque de DoS. Quando uma quantidade suficientemente grande de máquinas invadidas já está executando tal programa (estes sites são chamados *Zumbis*), o atacante dispara o ataque, e todos os zumbis iniciam, simultaneamente, a enviar pacotes ICMP para a vítima. Tais pacotes, apesar de pequenos, são muito numerosos e, para muitos sistemas, por se tratarem de pacotes de controle da rede, têm prioridade sobre os demais, resultando em uma incapacidade total da rede atacada de responder a solicitações legítimas. Tal ataque também é referenciado como DDoS, o "D" extra significa *Distributed*, realçando o caráter distribuído do ataque.

Outros ataques deste tipo como *Stacheldraht, Trinoo* ou o *TFN (Tribe Flood Network)* são documentados pelo CERT (*Computer Emergency Response Team),* mantido pela Universidade de Carnegie Mellon, que visa disponibilizar uma fonte permanente e atualizada de consulta sobre os ataques e vulnerabilidades conhecidas.

2.6.4 Escondendo a origem dos pacotes ("Spoofing")

Muitos sistemas apoiam-se em relações de segurança entre determinadas estações ou redes. Ou seja, baseado no endereço IP ou no nome de uma máquina, um acesso pode ser obtido ou facilitado. Portanto, a "personificação" de um terceiro por um invasor pode permitir ou facilitar o seu acesso a um determinado recurso.

Quando um pacote for enviado com endereço de origem falso, a máquina irá responder também para o endereço falso e, como o roteamento na internet é baseado no endereço IP, a resposta da máquina à qual o pacote foi endereçado será roteada para o endereço de origem (falso). Se o impostor tiver meios de interceptar este pacote, não encontrará problemas na personificação, no entanto tal situação não é comum.

Normalmente, o invasor tem primeiramente de retirar de operação através de um ataque de DOS a estação que deseja personificar, de forma que o pacote de resposta da máquina atacada não chegue ao seu destino. Depois disto, o atacante tem de continuar a conversação, às cegas, ou seja, sem conhecer as respostas da máquina atacada. Dificulta esta prática, o fato do TCP numerar os pacotes de forma a poder controlar a ordem dos datagramas durante uma conexão. Se a máquina atacada envia o pacote 1025, por exemplo, ele espera receber o 1026. A inicialização desta contagem deveria ser aleatória e fica a cargo da máquina atacada. O "deveria ser" se deve ao fato de alguns sistemas utilizarem processos determinísticos para este fim, facilitando o ataque.

Portanto, iniciada uma conexão com um endereço falso, o impostor tem de dar continuidade à comunicação sem acesso às respostas da máquina atacada. Como um vôo às cegas.

2.6.5 Ataque do Homem do Meio

Falhas na identificação de interlocutores quando se estabelece uma conexão pode deixar permitir o ataque do Homem do Meio, também conhecido como *Brigada de incêndios*, referência a antigas equipes de combate ao fogo que passavam baldes de mão em mão, levando a água até onde se fazia necessária.

Suponha que Ana queira se comunicar com Bruno e Carlos intercepte o primeiro pacote da comunicação. Carlos pode abrir um canal com Bruno como se fosse Ana e, a partir deste momento, repassar todos as informações que recebe de Ana a Bruno e vice versa, funcionando como um intermediário indesejado que conhecerá todas as mensagens trocadas entre Ana e Bruno, enquanto estes acham que estão trocando mensagens diretamente entre si.

2.6.6 Ataque de Reenvio

Este tipo de ataque é viável, mesmo quando mensagens são trocadas de forma cifrada, e o atacante não pode ler as informações que contêm.

Suponha que você envie uma ordem ao seu banco para transferir R$50,00 de sua conta para a de seu vizinho e este consiga interceptar os pacotes que contêm esta mensagem, guardando uma cópia e deixando que a comunicação prossiga, para que ele receba a quantia que lhe é devida.

No dia seguinte, as cópias guardadas são reencaminhadas ao Banco, que novamente transfere R$50,00 indevidamente.

Para evitar este tipo de ataque, os pacotes de segurança incluem números de ordem que não podem ser repetidos nos pacotes, ou indicadores de tempo de validade da mensagem.

2.6.7 Ataque do Aniversário

Quantas pessoas são necessárias para termos mais de 50% de probabilidade de encontrar dois indivíduos que façam aniversário exatamente no mesmo dia?

Quem responde baseando-se apenas no bom senso normalmente erra. A resposta é 23. Com 23 pessoas, temos 23 x 22 x 0,5 = 253 pares diferentes que comparados aos 365 dias do ano nos levam a uma probabilidade maior do que 50%.

Portanto, descobrir a chave utilizada na criptografia de uma determinada mensagem não é uma tarefa fácil mas, por outro lado, descobrir duas mensagens que contenham a mesma assinatura pode se tornar viável mais facilmente do que imaginamos.

Suponha que você possa sugerir uma mensagem para que alguém assine eletronicamente, utilizando uma chave secreta. Escolha um par de mensagens que gere a mesma assinatura, consiga a assinatura para uma mensagem e a troque posteriormente pelo outra, com assinatura idêntica. É como se você recortasse a assinatura de uma mensagem autêntica e a colasse em uma falsa.

2.6.8 Cavalos de Tróia

Contanto que o atacante tenha algum acesso à máquina atacada, ele pode instalar programas que realizem funções desconhecida do usuário. Estes programas são chamados "Cavalos de Tróia".

Por exemplo, pode-se substituir o programa responsável pela leitura do nome do usuário e senha por uma versão maliciosa que, além de promover o *login* desejado, transmite para um terceiro o nome e a senha do usuário.

Suponha que você receba pelo correio, um daqueles cartões eletrônicos de natal na forma de arquivo executável, que executa uma animação. Nada pode garantí-lo que

além dos bons votos para as festas, o programa não envie, também via e-mail, o arquivo de senhas de sua máquina para um terceiro.

2.6.9 Quebra de Senha

A maioria dos sistemas operacionais atuais dificulta bastante a entrada não autorizada em um sistema por tentativas exaustivas de nomes de usuário e senha. Muitos limitam o número de tentativas de entrada sem sucesso, desativando a conta depois de um número razoável de erros. Outros possuem processos de identificação intencionalmente lentos, inviabilizando ataques deste tipo.

No entanto, caso um atacante consiga uma cópia de um arquivo cifrado, é viável que ele tente, em sua máquina local, ataques deste tipo. Tais ataques deixam de depender da rede e são passíveis de serem feitos, por exemplo, por diversas máquinas simultaneamente, o que aceleraria consideravelmente o processo.

Senhas com até seis caracteres são possíveis de quebrar com recursos computacionais relativamente modestos. Hoje se consideram seguras chaves com pelo menos 128 bits. No entanto, este número terá de ser aumentado, com a evolução da tecnologia dos computadores. Por exemplo: tornados operacionais os computadores quânticos, as velocidades de processamento devem subir algumas ordens de grandeza, tornando inseguras senhas com comprimentos maiores que 128 bits.

Devemos considerar também que estas senhas ou chaves têm de ser completamente aleatórias pois, a escolha de palavras de nossa língua (para a facilitar a memorização, por exemplo) torna viável o "*Ataque do Dicionário*" onde o atacante experimenta todas as palavras de um dicionário, reduzindo o universo para procura da senha a um tamanho viável. Mesmo estratégias como usar uma palavra escrita de traz para frente, ou unir duas palavras, ou incluir um número no início da palavra, não melhoram muito a situação. Lembre-se: Tudo que você imaginar neste sentido pode ser também pensado por um invasor.

2.6.10 Falhas em programas

Programas que funcionam como provedores de serviço de rede são críticos no que tange a segurança. Imagine que um programa não verifique os limites da pilha de instruções e que, logo em seguida a esta pilha, na memória, venha uma área de dados. Se um atacante incluir como dados uma rotina qualquer, de seu interesse (ou um desvio

para esta rotina), e conseguir forçar um estouro de pilha de forma a executar seu programa, ele pode realizar qualquer atividade computacional, com privilégios do servidor cuja falha tenha sido utilizada. Tal brecha de segurança é comumente referenciada como *buffer overflow*.

Os programadores, de uma forma geral, não têm uma preocupação explícita quanto à segurança ou, em muitos casos, desconhecem que um descuido simples deste tipo possa levar a uma quebra de segurança tão grave. Some a isto a pressão a que estão submetidos para liberação das novas versões aos usuários e pronto, falhas do gênero são comuns e somente descobertas depois que usuários sofrem algum ataque.

2.7 Criptografia

Desde o tempo do império romano o homem utiliza-se de códigos secretos para escrever mensagens de forma que somente quem conhecesse um determinado segredo pudesse lê-la. O "Código de César" substituía cada letra pela terceira letra que a seguia, considerando-se a ordem do alfabeto.

A criptografia é peça chave na segurança de redes e é utilizada na garantia de autenticidade, privacidade, integridade e não repudiação de mensagens eletrônicas.

Técnicas de criptografia vêm sendo tratadas como segredo militar e têm sofrido, por muitos países, restrições à exportação, semelhante às impostas a armas. Os Estados Unidos, principal produtor mundial de software, encontra-se nesta categoria, o que contribui negativamente para o desenvolvimento dos programas de segurança. Sistemas desenvolvidos nos Estados Unidos encontram dificuldades para exportação e um cidadão americano que contribua, mesmo que sem finalidade de lucro ou vínculo empregatício, com iniciativas de desenvolvimento de softwares de uso público, estará violando a lei de seu país.

Tais restrições são de eficiência duvidosa, já que os algoritmos, apesar de alguns possuírem patentes registradas, aparecem publicados em artigos que recebem ampla divulgação e, portanto, são de conhecimento público. De posse do algoritmo, sua implementação é um detalhe menor e não será obstáculo a sua utilização. Possivelmente os Estados Unidos reverão, em futuro próximo, esta política.

Podemos classificar os algoritmos para criptografia em SIMÉTRICOS e ASSIMÉTRICOS. Os primeiros utilizam a mesma chave para criptografar e decriptografar a mensagem e os ASSIMÉTRICOS utilizam-se de chaves diferentes, criptografando-se com uma decriptando com a outra.

2.7.1 Princípios Básicos da Criptografia

O código de César, citado anteriormente, que a princípio pode parecer bastante ingênuo, contêm uma operação básica utilizada nos principais algoritmos de criptografia, a **substituição** de caracteres. Podemos aprimorar o algoritmo de César, gerando uma tabela de substituições para cada um dos caracteres do alfabeto a ser transmitido.

caractere	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	x	z
Código	*u*	*o*	*i*	*a*	*v*	*p*	*j*	*b*	*x*	*q*	*k*	*c*	*z*	*r*	*l*	*d*	*s*	*m*	*e*	*t*	*n*	*g*	*f*	*h*

Tabela 2.1: Possível código de substituição de caracteres.

Por exemplo, considerando a Tabela 2.1, a mensagem: "o cavalo ganhador sera o negro" seria cifrada para "*l iugucl gvripalm evmu l rvjml*". Se dispensarmos os espaços que separam as palavras a quebra ficara mais difícil mas, ainda assim possível. Existem estudos estatísticos sobre a freqüência de cada caractere e de alguns grupos de caracteres. Desta forma, sabendo-se que o texto foi escrito em português, se o símbolo mais freqüente for o "*u*" provavelmente ele representará a letra "a" que é a letra mais freqüente no português. Conhecido o "a", se "*cbu*" for o conjunto de 3 letras terminado em "*u*" mais freqüente, provavelmente "*c*" e "*b*" representem "l" e "h", respectivamente, formando a sílaba "lha". Desta forma podemos inferir sucessivamente o significado dos símbolos, até decifrarmos totalmente a mensagem. Com um programa de computador bem escrito e uma quantidade razoável de texto criptografado, algumas horas de processamento devem bastar para quebrar qualquer código do gênero.

A fraqueza dos algoritmos de substituição está na preservação da ordem dos caracteres. Algoritmos que alteram a ordem dos caracteres são chamados algoritmos de **transposição**. Por exemplo "ocavaloganhadorseraonegro" poderia ser cifrado trocando a ordem dos caracteres conforme indica a primeira linha da Tabela 2.2:

3	5	4	1	2
o	c	a	v	a
l	o	g	a	n
h	a	d	o	r
s	e	r	a	o
n	e	g	r	o

Tabela 2.2: Exemplo de transposição

Desta forma o texto cifrado resultaria em "*vaoaranrooolhsnagdrgcoaee*". Qualquer substituição pode ser enquadrada no esquema ilustrado na Tabela 2.2 com as devidas variações no número de colunas e na ordem em que as consideramos. Se descobrirmos estes dois parâmetros o algoritmo está quebrado.

Para quebrar um código de substituição o analista deve, primeiramente, descobrir se a técnica utilizada foi realmente uma transposição simples, o que pode ser conseguido por uma análise da distribuição estatística dos caracteres. Se a distribuição dos caracteres for a mesma do idioma utilizado para escrita do texto original, então provavelmente trata-se de uma transposição simples.

O primeiro passo para quebrá-la é descobrir quantas colunas foram utilizadas para a transposição. Tentativas inteligentes podem ser conseguidas se conhecermos algum grupo de caracteres presente no texto. Se soubermos que a mensagem trata-se de uma "barbada" para um determinado páreo, é razoável pensarmos que as palavras "cavaloganhador" farão parte do texto. Neste caso, se o algoritmo utilizar 3 colunas, os grupos "*cagho*", "*alaar*" e "*vond*" deverão estar presentes. Se o algoritmo utilizar 4 colunas, os grupos "*clno*", "*aohr*", "*vga*" e "*aad*" farão parte do texto cifrado. Podemos proceder assim sucessivamente, até encontrar o número de colunas correto. Feito isto, a seqüência em que foram consideradas as colunas pode ser descoberta por "força bruta", ou seja, tentam-se todas as possíveis combinações.

Se transformarmos a mensagem em uma cadeia de bits, por exemplo tomando os valores dos códigos ASCII dos caracteres, e realizarmos um **ou exclusivo** ("*x-or*") com uma cadeia de bits randômica de mesmo comprimento (que passamos a chamar *chave*), o código gerado não mantém qualquer correlação com o original e, por isto, não fornece

qualquer pista a possíveis analistas que tentem quebrá-lo. No entanto, manter uma chave tão longa quanto toda a mensagem não é prático. Decorá-la fica fora de questão na maioria dos casos e a distribuição desta chave pode substituir o problema original ao invés de resolvê-lo.

A maioria dos algoritmos modernos utiliza-se de combinações dos métodos anteriormente descritos, de forma a conseguir a menor correlação possível entre o texto original e o texto cifrado, com uma chave de tamanho razoável. Na Figura 2.1 podemos visualizar a simplicidade da implementação das técnicas básicas em hardware, e como poderíamos montar um algoritmo de criptografia. A caixa P simplesmente altera a ordem dos bits, substituindo um caractere por outro, de forma que podemos facilmente retornar ao caractere original desfazendo as trocas de posição das informações binárias. As caixas S codificam grupos de 3 bits em oito, de forma a termos somente 1 bit ativo no grupo de oito. A seguir empregamos uma caixa P, de forma a trocar o bit ativo de posição e recodificamos os oito bits no tamanho original (3). Desta forma trocamos o caractere inicial de 3 bits por outro, sendo a operação também facilmente reversível.

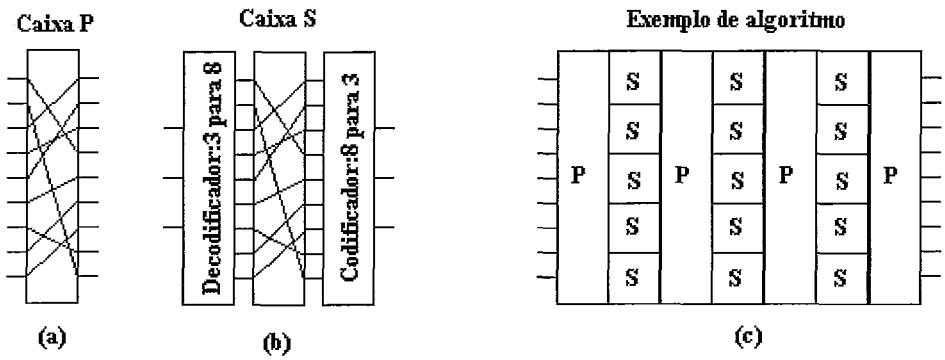


Figura 2.1: (a) Caixa P – Permutação; (b) Caixa S – Substituição; (c) Exemplo de Algoritmo utilizando caixas P e S.

2.7.2 Algoritmos Simétricos

Os algoritmos simétricos, são também conhecidos como algoritmos de chave secreta pois possuem apenas uma chave (Figura 2.2), que cifra e decifra os dados e, portanto, deve ser mantida em segredo.

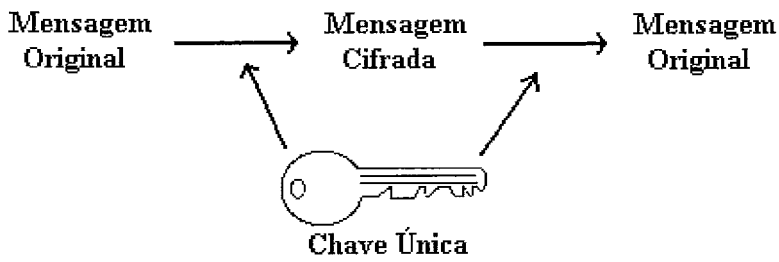


Figura 2.2: Criptografia Simétrica ou de Chave Secreta.

Podem ser expressos matematicamente como a seguir:

$$m = D_k(E_k (m))$$

m = mensagem original

D_k = Decifrar com a chave k

C_k = Cifrar com a chave k

A segurança dos algoritmos de criptografia simétricos baseia-se em dois aspectos [28]:

1- Não existe "atalho" que permita a um analista recuperar o texto original sem testar exaustivamente o espaço de possíveis chaves;

2- O espaço de chaves[1] é grande o sufuciente para inviabilizar o teste exaustivo citado no item anterior.

O problema da criptografia pode ser abordado probabilisticamente através da Teora da Informação, como um sistema envolvendo um canal com um nível de ruído muito alto. Na realidade com o maior ruído que o projetista do algoritmo pode conseguir. No entanto não existe uma forma definitiva de, dado um algoritmo de criptografia, determinarmos se ele tem ou não um "atalho" que possibilite a recuperação do texto original sem o teste exaustivo do espaço da chave. Portanto, algoritmos amplamente publicados e exaustivamente estudados por especialistas são considerados seguros pela comunidade.

[1] Espaço de chaves são todas as possíveis chaves de um algoritmo.

Quanto ao tamanho da chave, são consideradas seguras chaves com tamanho superior a 90 bits [28], atualmente sendo 128 o número de bits mais freqüentemente encontrado em sistemas de criptografia seguros.

A seguir descreveremos sucintamente os algoritmos simétricos mais utilizados na atualidade.

2.7.2.1 DES (Data Encription Standard)

O mais conhecido algoritmo de criptografia simétrico é o DES (*Data Encription Standart*), desenvolvido pela IBM em 1977. Foi adotado pelo governo americano como padrão para utilização com informações não classificadas.

O DES original encripta blocos de 64 bits. Para mensagens maiores que 64 bits, dividem-se as informações em blocos consecutivos daquele comprimento. Tal prática torna o algoritmo uma substituição monoalfabética, considerando-se um alfabeto com letras de 64 bits de comprimento. Todo bloco igual levará a textos cifrados iguais. Tal característica configura uma vulnerabilidade pois, caso o atacante possua acesso à distribuição probabilística dos caracteres deste alfabeto, e disponha de uma boa quantidade de texto criptografado, pode inferir, pela frequência relativa de cada caractere, toda a tabela de substituições e, assim, decifrar o texto.

A Figura 2.3 mostra o esquema de funcionamento do DES. 16 iterações tomam os 32 bits mais à direita e concatenam com o resultado de um "ou exclusivo" entre os 32 bits da esquerda, e o resultado de uma função cujos argumentos são os 32 bits da direita e uma chave de 56 bits. A função e a chave são alterados a cada iteração, aí residindo toda a complexidade do DES.

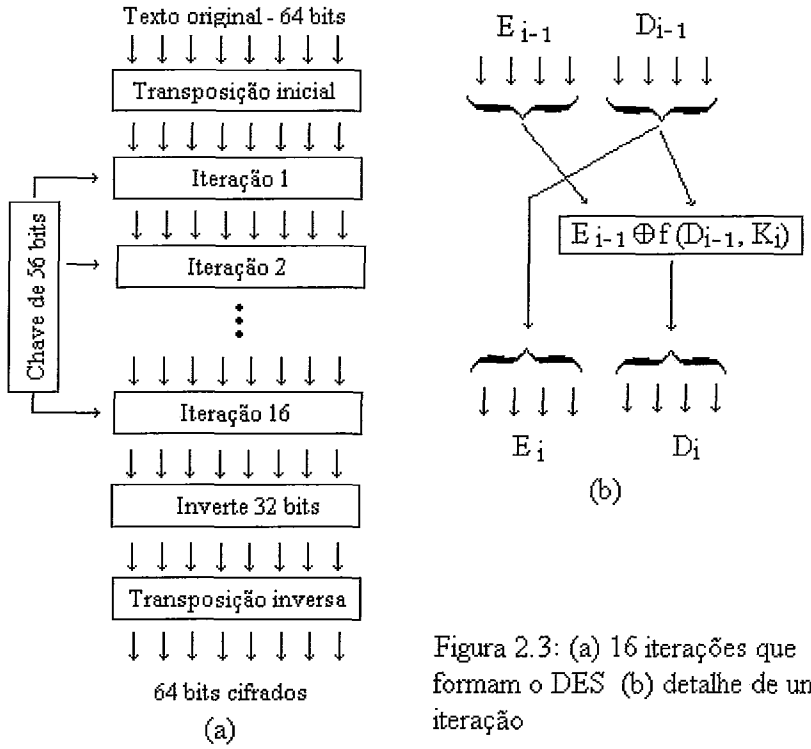


Figura 2.3: (a) 16 iterações que formam o DES (b) detalhe de uma iteração

Dadas as fragilidades do DES quando submetido à análises de freqüência, surgiram variações como o DES-CBC (CBC = *Cipher Block Chaining*), que utiliza a saída de cada bloco criptografado na encriptação do bloco seguinte, inviabilizando a quebra bloco a bloco, ou seja, com DES-CBC blocos idênticos de 64 bits levam a códigos diferentes, dependendo da posição na mensagem. Apenas o primeiro bloco não possui um anterior para ser usado como entrada do algoritmo. Por isto deve ser criado um "vetor de inicialização", para cifragem do primeiro bloco.

Na proposta inicial do DES, a IBM supunha chaves com 128 bits de comprimento mas, segundo [28], o comprimento da chave foi reduzido para 56 bits por pressão do governo americano que, supostamente, desejava um algoritmo seguro, mas não tão seguro que nem mesmo ele pudesse quebrar.

Ainda em 1977, Diffie e Hellman projetaram uma máquina de 10 milhões de dólares que quebraria o DES. Em 1994, Wiener apresentou uma máquina que faria o mesmo por 1 milhão de dólares. É consensual hoje que o DES não é seguro contra adversários bem aparelhados.

Surgiu, por isto, uma variação chamada de 3DES (*Triple DES*), que utiliza o DES por 3 vezes, com duas chaves diferentes. O 3DES é, atualmente, considerado seguro.

2.7.2.2 IDEA (International Data Encryption Algorithm)

Também tentando suprir as deficiências do DES, foi criado pelos suiços Lai e Massey, o IDEA (*International Data Encryption Algorithm*). Uma das vantagens atribuida ao IDEA é que, por ter sido desenvolvido na Suiça, estaria supostamente fora da área de influência do governo dos Estados Unidos. Como o DES, o IDEA trabalha com blocos de 64 bits, porém são utilizadas chaves de 128 bits, o que o torna imune a ataques de força bruta, pelo menos por enquanto.

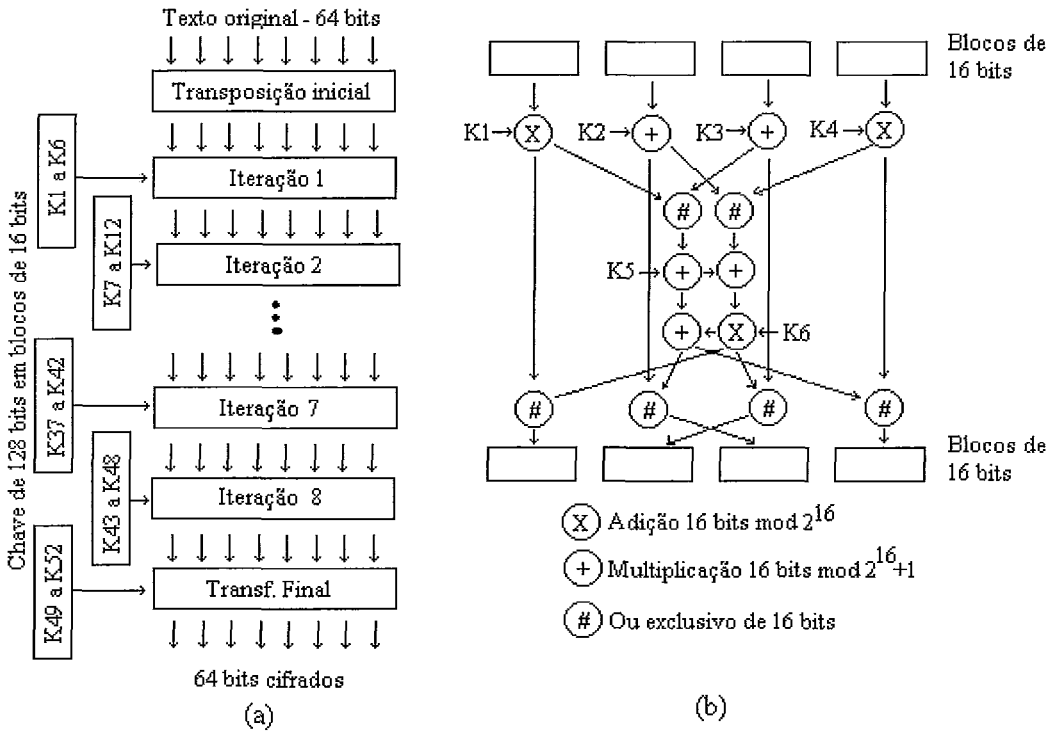


Figura 2.4: (a) IDEAl; (b) Detalhe de uma iteração.

Como podemos verificar na Figura 2.4, a chave de 128 bits é quebrada em 56 subchaves de 16 bits e cada iteração utiliza 6 subchaves, exceto a última, que utiliza apenas 4 bits. Todas as operações são facilmente implementáveis em hardware ou software, e as 8 iterações garantem que todos os bits da entrada influenciem na geração de todos os bits da saída, dificultando bastante sua quebra.

2.7.2.3 BLOWFISH

O BLOWFISH (Scheimer, 1994), utiliza chaves de tamanho variável, de 32 a 448 bits. Esta escalabilidade no tamanho das chaves é interessante, já que é difícil prever até quando 128 bits poderão ser considerados seguros. Conforme discutimos na seção 3.6, posteriormente neste trabalho, o BLOWFISH mostra desempenho muito superior a outros algoritmos amplamente utilizados.

O algoritmo gera, a partir da chave secreta, um vetor **P** de 18 elementos de 32 bits (**P1,P2,...,P18**) e 4 vetores (**S1, S2, S3** e **S4**) com 256 elementos também de 32 bits, e utiliza os valores em 16 rodadas conforme mostra a Figura 2.5.

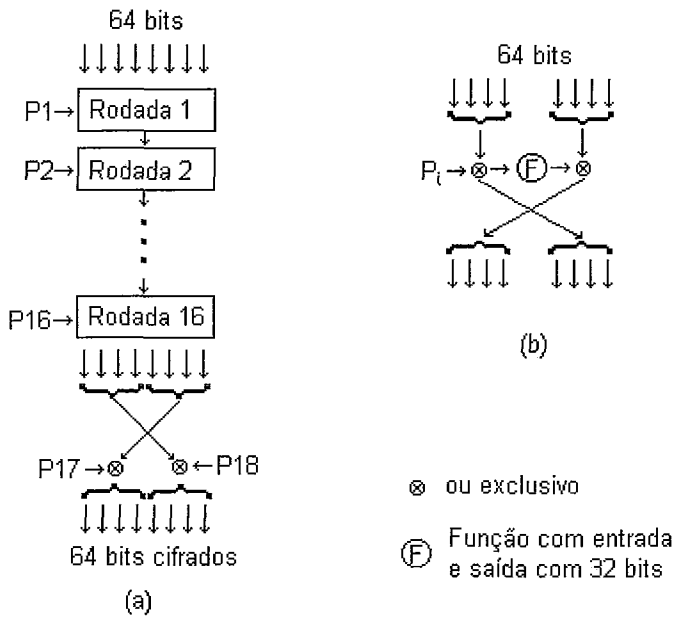


Figura 2.5: (a) Funcionamento do BlowFish; (b) Detalhe de uma rodada.

A função **F** da Figura 2.5 divide os 32 bits da entrada em 4 conjuntos de 8 bits chamados **a**, **b**, **c** e **d** que são utilizados como índices para os vetores S1, S2, S3 e S4, conforme a seguir:

```
F(x) = ( ( S1[a] + S2[b] ) XOR S3[c] ) + S4[d]

x = 32 bits de entrada;

S1...S4 = vetores (função da chave secreta);

a ... d = bits de "x" tomados 8 a oito
```

A geração dos vetores P e S segue o algoritmo abaixo:

1. Inicialize os vetores P e S com valores fixos (podem ser vistos no anexo A, fonte do WebSec);
2. Faça OUS EXCLUSIVOS entre P1 e os 32 primeiros bits da chave, P2 e os 32 bits seguintes, P3 e os 32 bits seguintes, e assim sucessivamente até completar os 18 elementos de P. Quando os bits da chave terminarem, volte ao início da mesma, quantas vezes forem necessárias;
3. Criptografe 64 bits zeros com o *Blowfish* e os valores de S e P iniciais;
4. Substitua P1 e P2 com o resultado obtido no item 3;
5. Cifre o resultado obtido no item 3 utilizando o *Blowfish* e os valores atuais de P e S;
6. Substitua P3 e P4 com o resultado obtido no item 5;
7. Continue o processo até que tenham sido substituidos todos os elementos de S e P.

O Blowfish concentra toda sua complexidade na geração de S e P, o que gera uma latência inicial maior que a maioria dos algoritmos similares; no entanto, este aumento de complexidade é amplamente compensado com um ganho muito grande de desempenho na criptografia dos dados propriamente dita. Como a geração das subchaves é feita apenas uma vez, o *Blowfish* torna-se especialmente indicado para criptografia de grande quantidade de dados.

O mesmo algoritmo descrito é utilizado para decifrar mensagens, exceto por uma inversão da ordem dos elementos da matriz P.

Este foi o método utilizado para construção dos protótipos do *WebSec,* de que trata este trabalho. A escolha está plenamente justificada nos capítulos que seguem.

2.7.3 Algoritmos Assimétricos

Nesta classe de algoritmos utilizamos duas chaves, uma **pública** e uma **privada.** Se utilizarmos a pública na cifragem, devemos utilizar a privada na decifragem e vice versa, se utilizarmos a privada na criptografia, é necessária a pública para o processo inverso. Como pode ser verificado na Figura 2.6, para enviar uma mensagem cifrada com esta classe de algoritmos devemos, conhecer a chave pública do destinatário, que é

utilizada para criptografar a mensagem. Desta forma, somente o conhecedor da chave privada correspondente poderá decifrá-la.

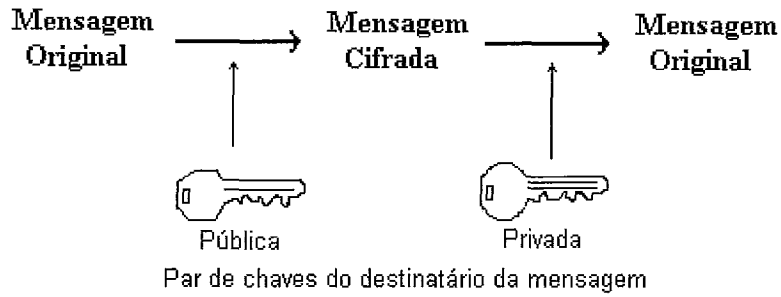


Figura 2.6:Troca de mensagens utilizando Criptografia Assimétrica.

Algoritmos de chave pública ou assimétricos podem facilitar grandemente a distribuição de chaves, já que uma das chaves (pública) pode ser amplamente divulgada, inclusive para possíveis invasores sendo que a outra (chave privada), não precisa ser divulgada nunca, nem mesmo ao seu interlocutor, o que facilita sua manutenção.

No entanto, o desempenho dos algoritmos assimétricos é muito inferior se comparado aos simétricos e o processo de escolha das chaves é mais complexo e dispendioso. As chaves dos algoritmos assimétricos devem ser números primos grandes pela própria natureza do processo e não se conhece processo eficiente para descoberta de tais primos. O que é feito é testar exaustivamente uma determinada faixa numérica à procura dos primos, o que pode ser computacionalmente custoso ([12] contem uma análise detalhada).

Por tal motivo, os algoritmos de chave pública são normalmente utilizados para uma troca inicial de mensagens, que contenham uma chave a ser compartilhada pelo par comunicante, em algoritmos de chave privada, bem mais eficientes.

Outra função interessante dos algoritmos de chave pública são as implementações de não repudiação, também chamada de "assinatura digital". Neste caso, o emissor da mensagem a cifra utilizando sua chave privada e o receptor ao decifrar a mensagem utilizando a chave pública do emissor, pode ter certeza de sua origem, pois somente o emissor conhece sua chave privada e, conseqüentemente, somente ele poderia ter gerado aquela mensagem. Desta forma o receptor pode, se necessário, provar que uma determinada mensagem foi enviada por um indivíduo, a menos que a chave privada seja descoberta, o que prejudica qualquer processo.

Se for desejada "não repudiação" e "privacidade", deve-se submeter a mensagem a uma dupla cifragem utilizando-se o algoritmo de chave pública. Primeiro com a chave privada do emissor e depois com a chave pública do destinatário. Desta forma somente o destinatário poderá decifrar a mensagem, pois será necessária sua chave privada no processo e, em seguida, decifra-se o resultado com a chave pública do emissor, que desta forma não poderá negar a autoria. A Figura 2.7 ilustra esta técnica.

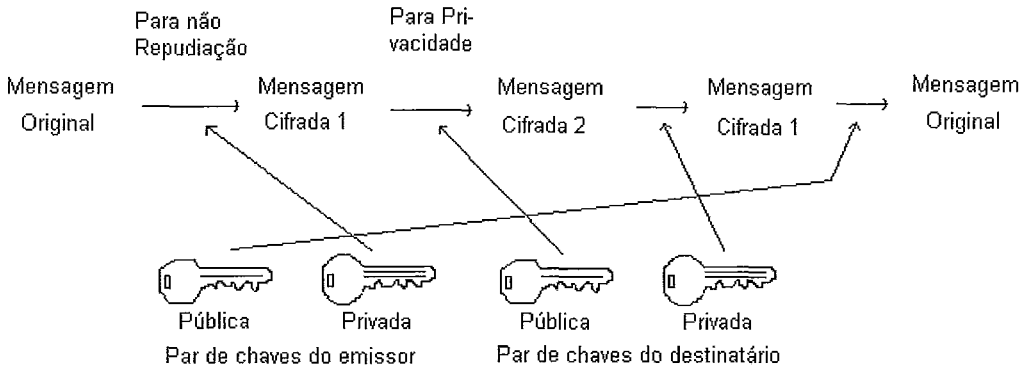


Figura 2.7: Utilização do Algoritmo Assimétrico para Não Repudiação e Privacidade.

Este algoritmo ainda é vulnerável ao ataque do "homem do meio" (abordado anteriormente), a menos que se obtenha um meio de descobrir, de forma segura, qual é a chave pública de um interlocutor. Esquemas de distribuição centralizada de senhas e Centrais de autenticação resolvem o problema, conforme estudaremos posteriormente.

2.7.3.1 Princípios Básicos para o Funcionamento dos Algoritmos Assimétricos

A idéia básica dos algoritmos assimétricos é a **aritmética modular**. As operações da aritmética modular são as mesmas da aritmética usual, exceto por se calcularem todos os resultados como o resto da divisão inteira (modulo ou simplesmente mod) por um determinado número chamado *base*.

Por exemplo, na aritmética modular base 10 existe a operação soma (+) definida como:

```
8 somamod10 9 = (8+9) mod 10 = 17 mod 10 = 7
```

Assim sendo, a seguir ilustramos na Tabela 2.3 a adição módulo 10.

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Tabela 2.3: Adição módulo 10.

Como podemos verificar pela Tabela 2.3, a adição módulo 10 pode ser usada como um algoritmo de criptografia (certamente não um bom algoritmo). Se somar-mos qualquer número a 4, por exemplo, ao subtrairmos 4 voltamos ao número original. Portanto a soma módulo 10 admite um inverso, quaisquer sejam os números somados.

Consideremos agora a Tabela 2.4 da múltiplicação módulo 10.

*	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Tabela 2.4: Multiplicação módulo 10.

A multiplicação por 5 certamente não pode ser utilizada como um algoritmo de criptografia. Ela resulta sempre em 0 ou 5, não sendo, portanto, reversível. O mesmo não acontece com a multiplicação por 3. Como podemos verificar na tabela, todos os números de 0 a 9 são levados a valores únicos quando multiplicados por 3. O mesmo ocorre com 1, 7 e 9. Portanto, a multiplicação por {1, 3, 7, 9} admite inverso multiplicativo. Por exemplo, o inverso multiplicativo de 3 é 7.

O *Algoritmo de Euclides* [12] nos mostra uma forma de encontrarmos inversos multiplicativos na aritmética modular. No entanto, quando os valores envolvidos são muito grandes, o algoritmo torna-se inviável computacionalmente, residindo aí um dos fatores de segurança dos algoritmos de chave pública.

Somente {1, 3, 7, 9} possuem inversos multiplicativos em nosso exemplo. Isto se deve ao fato destes números não possuirem fatores em comum com 10. Por isto podemos dizer, por exemplo, que 3 e 10 são **primos relativos**. Todos os números relativamente primos com a base utilizada possuem inversos multiplicativos e, portanto, podem ser utilizados como um algoritmo de criptografia, não necessariamente um "bom" algoritmo de criptografia mas, certamente, reversível.

Ao número de elementos do conjunto de números relativamente primos a uma base "n" chamamos $\phi(n)$.

Definição:

$\phi(n)$ = n° de elementos do conjunto dos números relativamente primos a n.

Se "n" for primo, todos os números menores que n serão relativamente primos a ele. Se "n" for um produto de dois primos, digamos p e q, então $\phi(n)$ será (p-1)(q-1).

$\phi(n) = (n-1)$; **se n é primo;**

$\phi(n) = (p-1)(q-1)$ **se n=pq; p,q são primos;**

2.7.3.2 O RSA

O RSA assim batizado em homenagem aos seus criadores (*River, Samir e Adleman*), é o principal algoritmo de chave pública existente. A chave e o bloco a criptografar não são de tamanho fixo, mas o bloco a criptografar tem necessariamente de ser menor do que a chave. Usualmente utilizam-se chaves de 512 bits.

Primeiramente, é necessário que se escolha a chave pública e sua correspondente chave secreta. Devem-se escolher 2 primos grandes, **p** e **q** (com mais que 256 bits cada). Chamamos **n** ao resultado da multiplicação de **p** por **q**. Os valores de **p** e **q** devem ser mantidos em segredo enquanto **n** será divulgado. Não se conhece atualmente forma eficiente de fatoração para números grandes, logo não é possível a inferência de **p** ou **q** a partir de **n**.

$$\mathbf{n = p \cdot q; \quad p,q \text{ são primos grandes.}}$$

Para gerar a chave pública, escolha um número "**e**" que seja relativamente primo a $\phi(n)$. Como conhecemos **p** e **q**, sabemos que $\phi(n) = (p-1)(q-1)$. A chave pública será **<e,n>**.

Para gerarmos a chave privada, devemos escolher um número **d** que seja o inverso multiplicativo de **e** módulo $\phi(n)$. **<d,n>** será nossa chave privada.

Para criptografar uma mensagem **m** devemos computar a mensagem cifrada $\mathbf{c = m^e \bmod n}$. Para decifrar devemos computar $\mathbf{m = c^d \bmod n}$. Assim,

$$\mathbf{\text{criptografar: } c = m^e \bmod n;}$$

$$\mathbf{\text{decriptografar: } m = c^d \bmod n;}$$

Onde, m = mensagem original;

c = mensagem cifrada.

2.7.4 Algoritmos para *Hash*

Algoritmos desta classe mapeiam, utilizando uma chave, mensagens em códigos não necessariamente únicos de forma que, executando o algoritmo com mesmas mensagem e chave seremos levados a um mesmo código mas, mesmo que se conheçam a chave e o código, é impossível conseguir-se a mensagem original.

Tais algoritmos são úteis para verificação de integridade, se usados como um *checksun* da mensagem, na garantia de autenticidade, quando utilizado como prova de conhecimento de um segredo (chave) por interlocutores; no entanto não podem ser utilizados para garantir privacidade.

Os mais utilizados são o MD5 (*Message Digest vrs. 5*) e o SHA (*Secure Hash Algorithm*).

2.8 Centrais de Distribuição de Senhas - CDS

Distribuição de senhas é um aspecto importante na segurança. Como poderíamos guardar senhas para todos os pontos com quem normalmente estabelecemos conexões? Os Centros de Distribuição de Senhas (CDS) se apresentam como uma solução interessante.

Todos os usuários mantêm senhas secretas com uma entidade central, da confiança de todos, o CDS. Quando um usuário A deseja uma conexão com o usuário B, contata inicialmente o CDS que gera uma senha e a transmite a A (cifrada com a senha que o CDS e A compartilham). O CDS poderia transmitir a senha gerada também a B, mas o que é realmente feito é a geração de um tíquete cifrado com a senha secreta mantida entre o CDS e B, contendo a identidade de A e a senha a ser utilizada. O fato de a mensagem ter sido cifrada com a senha mantida entre B e o CDS garante sua autenticidade. Neste ponto, A e B receberam de forma segura, uma senha para utilização com um algoritmo simétrico e podem utilizá-la para comunicação.

Quando a estação B recebe o tíquete de A, decifra-o com sua senha compartilhada com o CDS, recebendo, desta forma, a garantia do CDS que A não é um impostor e uma chave, para estabelecer uma conexão segura com A.

Esta solução, apesar de funcional, vai de encontro às próprias origens da INTERNET, que nasceu com o propósito de evitar a centralização em poucos pontos do controle da rede. Os CDS formam nós chave se implementados, sem os quais não é possível qualquer comunicação segura na rede. Tal concentração pode ainda gerar gargalos arriscados para muitos sistemas.

O protocolo de autenticação de *Needhan-Schroeder* baseia-se nas idéias citadas e este, por sua vez, inspira o *Kerberos* produto desenvolvido pelo MIT. Sobre este último aconselhamos firmemente a leitura da referência [3], que explica na forma de um diálogo, os problemas e preocupações no desenvolvimento do *Kerberos*.

A distribuição de senhas utilizando-se o *Domain Name Service* - DNS é cogitada, mas a necessidade de implementação de servidores DNS seguros praticamente torna preferível o englobar do DNS pelos CDS, pois os últimos se apresentam bem mais complexos que o DNS que, apesar da grande utilização, é pouco mais do que um grande banco de dados distribuído.

2.8.1 Centrais de Autenticação - CA

Utilizando-se chaves públicas o problema é mais simples. Pode-se conseguir uma conexão segura, mesmo na troca do primeiro pacote. Como os algoritmos deste tipo são mais complexos, podem ser utilizados para troca de uma chave secreta que passa a reger a troca de mensagens de forma mais eficiente. Persiste o problema da identificação dos usuários, onde reside a utilidade das CAs.

A CA emite um certificado digital, que atesta a identidade e a chave pública de um determinado indivíduo na rede e o criptografa com sua chave secreta. Quando alguém precisa provar sua identidade, apresenta o certificado digital que pode ser verificado com a chave pública do CA. Desde que as chaves secretas continuem secretas, a comunicação está segura.

Os certificados digitais são emitidos com validade determinada, e enquanto forem válidos podem ser reutilizados, sem a necessidade de se recorrer ao CA a cada nova conexão desejada. Isto resolve o problema de congestionamento gerado na utilização dos CDs. No entanto gera um novo. O que fazer se um certificado for gerado com validade de, digamos, 1 ano e a senha do dono do certificado for revelada? Prazos grandes de validade levam à necessidade de listas de Contra Ordem. Listas que contêm certificados não confiáveis ainda com validade. Tais listas devem ser consultadas antes de se aceitar qualquer conexão utilizando certificados.

Para termos certeza da chave pública do CA, ele apresenta, por sua vez, seu certificado, assinado por outro CA configurando, desta forma, uma árvore de certificações até um CA raiz, que certifica todos os demais. Para podermos confiar em um certificado devemos, portanto, verificar a árvore de CAs até encontrar um em que confiamos por outros motivos independentes do meio não confiável pelo qual desejamos estabelecer a comunicação segura. Por exemplo, tenhamos ido pessoalmente até o CA e trazido seu certificado em um disquete ou ele conste da relação de CA confiáveis que você instalou juntamente com seu software cliente.

2.9 IPSec / IPv6

Trazer segurança para a camada IP é a proposta interessante de [2]. Com apenas um esforço, todas as aplicações gozariam dos benefícios. Na verdade, o IPSec é proposto como parte obrigatória da nova versão do IP, o IPv6 e é passível de implementação opcional com o IPv4, a versão atual.

O IPSec é projetado para a versatilidade. São definidos dois cabeçalhos: o primeiro visando autenticação, chamado AH (*Authentiction Header*) e o segundo visando autenticação e privacidade chamado ESP (*Encapsulation Security Payload*).

No AH, é utilizada uma função de *hash* cujo resultado é transmitido como um *cheksum* que inviabiliza alterações na mensagem durante seu trajeto e provê certeza de que a mensagem foi originada pelo detentor da chave utilizada.

Com o ESP, um algoritmo de criptografia simétrica encripta toda a área de dados da mensagem ou opcionalmente, no modo de tunelamento, toda a mensagem (inclusive cabeçalho). Neste caso um novo cabeçalho deve ser criado, de forma a rotear o pacote até o seu destino. O modo tunelamento pode ser utilizado para o estabelecimento de conexões seguras através de *firewalls*. Neste caso, a mensagem é tunelada até o *firewall* que decifra o pacote e o coloca na rede interna, por ele protegida. Esta opção pode facilitar a implantação do IPSec, já que permite a instalação do recurso em apenas uma máquina (o *firewall*), que serviria a toda uma rede.

O IPSec prevê a utilização, como algoritmo padrão, do MD5 para o AH e do DES para o ESP. No entanto a utilização destes algoritmos não é obrigatória e uma grande quantidade de opções são previstas. Entre elas: 3-DES, RC5, IDEA, CAST e BLOWFISH.

É criada a figura das *Associações de Segurança,* onde são armazenados os principais parâmetros de uma conexão no que tange a segurança como: Algoritmo utilizado para ESP e/ou AH, vetores de inicialização (quando necessários), chaves, tempo de validade da associação.

O IPSec obriga a implementação da distribuição manual de senhas mas prevê tanto a utilização de Centrais de Distribuição de Senhas quanto as Autoridades Certificadoras, já abordadas neste trabalho.

Trazendo a segurança para a camada rede, o que parece ser uma tendência, várias soluções já existentes para camada aplicação deixam de ser necessárias. Entre elas destacamos: PEM, PGP e o protocolo X-400 para correio eletrônico, SSH (*Secure* Shel) para Terminal remoto, etc.

Outra tecnologia que tende a ser substituída pelo IPSec é o VPN (*Virtual Private Networks*), que estabelece túneis criptografados através de redes não confiáveis. Com o IPSec a criptografia é estabelecia entre estações, dispensando os túneis e estendendo a segurança também ao interior da rede.

2.10 FireWalls

Estes sistemas visam isolar uma rede local contra acesso não autorizado. Funcionam como porteiros eletrônicos, colocados como um *gateway* entre a rede local e a INTERNET, decidindo quais pacotes podem e quais não podem passar.

Existem dois tipos básicos de *firewall*: O Filtro de Pacotes rede e o *Proxy*[2]

Os Filtros de Pacotes atuam nas camadas rede e transporte decidindo com base nos cabeçalhos destes protocolos e em um conjunto de regras definidas pelo usuário, que pacote pode ser colocado na rede interna e que pacote deve ser filtrado. As regras baseiam-se normalmente nos endereços de origem e/ou destino do pacote, qual o protocolo da camada transporte, qual a porta para qual a conexão é desejada.

Os *firewalls* do tipo *proxy* realmente isolam uma rede do tráfego externo. Atuam na camada aplicação e não é permitido que nenhuma máquina da rede interna faça conexões externas diretamente. Todas as conexões devem ser feitas com o *proxy* que, por sua vez, se conecta à máquina externa desejada. Portanto, para as máquinas internas todas as informações vêm do *proxy,* e para as máquinas externas não existe uma rede e sim apenas uma máquina, o *proxy.* Atuando desta forma, o *firewall* pode estabelecer filtros muito mais complexos, baseados na informação vista da camada aplicação. Pode-se, por exemplo, permitir e-mails mas sem arquivos anexados ou mais ainda, permitir

---

[2] Servidores *Proxy* são sistemas que funcionam como intermediários entre clientes e servidores, solicitando dados em nome do cliente ao servidor e repassando as informações ao cliente quando recebidas do servidor. Normalmente estes sistemas guardam cópias locais das informações mais freqüentemente solicitadas, funcionando como um *buffer*.

arquivos anexados somente de determinados tipos como ASCII, filtrando arquivos DOC ou EXE, mais susceptíveis a vírus.

Os *Firewalls* consomem mais largura de faixa e quanto mais complexas forem as regras estabelecidas maior pode ser o comprometimento de velocidade. *Proxys* são especialmente problemáticos no item desempenho, pois concentram todas as conexões externas de uma rede. Além disto, não são 100% seguros. Invasores sempre podem atacar com endereços falsos e esconder características que sabem ser filtradas pelo *firewall*.

2.11 Controle de Ingresso

A [8] propõe que as tabelas de roteamento sejam verificadas na aceitação de um novo pacote. Caso o endereço de rede não seja condizente com a porta por onde o pacote está entrando, este deve ser rejeitado. Tal prática dificultaria a utilização de endereços de origem falso, facilitando a localização de invasores e tornando os *firewalls* mais seguros.

Existem alguns problemas em sua implementação, além dos óbvios, de perda de desempenho no roteamento e o de "convencer" toda a comunidade a implementá-lo. Uma das tendências mais fortes da atualidade, o IP Móvel terá sua implantação dificultada.

Uma estação móvel em um ambiente com filtro de ingresso teria de alterar seu IP a cada nova rede a que se conectasse. Supondo mobilidade do tipo da que se observa hoje com telefones celulares, onde as estações de trabalho teriam de trocar de base sem que as conexões fossem interrompidas estaríamos em um cenário em que endereços IP teriam de ser trocados durante uma conexão, sem interrompê-la. Isto é possível e proposto em [18] no entanto mais largura de banda é consumida.

Outro problema relacionado à mobilidade é a escassez atual de endereços IP. Cada rede teria de prever um número grande de endereços para seus possíveis hospedes. Este problema é totalmente resolvido com o IPv6 e seu aumento do espaço para endereçamento.

2.12 Identificação de Pessoas – Biometria

A identificação de programas e estações através de um segredo compartilhado ou da prova do conhecimento de uma chave privada é segura, pois estas máquinas não se importam em "lembrar" senhas longas e completamente aleatórias. Quando tratamos de pessoas estamos diante de uma situação diferente.

Guardar uma senha do tipo "T7~GBM1#%Y004-MA" está além do limite da praticidade para a grande maioria dos usuários. Quando os administradores de sistemas tentam impor esta necessidade, o que vemos são pequenos papéis fixados nos monitores e nas mesas de trabalho, contendo estes códigos. Uma grande falha na segurança.

A identificação de uma forma geral, segundo [12] pode basear-se em três aspectos: O que você sabe; O que você possui ou O que você é.

Senhas e chaves enquadram-se na primeira categoria. Um cartão eletrônico de um banco enquadra-se na segunda e a terceira envolve reconhecimento de padrões biológicos como impressão digital, formas da face, desenhos da íris ou da retina. Por exemplo, os bancos brasileiros normalmente baseiam a identificação de seus clientes em dois dos três aspectos citados. Eles exigem que você SAIBA uma senha secreta e que você POSSUA um cartão eletrônico. A identificação via biometria provavelmente é o mais promissor, apesar de ser inviável atualmente.

O uso de padrões biológicos para identificação de pessoas é atualmente amplamente pesquisado. Um grande número de fatores é cogitado. A Figura 2.8 mostra um gráfico comparativo entre estes fatores.

- Voz;
- Padrão de digitação (intervalo entre digitação das teclas);
- Formas da mão;
- Impressão digital;
- Assinatura;
- Formas da face;
- Configuração dos micro-vasos da retina (fundo do olho);
- Desenhos da Íris (parte colorida do olho);

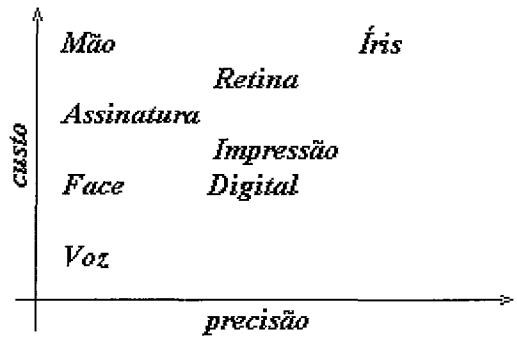


Figura 2.8: Relação Custo/Precisão observada por [17]

Ao contrário das senhas e chaves, todos estes sistemas devem aceitar margens de erro próprias ao fator que identificam e decorrentes de ruídos dos equipamentos utilizados na coleta dos dados para identificação. Por exemplo, a face de uma pessoa deve ser reconhecida, mesmo que ela não se barbeie todos os dias, e mesmo que a lente da câmera utilizada esteja ligeiramente empoeirada. Portanto, a identificação biométrica não leva necessariamente sempre ao mesmo valor digital, o que impede sua utilização direta como chave em algoritmos de criptografia. A identificação deve ser feita e a chave tem de ser consultada em um banco de dados, o que não é a situação ideal para segurança.

Biometria normalmente envolve o desenvolvimento de hardware específico e, dependendo do fator biométrico utilizado, o custo pode ser alto. Equipamentos para digitalização da íris envolvem dois *scanners*, um para encontrar a íris na face ou na região do olho e outro, de alta resolução, para digitalização da íris propriamente dita (um aparelho assim é descrito em [15]).

2.13 Ferramentas de "Log", Auditoria e Detecção de Invasões

Arquivos de "log" são arquivos criados e mantidos automaticamente pelos sistemas operacionais que registram as dados sobre as operações executadas pelos diversos usuários do sistema, a fim de possibilitar posterior auditoria caso necessário.

Uma das primeiras preocupações de um invasor é não deixar vestígios de sua presença e, por isto, sempre os primeiros alvos de alterações indevidas são os arquivos

de log. Desta forma os invasores tentam inviabilizar auditorias depois que uma invasão é identificada, escondendo sua identidade e os métodos utilizados na invasão.

Ferramentas de log alternativas, suplementares às fornecidas com os sistemas operacionais podem trazer grandes melhorias neste aspecto. A idéia é deixar a ferramenta tradicional, fornecida com o sistema operacional, que provavelmente deixará um *hacker* seguro quando corrompida, mas manter outra, ou outras, que seriam consultadas para verificação da integridade da primeira pois, muito provavelmente, passarão despercebidas, obviamente dependendo do nível de sofisticação do ataque.

Outras soluções são também interessantes, como a gravação em um meio *read only* como uma impressora, um disco óptico ou até mesmo em uma máquina dedicada, conectado à rede de forma que só seja possível o envio de informações da rede para a máquina, como se fosse uma impressora.

A detecção de uma invasão em andamento é outro problema a considerar. Normalmente trabalha-se no sentido de monitorar a rede por atividades suspeitas, típicas de invasores que, por sua vez, tentam burlar estes sistemas, tornando seus métodos o mais "normais" possíveis.

2.14 SHTTP – Secure Hyper Text Transfer Protocol

O *SHTTP* é definido pelo *Internet Engenearing Task Force* - IETF em [19]. Foi originalmente criado pela *CommerceNet*, uma coalizão de empresas interessadas no desenvolvimento da internet para propósitos comerciais. É um padrão bastante aberto, suportando a utilização de certificados digitais, criptografia simétrica e assimétrica, entre outros recursos já descritos neste documento.

Em cabeçalhos SHTTP pré-formatados são trocadas, entre outras, as informações abaixo.

- Forma de encapsulamento dos dados (PEM, PKCS-7,...);
- Formato de codificação dos certificados;
- Algoritmo para troca segura de chaves;
- Algoritmo para assinatura digital;
- Algoritmo simétrico utilizado para garantir a confidencialidade da troca de dados efetiva;

A identificação do cliente é opcional no SHTTP, como convém a maioria das aplicações na WWW, onde freqüentemente o cliente é totalmente desconhecido do servidor.

2.15 SSL – SecureSocket Layer

O SSL propõe um mecanismo semelhante ao do SHTTP. Porém, como uma camada intermediária entre a de *transporte* e a de *aplicação*, desde que a camada transporte assegure confiabilidade na entrega dos dados. O SSL provê privacidade e autenticação do servidor, para qualquer protocolo da camada aplicação e é o mecanismo de segurança mais utilizado atualmente na INTERNET. Assim como o SHTTP, é um protocolo extensível e versátil, prevendo várias formas de criptografia, autenticação e troca de chaves simétricas.

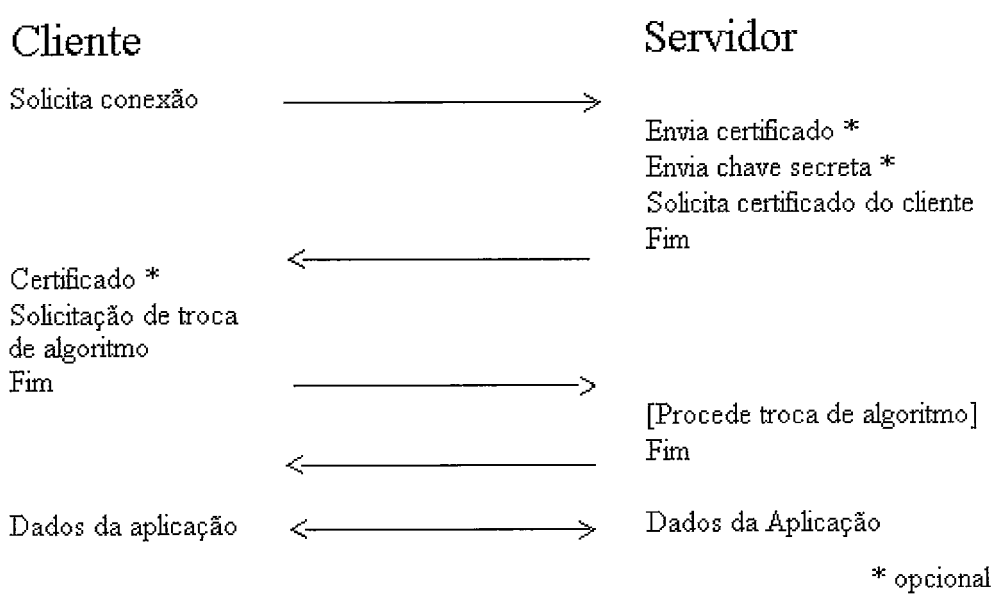


Figura 2.9: Troca inicial de mensagens com o SSL.

O SSL é de grande facilidade de uso pelo desenvolvedor WEB. Nas implementações mais comuns do SSL, basta que o URL seja representado como **https://www....** para que a seja estabelecida uma conexão com a porta 901 em lugar da tradicional porta 80 do HTTP. O servidor que implementa o SSL atende nesta porta e os

dados são transmitidos criptografados depois das negociações de chave e algoritmos descritos na Figura 2.9.

O WebSec pretende oferecer uma alternativa à esta arquitetura, conforme oportunamente será discutido no capítulo 3.

3 Projeto do Sistema WebSec

3.1 Introdução

Poucos são os governos e instituições que menosprezam o papel da comunicação em praticamente qualquer atividade em todos os tempos. A comunicação segura vem tendo interesse particularmente nos períodos de conflito entre nações, de forma que a criptografia acelera seu desenvolvimento a cada grande guerra por que passa a humanidade.

Em decorrência disto, algoritmos criptográficos vêm sendo tratados pela legislação de muitos países como arma ou munição. Nos EUA, por exemplo, até 1 de outubro de 1996, a exportação de softwares contendo recursos de criptografia era regida pelo *International Traffic in Arms Regulation – ITAR.* Somente após esta data o controle sobre o assunto foi transferido para o *Export Administration Regulations of the Department of Commerce* e teve alguma flexibilização mas, ainda assim, não era permitido a cidadãos americanos exportar software que implementasse criptografia cuja chave fosse superior a 56 bits, em outras palavras, os EUA desejavam manter a possibilidade de espionagem, restringindo as chaves a comprimentos passíveis de quebra.

Obviamente este procedimento é de validade questionável visto que os algoritmos criptográficos são de domínio público e facilmente implementáveis em qualquer parte do mundo. Tanto que em 23 de outubro de 2000 o governo dos EUA flexibilizou ainda mais a legislação, restringindo a exportação de software a apenas alguns países [13]. Tendo este contexto em mente, consideramos que a utilização de softwares de segurança produzidos em outros países fica prejudicada ou, em alguns casos até mesmo inviabilizada.

Além disto, como a maioria dos softwares utilizados não possuem código aberto, a insegurança quanto à existência de funções não documentadas que possibilitem ao desenvolvedor do produto decifrar informações sem o conhecimento do destinatário é justa. A história nos mostra que potências internacionais freqüentemente assumem posturas eticamente questionáveis quando seus interesses estão em questão.

Existe a possibilidade do desenvolvimento de uma plataforma completa onde certamente conseguiríamos vantagens de desempenho, no entanto, esta solução teria custo de desenvolvimento e implantação superiores, e possivelmente, no treinamento dos usuários na sua utilização.

Portanto, passamos a descrever nossa proposta de solução.

3.2 Sistema Proposto

Nosso interesse neste trabalho é especificamente voltado para a WWW. Propomos o desenvolvimento de software que implemente criptografia de dados entre um servidor e um cliente HTTP que seja compatível com as versões comerciais destes softwares e opere sem necessidade de alteração nem no cliente nem no servidor. Em outras palavras, desejamos colocar um software entre o cliente e o servidor web que implemente segurança, sem necessidade de alteração em nenhum dos dois softwares.

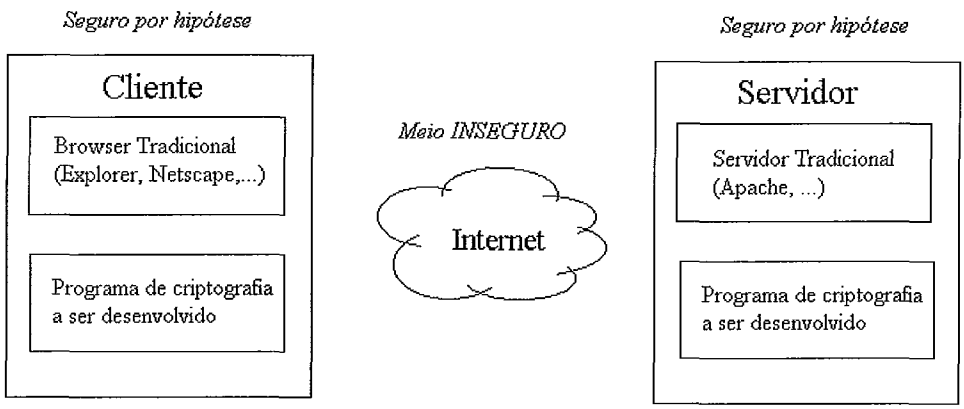


Figura 3.1: Visão geral do sistema WebSec.

Se supusermos que os ambientes do Cliente e do Servidor são seguros e que a insegurança reside exclusivamente no meio que os conecta, no caso prático, a internet, devemos criptografar todas as mensagens que trafeguem pelo meio, sem necessidade de preocupação das comunicações internas ao cliente e ao servidor.

A integração do sistema com o HTTP tradicional deve ser o mais transparente possível ao usuário e ao desenvolvedor WEB, preferencialmente de tão simples utilização quanto o SSL atualmente utilizado, de forma a evitarmos necessitades de treinamento de desenvolvedores e usuários finais das informações.

Devemos também comparar o desempenho do sistema com o SSL, padrão atualmente mais utilizado para a mesma finalidade. O principal parâmetro de desempenho a considerar, deve ser o tempo de transmissão completa da página HTML, desde a solicitação inicial do usuário do navegador, até a completa exibição das informações no *browser*.

3.3 Principais Parâmetros Considerados no Projeto

Os principais critérios para escolha das plataformas de desenvolvimento do protótipo do WebSec foram desempenho e difusão do uso da plataforma/software.

A difusão na utilização foi escolhida como critério principal de decisão pela própria característica prática do projeto. Desejamos oferecer uma alternativa real ao padrão vigente e, para isto, desejamos um protótipo o mais próximo possível da realidade da maioria dos usuários da *Web*. Acreditamos que um protótipo popular facilite a popularização do projeto, mesmo que, em alguns casos, nossa escolha não recaia sobre a melhor plataforma do ponto de vista técnico.

Como pretendemos uma solução que venha a ser agregada a softwares já existentes sem, no entanto, alterarmos seu código original, prevemos um acréscimo de esforço computacional decorrente das trocas de mensagens entre os dois aplicativos, que deverão trabalhar coordenados. Na plataforma SSL, os recursos de segurança estão incorporados ao servidor e o cliente não sofrendo sobrecarga decorrente de troca de mensagem entre aplicativos distintos. Portanto, desejamos plataformas de desenvolvimento que não comprometam desempenho, a fim de produzirmos uma solução que não tenha um overhead muito grande se comparado ao SSL, no que tange a velocidade de transmissão fim a fim.

3.4 Projeto do Servidor WebSec - SWS

Batizamos a nossa implementação do servidor de SWS, acrônimo para **Servidor WebSec**.

SWS funciona de forma parecida com um *proxy*. Porém esta próximo do servidor HTTP e não do cliente como um *proxy* tradicional. Receberá solicitações do cliente e as repassará ao servidor HTTP tradicional como se fossem suas. Ao receber as informações do servidor HTTP tradicional as criptografará e enviará ao cliente. Ao

receber informações criptografadas do cliente as decifrará e repassará ao servidor HTTP tradicional.

Na Figura 3.2 podemos verificar como SWS se interpõe entre a Internet e o servidor tradicional, criptografando os dados destinados a màquinas externas e decriptografando os dados que chegam de máquinas externas.

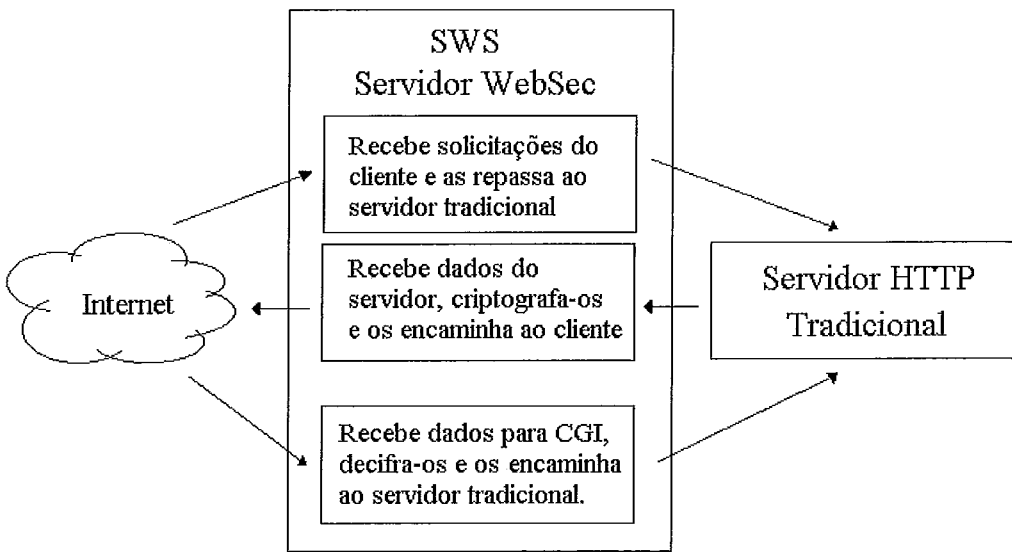


Figura 3.2: SWS, Servidor WebSec.

3.4.1 Escolha da Plataforma

Seguindo os critérios anteriormente descritos, escolhemos a plataforma CONECTIVA LINUX 4.2 para implementação do servidor.

Segundo [25] e [7] o LINUX movimentou o mercado de servidores no ano de 1999.

> *"O tamanho do mercado de servidores foi de aproximadamente 17 bilhões de dólares. O Windows gerou aproximadamente 8 bilhões de dólares em receitas, enquanto que o Linux gerou menos de 100 milhões. Se considerarmos que o Linux já detém uma parcela substancial do mercado, estes números são surpreendentes."*

Revista do Linux, outubro de 2000.

Os números são ainda mais surpreendentes se considerarmos que o Linux é adquirido por uma fração do preço das licenças para o sistema da Microsoft, ou mesmo sem qualquer custo, através de revistas ou de sitios de *ftp* (*File Transfer Protocol)* gratuitos na internet.

A Figura 3.3 mostra os números do IDC, instituto de pesquisas Norte Americano, sobre novos licenciamentos de softwares no ano de 1999.

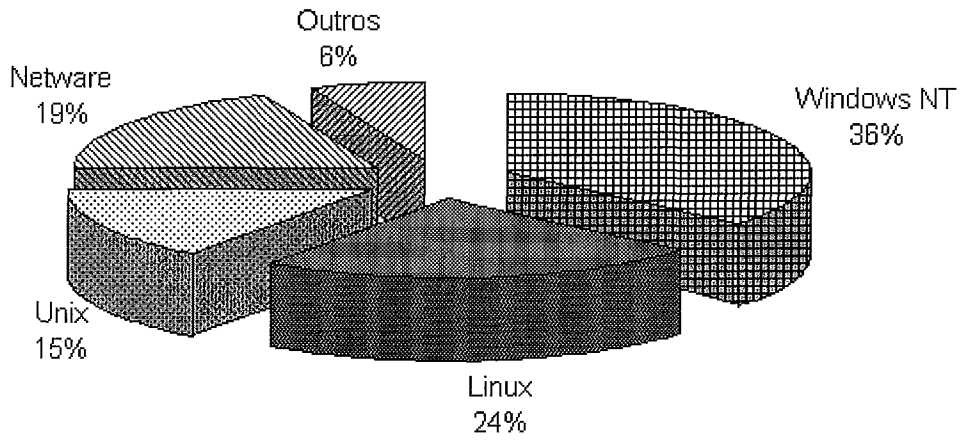


Figura 3.3: Licenciamentos para ambientes operacionais de servidores de rede no ano de 1999 [7].

Se considerarmos que o Linux é um sistema da família UNIX, e que os softwares desenvolvidos para Linux são de portabilidade imediata para o UNIX, podemos dizer que 39% das novas licenças para software de servidores de rede no ano de 1999 foram do mundo UNIX, contra 36% do Windows NT.

Escolhemos o Conectiva Linux por ser esta distribuição, em nosso entendimento, a que possui melhor chance de popularização no Brasil. A conectiva é a terceira maior distribuidora Linux do mundo, com mercado centrado na América Latina e Península Ibérica. É a distribuição com maior interesse na tradução para o idioma português e finalmente, mas não menos importante, traz conhecimento para o Brasil, formando técnicos e *know-how* nacional.

No Linux escolhemos o GCC o *GNU C Compiler*, que é compilador das linguagens C e C++, por ser a ferramenta de desenvolvimento mais popular do Linux.

Projetado SWS, passamos a definição de seu principal interlocutor, o cliente.

3.5 Projeto do Cliente WebSec – CWS

CWS deve ser invocado pelo navegador sempre que forem recebidos dados criptografados de SWS, ou se for necessária a criptografia de informações a serem remetidas ao servidor via CGI. A Figura 3.4 ilustra estas funcionalidades desejadas para CWS.

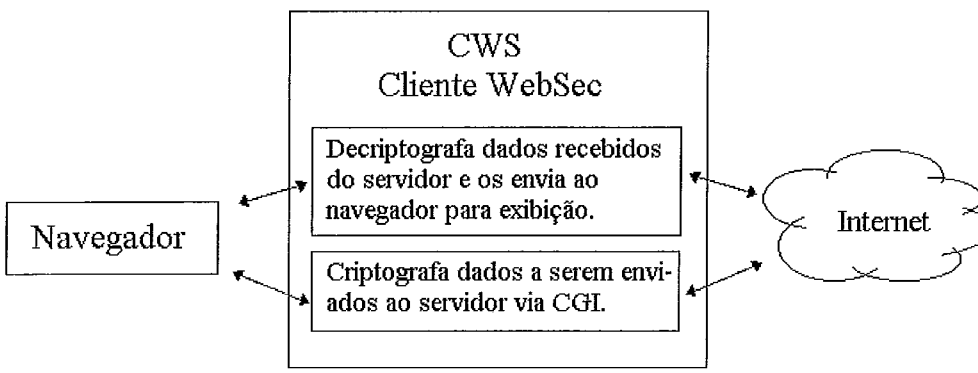


Figura 3.4: CWS, Cliente WebSec.

3.5.1 Java X *Plugin*

Duas grandes opções surgem neste ponto, desenvolver um *applet* java ou um *plugin*. Ambas as opções são tecnicamente viáveis e possuem vantagens e desvantagens.

A linguagem Java é orientada a objeto, o que facilita a reutilização de código, é de fácil aprendizado devido a sua semelhança com linguagem C, possui gerenciamento de memória semelhante ao LISP ou PROLOG, com liberação de memória automática, desocupando o programador desta tarefa, possui grande quantidade de objetos já construídos, inclusive para interface gráfica com usuário e, finalmente a principal vantagem de todas, a portabilidade.

Quando um programa java é compilado, ele não é totalmente compilado. Na verdade é gerado um código chamado "*byte code*" que não é o código nativo da plataforma onde o programa vai ser executado e sim um código a ser interpretado por

um programa, a máquina virtual java (*"java virtual machine"*) que funciona como um interpretador e que finalmente traduz o código para a linguagem nativa da plataforma. Como a maioria dos navegadores da atualidade implementam máquinas virtuais java, o código pode ser executado virtualmente em qualquer plataforma.

Esta última vantagem traz também a maior deficiência para Java. O desempenho. Como todo código interpretado, ele tem de ser primeiro traduzido para a linguagem nativa da plataforma, e isto torna Java de 7 a 15 vezes mais lento do que a linguagem C [11]. A Figura 3.5 ilustra o fato.

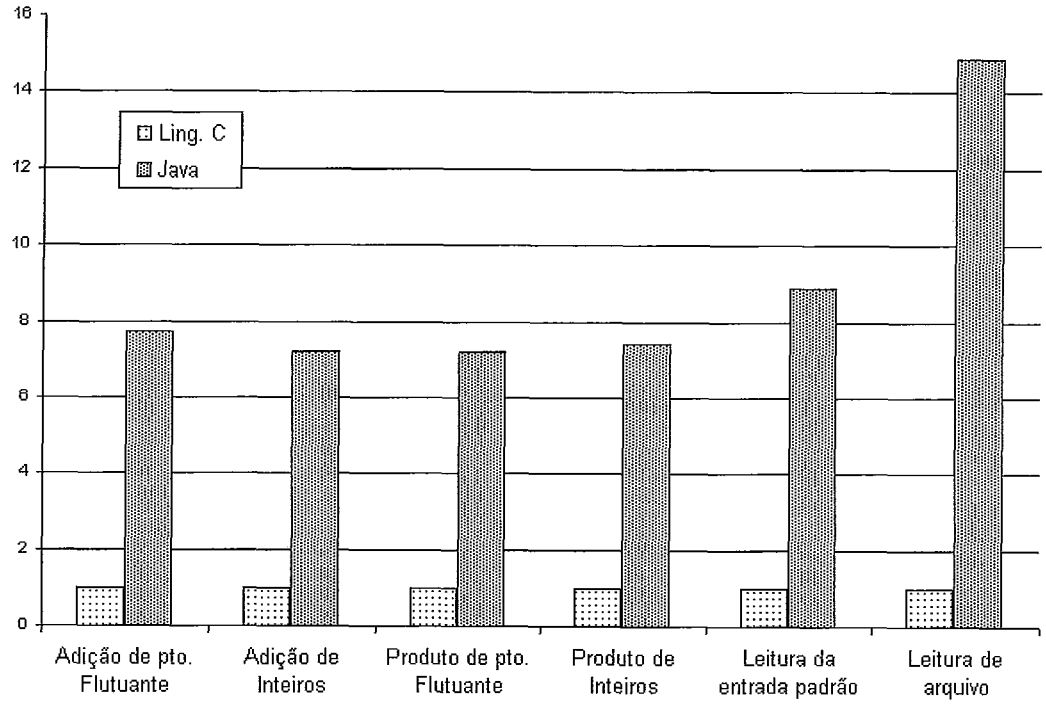


Figura 3.5: Comparação de desempenho das linguagens Java e C [11].

Outro ponto importante é a forma como os *applet*s são implementados. Um *applet* é transmitido ao cliente, toda vez que for necessário. Isto tem dois inconvenientes muito importantes.

O primeiro, e mais evidente, é que o tempo final do ponto de vista do usuário será acrescido do tempo de transmissão do *applet*, que não é desprezível. Enquanto que, no caso do *plugin*, este tempo é gasto apenas uma vez, na primeira transmissão. Depois que o *plugin* estiver instalado na máquina cliente, ele é carregado diretamente do disco local, muito mais eficiente do que o primeiro caso. A forma de operação dos *applet*s é vantajosa para aplicações que necessitam de atualizações de código freqüente, neste

caso, a atualização é feita naturalmente, de forma transparente para o usuário. Na verdade, atualiza-se todas as vezes, precisando ou não. O *plugin* é disparado por um cabeçalho MIME[3] no arquivo que se recebe, conforme discutiremos em seguida e, portanto, se desejamos atualizar seu código, temos que alterar o cabeçalho MIME dos dados para forçar um nova transferência do *plugin*. Como a troca freqüente de tipo MIME não é aconselhável, resta-nos a atualização manual, ficando a cargo do usuário a tarefa de apagar o arquivo correspondente ao *plugin* e forçar assim sua atualização via rede. Para o nosso caso, onde as atualizações só serão necessárias quando uma alteração de protocolo for exigida, o que não deve ser muito freqüente, o *plugin* é a melhor opção.

O segundo diz respeito à segurança. Tanto *plugin* quanto *applet* deverão ser transmitidos pela rede e, devem ser feitos de forma segura sob pena de comprometer todo o esforço do WebSec. Logo teremos de utilizar recursos padrão de segurança para transmitir *applet* ou *plugin* já que o WebSec não estará operacional sem que o cliente esteja instalado. Como visamos justamente aumentar a segurança dos recursos padrão, devemos minimizar a utilização dos mesmos. A transmissão rotineira do *applet* cifrado com o SSL, facilita a análise criptográfica, enfraquecendo de forma geral a segurança do sistema. Já o *plugin* seria transmitido apenas esporadicamente, a princípio, somente uma vez para cada cliente, o que dificultaria sobremaneira a quebra dos recursos do sistema de segurança como um todo.

Evidentemente, para aplicações críticas, o *plugin* pode ser instalado manualmente, por meio de um disco removível, ou uma rede local totalmente confiável (desconectada da INTERNET) evitando-se, assim, completamente, a utilização da segurança padrão da internet.

3.5.2 Escolha da plataforma

A escolha óbvia para a plataforma de CWS é o MS Windows. Líder inconteste do mercado de software para estações de trabalho no mundo, a plataforma gráfica da Microsoft não mostra sinais de que esteja sofrendo ameaças de qualquer de seus concorrentes. Segundo [7], 87% das novas licenças para ambientes operacionais vendidas no mundo foram para o Windows, como podemos verificar na Figura 3.6.

---

[3] MIME – *Multiporpose Internet Mail Extensions*. Conjunto de cabeçalhos especificados nas RFCs 822, , 1341 e 1521.

Portanto, apesar dos tão conhecidos problemas de estabilidade desta plataforma, ela se ajusta no perfil desejado, e foi utilizada para o protótipo de CWS.

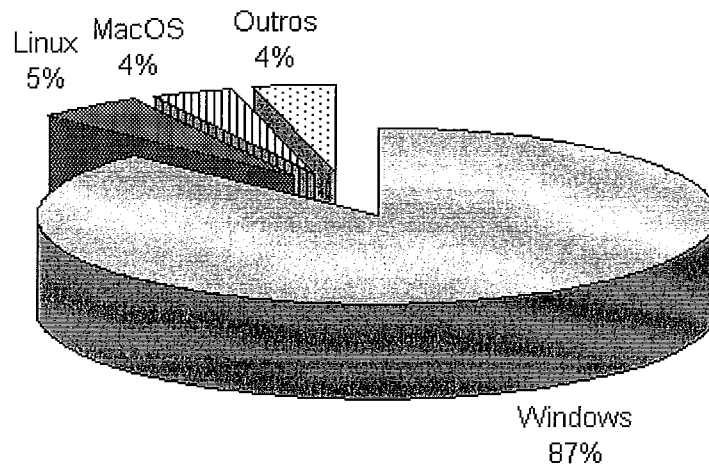


Figura 3.6: Licenciamentos para ambientes operacionais de estações de trabalho no ano de 1999 [7].

Selecionou-se o Microsoft Visual C++ para desenvolvimento de CWS, principalmente pelo Netscape disponibilizar um ambiente de desenvolvimento de *plugin*s para este compilador. É o *Netscape Plugin Software Developers Kit (pluginSDK)*, que facilitou sobremaneira o desenvolvimento do protótipo. O MS Visual C++ parece ser a escolha natural para o desenvolvimento de *plugin*s, pois a pequena literatura disponível sobre o assunto utiliza-o em seus exemplos e tutorais.

3.6 Algoritmo de criptografia - Blowfish

Apresentado por B. Schneier em [21], o Blowfish é um algoritmo de chave secreta (simétrico), livre de patentes, que cifra blocos de 64K com chave com tamanho variável de no máximo 448 bits.

Não existe forma conhecida de se provar matematicamente a segurança de um algoritmo de criptografia. Algoritmos são considerados bons quando um grande número de especialista tenta quebrá-lo por um período de tempo grande sem sucesso. Este é o caso do Blowfish, que vem sendo adotado por mais de 130 softwares diferentes, inclusive pelo FreeBSD, ambiente operacional conhecido por proporcionar ambientes muito seguros. Uma lista completa dos sistemas que utilizam o algoritmo pode ser obtida em "http://www.counterpane.com/products.html".

O Blowfish compreende duas fases, a primeira gera a partir da chave escolhida pelo usuário, uma chave muito maior, de 4.168 bytes, devendo aí o nome do algoritmo que "incha" a chave do usuário como o peixe conhecido em algumas regiões do Brasil como "baiacu".

Após a expansão da chave, o algoritmo utiliza uma função relativamente mais simples, por 16 vezes. Portanto, o que o Blowfish realmente faz é transferir a complexidade da tarefa para uma fase inicial, que é executada apenas uma vez, depois, a tarefa repetitiva de criptografia fica simplificada, o que diminui drasticamente a complexidade para volumes de dados muito grandes, sem comprometer o desempenho quando o volume não é tão grande assim.

Podemos verificar na Tabela 3.1 e na Figura 3.7 que o *Blowfish* apresenta considerável ganho de desempenho em relação a seus principais concorrentes.

Comparações de desempenho em um Pentium				
Algoritmo	**Ciclos por rodada**	**Número de rodadas**	**Ciclos por byte cifrado**	**Observações**
Blowfish	9	16	18	Livre utilização, não patenteado
Khufu / Khafre	5	32	20	Patenteado pela Xerox
RC5	12	16	23	Patenteado pela *RSA Data Security*
DES	18	16	45	Chave de 56 bits
IDEA	50	8	50	Patenteado pela *Ascom-Systec*
Triple-DES	18	48	108	

Tabela 3.1: Comparação de desempenho entre algoritmos de criptografia. Fonte: http://www.counterpane.com/speed.html

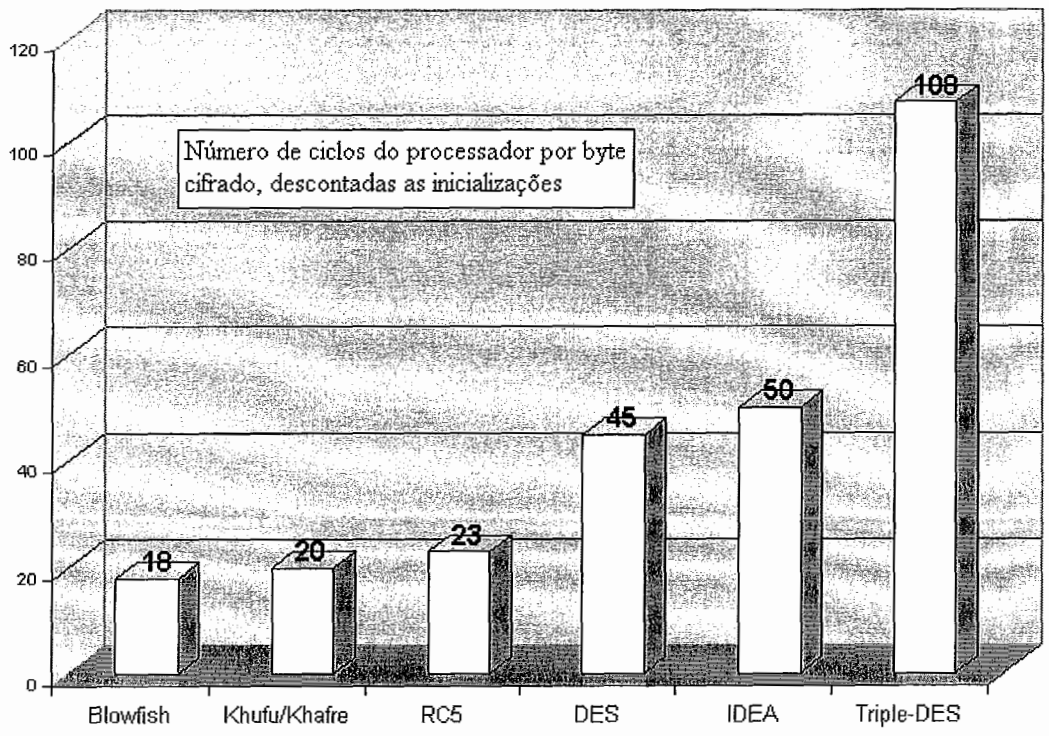


Figura 3.7: Comparação de desempenho entre algoritmos de criptografia. Fonte: http://www.counterpane.com/speed.html

Os números apresentados não envolvem a fase de inicial, o que é muito conveniente para o Blowfish, fato possivelmente explicado por terem sido gerados pela *Counterpane*, empresa de propriedade de Bruce Schneier, autor do algoritmo. Consideramos no entanto que as diferenças são muito grandes, dificilmente compensadas, mesmo com pequenos volumes de dados.

O item seguinte aborda uma importante ferramenta para desenvolvimento de software de segurança, que implementa os principais algoritmos de criptografia utilizados na atualizada, inclusive o *Blowfish*.

3.7 Biblioteca "cryptlib"

Cryptlib é um poderoso conjunto de ferramentas desenvolvido na Nova Zelândia por Peter Gutmann, que pode ser livremente utilizado e está disponibilizado via *ftp* em http://www.cs.auckland.ac.nz/~pgut001/cryptlib/, site situado na Universidade de Auckland. Na mesma localização são encontrados os fontes da biblioteca, desenvolvida

em C++ eliminando, portanto, qualquer possibilidade de *backdoors* ou recursos não documentados que enfraqueçam a segurança do pacote.

A biblioteca implementa diversos algoritmos de criptografia : Blowfish, CAST-128, DES, Triple DES, IDEA, RC2, RC4, RC5, Skipjack, MD2, MD4, MD5, RIPMD-160, SHA, HMAC-MD5, HMAC-RIPEMD-160, Diffie-Hellman, DAS, Elgamal, RSA. Suporta também gerenciamento de certificados padrão X-509v3, IETF PKIX, SET, SigG, Microsoft Authenticode, S/MIME, certificados da arquitetura cliente servidor SSL, manejo de listas de revogação de certificados, verificação de certificados, funções para implementação de uma central de certificação (CA) como geração de chaves para algoritmos de chave pública e geração de certificados. Suporta a criação de bancos de dados de chaves ou provê interface com uma grande quantidade de bancos de dados comerciais que podem ser utilizados para armazenamento de chaves e certificados. Suporta aceleradores de hardware como o cartão *FORTEZZA*, dispositivos PKCS#11 e *smart cards*.

Sem esta biblioteca dificilmente seria possível a implementação completa do protótipo do WebSec, principalmente no que é relativo ao gerenciamento de certificados. Certificados são definidos em ASN.1 e depois codificados segundo a *BER Basic Encoding Rules* (Regras Básicas de codificação) ou uma variação do *BER*, o *DER (Distinguished Encoding Rules)*. O resultado é novamente transformado segundo as regras denominadas *base-64*, que aumentam a redundância na codificação, evitando os caracteres estendidos da tabela ASCII. Desta forma evita-se troca de caracteres desconhecidos por siteos intermediários, evitando a corrupção do certificado. Cada grupo de 5 caracteres ocupa 64 bits se codificados via *base-64*, daí decorre o nome, contra os 48 usuais da tabela ASCII.

Toda esta codificação não é muito simples e, como se isto já não bastasse, erros de implementação em softwares de ampla utilização tornam a compatibilização de formatos para os protocolos uma tarefa ingrata. A *cryptlib*, segundo sua documentação, reconhece doze formatos diferentes de certificados (incluindo alguns codificados com erros) e dá acesso ao programador a estes dados de forma automática, com reduzido esforço de programação.

Definidos os componentes individuais do sistema, podemos passar a uma visão geral do fluxo de dados da proposta.

3.8 Visão Geral do Fluxo de Dados

A Figura 3.8 ilustra o funcionamento do WebSec como um todo, englobando clientes, servidores.

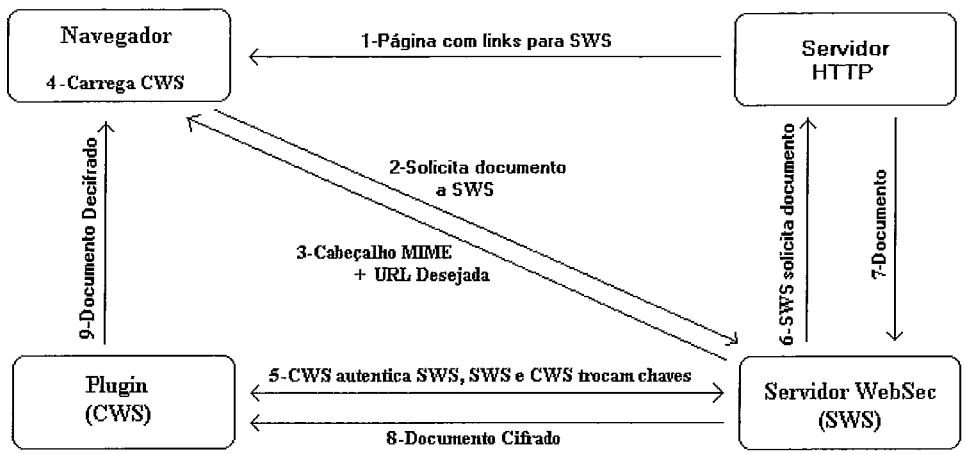


Figura 3.8: Fluxo de dados no Sistema WebSec.

A seguir descrevemos mais detalhadamente cada passo ilustrado na Figura 3.8.

1 – Página com links para SWS:

O desenvolvedor WEB deverá criar links para os recursos aos quais deseja uma transmissão segura, da forma tradicional, exceto por fazer referência a uma porta especial, no caso do protótipo desenvolvido, a porta 500.

2 – Solicita documento a SWS:

Quando o usuário selecionar um dos links para a porta do WebSec, o navegador estabelece uma conexão com a porta atendida por SWS e faz a solicitação do documento.

3 – Cabeçalho MIME + URL desejada

O servidor devolverá a URL solicitada ao navegador acrescida de um cabeçalho MIME **"application / x-websec"**.

A adição do tipo MIME é importante, pois o navegador associa um *plugin* com seus dados através de um tipo MIME. Quando um navegador é carregado, ele lê os *plugins* instalados, e registra os tipos MIME relativos a cada um deles. Desta forma o navegador sabe qual *plugin* carregar quando um tipo registrado é recebido.

4 – Carrega CWS:

Com a resposta do servidor, o *plugin* é carregado e recebe em uma *stream* de dados a URL desejada. Ele deverá então estabelecer uma conexão com SWS para buscar estes dados.

5 – CWS autentica SWS, SWS e CWS trocam chaves:

CWS e SWS trocam certificados digitais que podem ser verificados garantindo assim, a identidade das partes comunicantes. Feito isto, os programas utilizarão o algoritmo de chave pública RSA para troca de uma chave privada, a ser utilizada em um algoritmo de chave simétrica, o Blowfish no caso do protótipo.

CWS envia a URL desejada, para ser levantada por SWS e transmitida de forma segura.

6 – SWS solicita o documento:

SWS abre uma conexão com o servidor HTTP tradicional e solicita o recurso desejado.

7 – Documento:

O servidor HTTP convencional passa os dados do documento solicitado a SWS que o criptografa com a chave trocada com CWS.

8 – Documento Cifrado:

O documento criptografado pode então ser transmitido pelo meio inicialmente inseguro (internet) ao *plugin* que irá decifrá-lo.

9 – Documento Decifrado:

CWS decifra o documento recebido e o exibe na janela do navegador.

3.8.1 Autenticação e troca de chaves

O item 5 do esquema da Figura 3.8 esconde alguma complexidade e merece maiores esclarecimentos. Conforme podemos observar na Figura 3.9, após o envio da solicitação inicial do cliente ao servidor, SWS enviará seu certificado que deverá ser verificado por CWS, uma informação binária que indica a exigência ou não de envio de certificado pelo cliente e o tamanho da chave simétrica a ser utilizada com o *blowfish.*

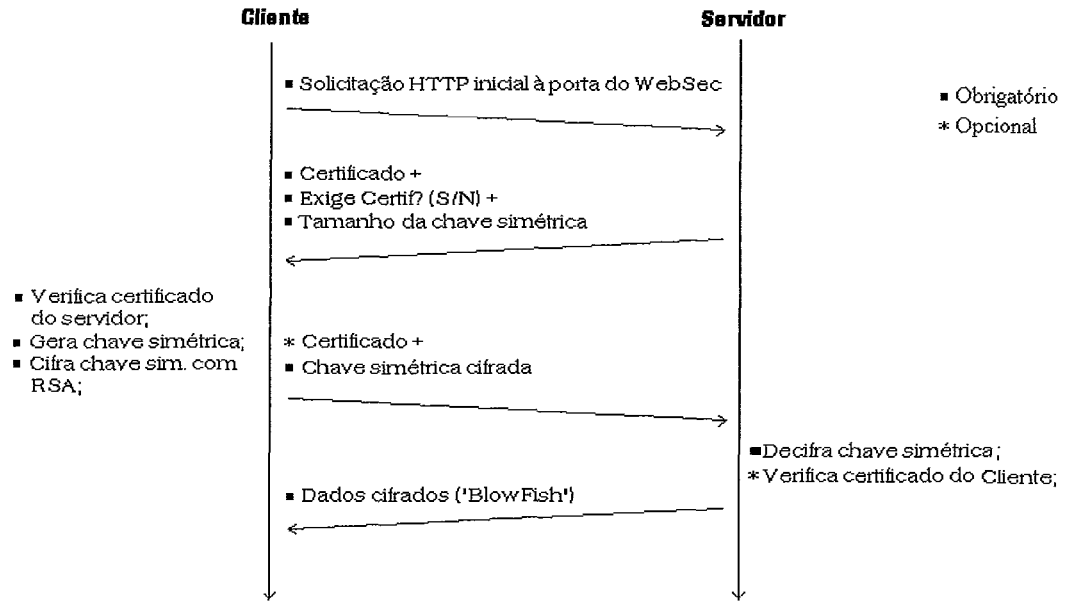


Figura 3.9: Autenticação e troca de chaves entre SWS e CWS.

O cliente deverá então gerar uma chave randômica do tamanho especificado e enviar ao servidor, criptografada com o RSA, utilizando a chave pública do servidor, recebida com seu certificado no passo anterior. Neste momento será enviado também o certificado do cliente, caso solicitado pelo servidor.

Desejamos manter a utilização do certificado do cliente opcional pois, em algumas aplicações, como no comércio eletrônico, não existe necessidade de identificação do cliente.

Tendo o servidor recebido a chave simétrica cifrada, deverá decifrá-la com sua chave secreta e iniciar o envio de dados criptografados com o *blowfish*.

O sistema prevê também o envio de dados no sentido inverso, do cliente para o servidor.

3.8.2 Envio de dados do cliente para o servidor

Para enviar dados de forma segura com o WebSec, não será possível a utilização de uma estrutura tão transparente ao programador *WEB* quanto foi o envio de dados do servidor ao cliente pois, caso utilizemos um formulário normal, com o marcador *FORM* do HTML, ficamos sem os meios para carregar o *plugin* e, conseqüentemente, impossibilitados de criptografar os dados para envio ao servidor.

Portanto, utilizaremos um arquivo de formato especial, onde estarão definidos *labels* e nomes de campos para que o *plugin* possa criar o formulário na área destinada a ele na janela do navegador e, quando o botão "enviar" for pressionado, o *plugin* criptografará os dados e os remeterá à SWS, para que as informações possam ser decifradas e encaminhadas por SWS ao servidor HTML tradicional.

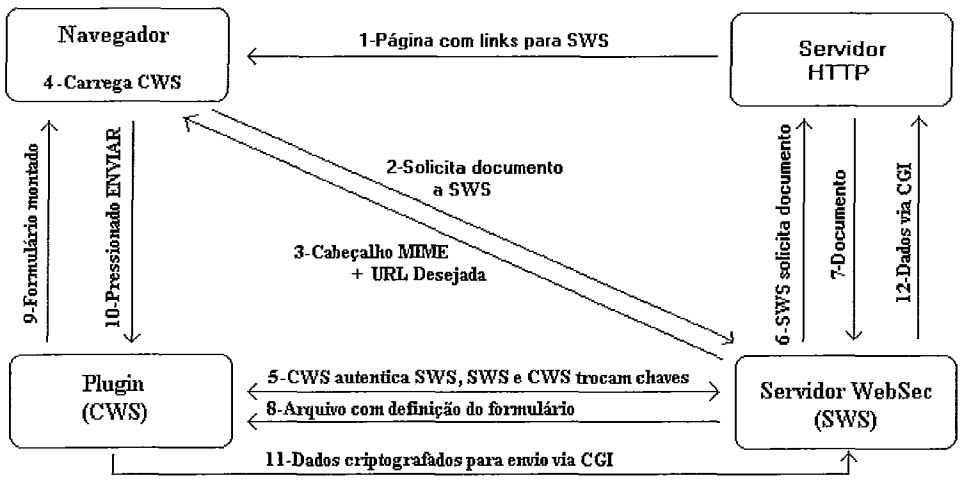


Figura 3.10: Fluxo de dados no sistema WebSec, incluindo formulários.

Até o item 7, o fluxo de dados mostrado na Figura 3.10 é idêntico ao da Figura 3.8, exceto pelo fato do documento solicitado conter um arquivo de formulário. Tal formato especial será reconhecido pelo *plugin* e este, ao invés de exibir os dados do documento recebido, os processará formando um conjunto de campos e botões que poderão ser editados pelo usuário como se fosse um formulário usual da WEB.

Os itens 9, 10, 11 e 12 representam o envio seguro dos dados ao servidor. O usuário clica em "enviar" no formulário montado pelo *plugin*, o *plugin* cifra os dados e os envia a SWS, SWS decifra e encaminha por meios tradicionais as informações ao servidor HTML.

O sistema, assim proposto, pode ser comparado com o SSL.

3.9 WebSec X SSL

Uma avaliação comparativa pode ser vista na Tabela 3.2 entre o WebSec e o padrão de fato para a transmissão de dados segura na WEB.

Nr.	SSL	WebSec
1	Melhor integração entre softwares de criptografia e demais aplicações pois a aplicação foi projetada levando em consideração a arquitetura do sistema de segurança desejada.	Maior *overhead* para troca de mensagens entre programas, pois tratamos aqui de processos independentes ou recursos das plataformas que não foram exatamente projetadas para a utilização que dela fizemos.
2	Padrão de criação de páginas amplamente difundido entre os desenvolvedores WEB	Pequenas adaptações são necessárias na forma de criação dos documentos hipertexto.
3 ☺	Utilização de novas tecnologias depende de implementação pelo fornecedor das aplicações.	Totalmente independente do fornecedor da plataforma de comunicação, podendo ser implementados novos algoritmos de criptografia ou tecnologias de distribuição de chaves sem que o fornecedor nem mesmo conheça do fato.
4 ☺	Necessidade de confiança nas informações do fornecedor sobre a segurança incluída nos produtos adquiridos	Recursos de segurança desenvolvidos localmente, sobre os quais podemos ter total certeza sobre a forma de implementação, mesmo utilizando softwares comerciais, sobre os quais não possuímos controle ou informações
5 ☺	Dependente de padronização entre plataformas cliente / servidor fornecidas por empresas diferentes.	Necessita desta padronização apenas para transmissão inicial do *plugin* e troca de chaves. A transmissão dos dados propriamente dita é independente de padronização.

Tabela 3.2: Comparação entre o SSL e o WebSec. O símbolo ☺ na primeira coluna indica vantagem do WebSec.

Em nosso protótipo, esperamos que a desvantagem do WebSec citada no item 1 seja compensada pela maior eficiência do *blowfish* em relação ao *DES*. Evidentemente, quanto maior for o volume de dados transmitido, maior pode ser o ganho do WebSec, podendo haver desvantagem apenas para pequenos arquivos. Do ponto de vista do usuário final, acréscimos em transferências pequenas, tendem a ser desprezíveis.

O item 2 também configura uma desvantagem de menor importância pois estimamos que o treinamento necessário para um desenvolvedor WEB se adapte ao WebSec não passe de algumas horas, na pior das hipóteses, mesmo no caso de formulários para transmissão de dados do cliente para o servidor, pois o formato do arquivo para criação de formulários foi definido bem mais simples do que no HTML padrão. Tal simplificação sendo possível por termos, no WebSec, um arquivo à parte para o formulário, enquanto no HTML padrão, os marcadores para criação de formulários são colocados junto com as demais *tags* da página.

Ilustra bem a vantagem descrita no item 4 da tabela, o conteúdo do aviso do CERT número *CA-2001-01*, de 09 de janeiro de 2001: *"Interbase Server Contains Compiled-in Back Door Account"*, que pode ser encontrado em *"http://www.cert.org/advisories/CA-2001-01.html"*. O documento descreve a existência da conta do usuário LOCKSMITH, incluída diretamente no código do servidor de banco de dados *Interbase* que provê acesso irrestrito a qualquer informação ali armazenada e, pior ainda, possibilita a inclusão de "cavalos de tróia" como trechos de código armazenados em bancos de dados. O usuário não pode ser apagado ou ter sua senha alterada, e instituições que utilizam o sistema dependem da correção do problema por parte do fabricante. O *backdoor* foi incluído pelos desenvolvedores do software para facilitar tarefas de manutenção e quando, recentemente, a Borland abriu o código do Interbase, a comunidade de usuários descobriu a vulnerabilidade. Ficam as questões: outros softwares de código fechado possuem características semelhantes? Quantos funcionários da Borland que detinham esta informação trocaram de emprego, antes de descobrirmos esta característica do Interbase?

4 Implementação dos Protótipos

Nas próximas seções realizamos alguns comentários sobre os códigos fonte que se encontram listados como anexos deste trabalho, que consideramos pertinentes.

4.1 Servidor - SWS

SWS foi implementado como um servidor orientado à conexão concorrente clássico. Ele cria um *socket*, e o conecta à porta 500, que convencionamos ser a porta do WebSec neste protótipo. O programa então passa a escutar esta porta até que uma conexão seja passada pelo sistema operacional. Neste ponto SWS se divide em dois processos filhos, um que irá tratar a conexão recém chegada e outro que irá continuar a aguardar por novos possíveis usuários que serão atendidos de forma concorrente.

O processo que tratará a nova conexão recebe a solicitação HTTP do cliente. Se tratar-se de um GET normal, o sistema entende que o *plugin* ainda não foi carregado no cliente e devolve a solicitação acrescida do MIME que causará a execução do *plugin* pelo navegador do cliente. Se um SGET for recebido, a solicitação está sendo feita por CWS. Neste caso o servidor envia um "N" pela conexão indicando que não é necessária a transmissão de certificado pelo cliente, seguido de "128" indicando que é desejada uma chave simétrica de 128 bits e finalmente o seu certificado.

Desta forma, estamos autenticando apenas o servidor, e não o cliente, como na maioria das transações de comércio eletrônico, onde a autenticação do servidor é mais importante, pois o cliente, que efetua o pagamento, já é autenticado na aplicação, através do fornecimento do número de seu cartão de crédito, ou simplesmente sua autenticação não é importante. Apesar disto, a autenticação do cliente é perfeitamente possível utilizando-se o WebSec. Para fins de comparação com os sistemas de segurança atualmente utilizados na WEB, não desejamos, neste protótipo, autenticar o cliente.

Por mostrar-se atualmente eficiente contra ataques de força bruta, foi fixado o tamanhos de 128 bits para a chave simétrica, mas em futuras versões destinadas à efetiva produção, e não somente a teste, como este protótipo, o ideal é utilizar um arquivo de configuração onde os usuários possam definir ambas as funcionalidades, a autenticação do cliente e o tamanho da chave simétrica.

Ao final da transmissão do seu certificado, SWS aguarda a recepção da chave simétrica. Depois de recebida e decifrada com sua chave secreta por meio do RSA, o servidor esta apto a estabelecer um diálogo seguro com CWS. Para tal, ele executa um novo *fork*, dividindo-se novamente em 2 processos concorrentes. Um que trata as mensagens originadas no cliente com destino ao servidor e outro que tratará as mensagens originadas no servidor com destino ao cliente.

O primeiro processo, que repassa dados do cliente ao servidor HTTP tradicional, o faz após retirar o "S" do início do comando "SGET" recebido, reconstituindo o comando "GET" previsto pelo HTTP, e decifra eventuais dados passados através do preenchimento de formulários, do cliente ao servidor, utilizando o *blowfish*. Exceto por estas suas alterações, o sistema simplesmente copia do *socket* mantido com o cliente para o *socket* mantido com o servidor.

O segundo processo, recebe dados do servidor HTTP tradicional e os repassa ao cliente, somente após criptografar todos os dados recebidos, exceto o cabeçalho MIME que acompanha todos os arquivos transmitidos via HTTP.

4.2 Cliente - CWS

4.2.1 Plugins

Plugins são, em essência, bibliotecas de vínculo dinâmico que residem em um diretório especial definido pelo *browser*. Em tempo de desenvolvimento, criam-se recursos (*windows resources*) que definem que tipo MIME o *plugin* está preparado para tratar e qual extensão de arquivo deve ser interpretada por ele. Quando o navegador é carregado, ele consulta estes recursos e registra as associações *plugin* e MIME, extensão. Quando recebe um conjunto de dados com um tipo MIME/extensão registrados, o *plugin* é carregado para tratar aquele dado. Portanto, a instalação de um *plugin* resume-se a copiar o arquivo de extensão DLL para o diretório definido pelo navegador.

Utilizando o *Netscape Communicator 4.03*, podemos verificar os *plugins* registrados acionando *Help / Sobre módulos externos*, que produz uma página semelhante à reproduzida na Figura 4.1.

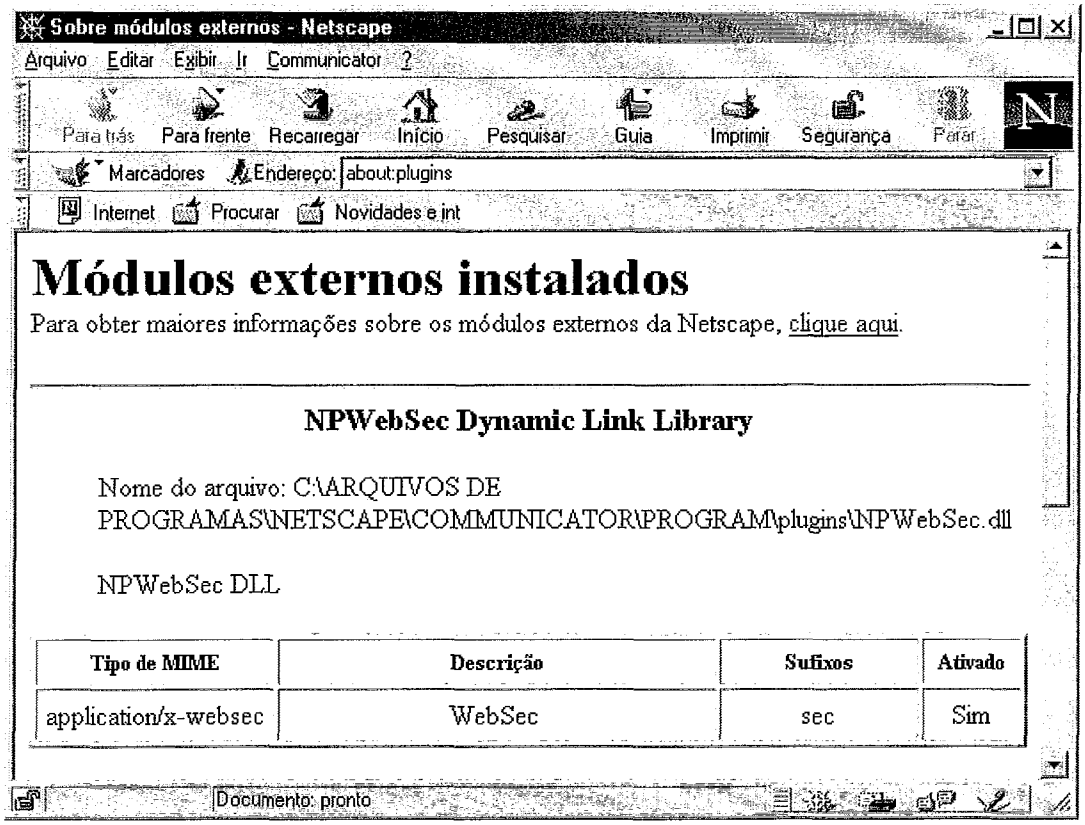


Figura 4.1: Página "About Plugins" do "Netscape Comunicator".

O criador da página WEB que deseja incluir um tipo de dados a ser tratado por um *plugin* deve fazê-lo utilizando o marcador *EMBED*, onde ele pode definir a área da janela do navegador que fica reservada ao *plugin* com os atributos *"WIDTH"* e *"HEIGHT"*. Outro atributo importante do marcador *EMBED* é o *PLUGINURL*, que define a fonte de onde podemos instalar o *plugin* caso ele não se encontre na máquina local. Pode-se também utilizar atributos não padronizados no HTML, que são passados ao *plugin* para que este os interprete. Em CWS, utilizamos o atributo FORM, que indica se o arquivo recebido deve ou não ser tratado como um formulário.

Diversas funções padrão devem ser definidas no *plugin* e são chamadas pelo navegador. Por exemplo, quando é carregado, uma instância do *plugin* é criada com uma chamada à função *NPP_New* que deve ser definida pelo usuário.

Duas interfaces são definidas, uma com funções criadas pelo *plugin* e utilizadas pelo navegador, e uma com funções implementadas no navegador que podem ser chamadas pelo *plugin*. A primeira tem as letras "NPP_" (*Netscape Plugin: Plugin Defined)* como prefixo de todas as suas funções e a segunda tem suas funções iniciadas

por "NPN_" (*Netscape Plugin: Navigator Defined*). A título ilustrativo temos na Tabela 4.1, adaptada de [16], as principais funções definidas nas duas interfaces.

Função	**Descrição**
`NPP_Destroy`	Termina uma instância do *plugin.*
`NPP_DestroyStream`	Chamada quando uma transferência de dados é completada.
`NPP_GetJavaClass`	Retorna uma classe java associada ao *plugin.*
`NPP_HandleEvent`	Gerenciador de eventos específico para a plataforma *Macintosh.*
`NPP_Initialize`	Inicialização Global.
`NPP_New`	Cria uma nova instância do *plugin.*.
`NPP_NewStream`	Chamada quando um novo fluxo de dados é criado.
`NPP_Print`	Gerenciador de impressão.
`NPP_SetWindow`	Chamado quando é necessária atividade na janela de responsabilidade do *plugin.*
`NPP_Shutdown`	Finalização global.
`NPP_StreamAsFile`	Associa um fluxo de dados a um nome de arquivo.
`NPP_URLNotify`	Notifica o final de uma URL requisitada.
`NPP_Write`	Chamada para passar dados ao *plugin.*
`NPP_WriteReady`	Chamada para verificar se o *plugin* está apto a receber mais dados.
`NPN_DestroyStream`	Termina um fluxo de dados.
`NPN_GetJavaEnv`	Retorna o ambiente de execução java.
`NPN_GetJavaEnv`	Retorna o ambiente de execução java.
`NPN_GetJavaPeer`	Retorna o objeto java associado ao *plugin.*

NPN_GetURL	Requisita a criação de um fluxo de dados.
NPN_GetURLNotify	Requisita a criação de um novo fluxo de dados com notificação.
NPN_MemAlloc	Aloca memória.
NPN_MemFlush	Específico para Macintosh.
NPN_MemFree	Libera memória.
NPN_NewStream	Cria um novo fluxo de dados.
NPN_PostURL	Processa um comando HTML "POST" em uma determinada URL.
NPN_PostURLNotify	Processa um comando HTML "POST" em uma determinada URL e notifica o resultado.
NPN_RequestRead	Solicita dados de um fluxo.
NPN_Status	Mostra uma mensagem na barra de status.
NPN_UserAgent	Obtém o campo "UserAgent" do navegador.
NPN_Version	Obtém a versão da API corrente.
NPN_Write	Escreve a um fluxo de dados.

Tabela 4.1: Funções definidas na API entre o navegador e um *plugin*.

A descrição detalhada de cada uma das funções citadas na Tabela 4.1 pode ser encontrada em http://developer.netscape.com, com os parâmetros necessários e os valores de retorno de cada função e é consulta obrigatória para o desenvolvimento de qualquer *plugin*.

Não excencial, mas de grande ajuda é o *Netscape Plugin SDK (Software Developers Kit)* que pode ser encontrado na mesma URL. Ele traz um projeto completo para ser utilizado com o *Microsoft Visual C++*, com um esqueleto de um *plugin* montado, com protótipos para todas as funções citadas na Tabela 4.1, e necessitamos basicamente adicionar nosso próprio código em linguagem C a cada uma das funções de forma a dar ao *plugin* as funcionalidades desejadas.

4.2.2 O cliente WebSec

A função NPP_New, executada no instanciamento do *plugin*, recebe dois argumentos "argn" e "argv", contendo os argumentos do marcador EMBED utilizado pelo criador da página para referenciar o documento cuja transmissão segura é desejada. Portanto, nesta função procuramos pelo parâmetro FORM caso ele contenha o valor "*true*" (utilizada a palavra em inglês a fim de manter homogeneidade com o HTML padrão) o sistema sabe que estará recebendo um formulário, e deverá criar os objetos para entrada de dados pelo usuário. Caso FORM contenha o valor "*false*", sabemos que os dados devem ser exibidos diretamente na área da janela do *browser* reservada para o *plugin* e, caso o parâmetro FORM simplesmente não esteja presente, entendemos que o marcador EMBED não tenha sido utilizado e CWS abre uma novo fluxo de dados para o *browser*, de forma que os dados recebidos pelo sistema WebSec sejam exibidos em uma nova instância do navegador, e não inseridos como parte de uma página. Uma variável global denominada "tipo" é ajustada com o valor 0, 1 ou 2 conforme o atributo FORM seja "*false*", inexistente ou "*true*", respectivamente.

Caso um novo fluxo de dados seja necessário entre o *plugin* e o browser, ele será criado quando a função NPP_NewStream, que notifica a criação do fluxo no sentido do navegador para o cliente, for chamada pelo navegador.

A função NPP_WriteReady deve retornar o número de bytes que o *plugin* está pronto para receber e, portanto, definirá o tamanho do bloco de dados a ser processado em cada iteração do algoritmo de criptografia. Fixamos então o retorno desta função em 8, tendo em vista que o algoritmo de criptografia *blowfish* trabalha com blocos de 64 bits.

NPP_Write recebe efetivamente os dados que são decifrados e guardados em variáveis globais para que possam ser exibidos posteriormente na área reservada para o *plugin*, exceto no caso em que são transmitidos de volta, em um novo fluxo de dados. A função de construção da janela é a *PluginWindowProc*, que constrói o formulário ou exibe os dados recebidos, conforme o caso.

5 Avaliação da Solução Proposta

A seguir discorremos sobre os principais fatores que influem no desempenho da solução proposta, bem como discutimos alguns dados experimentais obtidos com os protótipos, tendo em mente principalmente a comparação do WebSec com o SSL.

5.1 Tempo de latência inicial

Como podemos verificar na Figura 3.8, o processo de carga do plugim envolve uma requisição inicial feita a SWS pelo browser, que neste momento ainda não tem o *plugin* carregado e, por este motivo, ainda não está apto a receber dados criptografados. Portanto, o servidor é obrigado a devolver o conteúdo do comando GET padronizado pelo HTTP ao browser, acrescido de uma linha no cabeçalho contendo o tipo MIME que está associado ao *plugin* (application / x-websec) para provocar a carga do *plugin* e possibilitar o inicio da troca segura de dados.

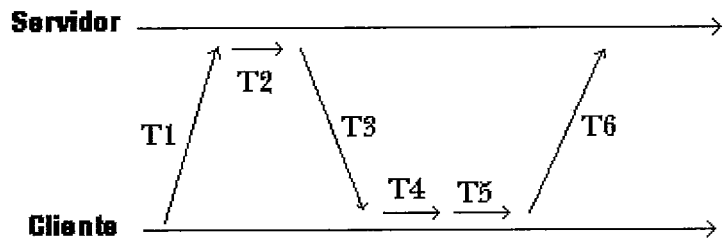


Figura 5.1: Tempos envolvidos no início da troca de dados com o WebSec

Todo este processo pode ser visualizado na Figura 5.1 onde temos:

T1 – Solicitação inicial do *browser* para o SWS;

T2 – SWS inclui cabeçalho MIME aos dados recebidos;

T3 – SWS devolve dados alterados ao *browser*;

T4 – *Browser* recebe dados e carrega o *plugin* (CWS) para tratálos;

T5 – CWS recebe solicitação;

T6 – CWS estabelece conexão *socket* com SWS.

Os tempos de T2 a T6 são decorrentes da tecnologia de *plugin* adotada no WebSec e, portanto, não são necessários quando utilizamos o SSL. Por este motivo consideramos importante conhecermos quantitativamente o período de tempo de T2 a T6, que pode facilmente ser medido através de pequenas alterações em SWS, de forma a registrarmos o relógio interno do sistema quando é recebida a primeira solicitação do browser (ponto entre T1 e T2) e o recebimento por SWS da solicitação de CWS (final de T6). Chamamos "T2/T6"o período de tempo assim definido.

Como o período T2/T6 envolve a comunicação entre cliente e servidor, é influenciado pelo tempo de propagação dos pacotes entre estes pontos, portanto, realizamos medições em 3 situações distintas. Para cada situação as medições foram repetidas por 10 vezes.

Situação A

Realizamos os testes com dois computadores pessoais padrão IBM-PC ligados por meio de um cabo coaxial fino padrão BNC e dois cartões de rede padrão ETHERNET tipo NE2000, de 10 Mbits / s.

Como SERVIDOR foi utilizado um sistema com processador Pentium com clock de 200 MHz, com 32 Mbytes de memória RAM instalada, rodando Linux 2.2.13-9cl distribuído pela Conectiva SA (distribuição 4.2). Servidor Apache versão 1.3 acrescido do pacote mod_SSL que implementa uma *interface* entre o Apache e o OpenSSL, o software que finalmente implementa a criptografia. A versão do OpenSSL utilizada foi a 0.9.4, de agosto de 1999.

O cliente foi executado em um sistema com processador AMD K6II com clock de 450 MHz, 64 Mbytes de memória RAM, sistema operacional Windows 98SE, com 86% dos recursos livres. Repetimos o teste para os Navegadores Explorer 5.00.2314.3500IC e Netscape Navigator 4.03.

Tempo médio do trajeto de ida e volta completo dos pacotes, obtido com o utilitário PING, enviados 50 pacotes: menor que 10 milisegundos.

T2/T6 médio: 53 milisegundos.

Desvio Padrão: 3,17

Situação B

Cliente: Mesmo K6 II descrito na situação A, conectado à um provedor de acesso INTERNET comercial atrevés de linha ISDN e cartão TRELIS a 56 Kbits/S.

Servidor: Pentium III Clock: 500, 256 Mbytes de memória RAM rodando FreeBSD 4.0 e servidor Apache versão 1.3.14.

Tempo médio do trajeto de ida e volta completo dos pacotes, obtido com o utilitário PING, enviados 50 pacotes : 88 milisegundos.

T2/T6 médio: 152 milisegundos.

Desvio Padrão: 3,56

Saída do comando *tracert* (*traceroute*) entre cliente e servidor:

Rastreando a rota para ravel.ufrj.br [146.164.32.67]
com no máximo 30 saltos:

1 42 ms 54 ms 42 ms stanislaw.artnet.com.br [200.251.141.74]
2 55 ms 41 ms 55 ms villalobos.artnet.com.br [200.251.141.65]
3 41 ms 55 ms 55 ms artnet-S8-1-acc01.bhe.embratel.net.br [200.251.254.137]
4 54 ms 42 ms 41 ms ebt-F5-0-0-dist01.bhe.embratel.net.br [200.255.153.225]
5 82 ms 55 ms 96 ms ebt-A8-0-2-core03.rjo.embratel.net.br [200.255.153.42]
6 55 ms 123 ms 96 ms ebtdist01.bsa.embratel.net.br [200.255.197.161]
7 96 ms 96 ms 137 ms rnp-br- -dist01.bsa.embratel.net.br [200.252.247.138]
8 97 ms 96 ms 123 ms rj.bb3.rnp.br [200.143.255.130]
9 69 ms 96 ms 96 ms cisco7206-atm-cbpf.rederio.br [200.20.94.41]
10 110 ms 83 ms 110 ms 200.20.94.9
11 82 ms 96 ms 110 ms gw-fddi.ravel.ufrj.br [146.164.1.20]
12 69 ms 96 ms 96 ms protheus.ravel.ufrj.br [146.164.32.67]

Situação C

Cliente: Mesmo K6 II descrito na situação A, conectado a um provedor de acesso à INTERNET comercial através de linha ISDN e placa PCI TRELIS a 56 Kbits/s.

Servidor: Athlon com clock de 700 MHz, 64Mbytes de memória RAM instalada, executando Linux 2.2.13-9cl distribuído pela Conectiva SA (distribuição 4.2). Servidor HTTP Apache versão 1.3. Esta máquina é utilizada como servidor HTTP do Colégio Técnico Universitário da Universidade Federal de Juiz de Fora (MG) e se conecta com link dedicado de 128Kbits/s com a rede TCP/IP da Universidade Federal de Juiz de Fora.

Tempo médio do trajeto de ida e volta completo dos pacotes, obtido com o utilitário PING, enviando 50 pacotes: 230 milisegundos.

T2/T6 médio: 305 milisegundos.

Desvio Padrão: 3.89.

Saída do comando *tracert* (*traceroute*) entre cliente e servidor:

Rastreando a rota para host1.ctu.ufjf.br [200.131.56.207]

com no máximo 30 saltos:

1 69 ms 41 ms 55 ms stanislaw.artnet.com.br [200.251.141.74]

2 55 ms 55 ms 55 ms villalobos.artnet.com.br [200.251.141.65]

3 55 ms 55 ms 41 ms artnet-S8-1-acc01.bhe.embratel.net.br [200.251.254.137]

4 41 ms 41 ms 55 ms ebt-F5-1-0-dist01.bhe.embratel.net.br [200.255.153.233]

5 54 ms 42 ms 55 ms ebt-A8-0-1-core03.rjo.embratel.net.br [200.255.153.14]

6 69 ms 109 ms 83 ms dist01.bsa.embratel.net.br [200.255.197.153]

7 82 ms 96 ms 96 ms rnp- dist01.bsa.embratel.net.br [200.252.247.138]

8 110 ms 110 ms 96 ms sp.bb3.rnp.br [200.143.255.129]

9 124 ms 164 ms 248 ms mg.bb3.rnp.br [200.143.255.131]

10 69 ms 123 ms 165 ms co-bhz1.bb.redeminas.br [200.19.159.52]

11 82 ms 110 ms 110 ms 200.19.158.46

12 123 ms 151 ms 151 ms gateway.cpd.ufjf.br [200.131.19.225]

13 96 ms 96 ms 96 ms cisco07.cpd.ufjf.br [200.131.19.228]

14 178 ms 151 ms 137 ms 200.131.56.193

15 151 ms 137 ms 151 ms 200.131.56.207

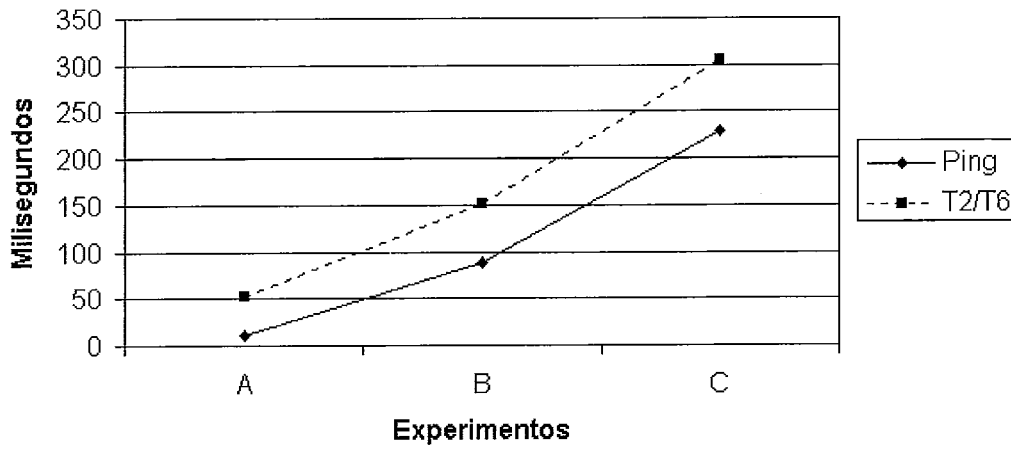


Figura 5.2: Tempos médios de latência nas situações A, B e C.

Como podemos verificar na Figura 5.2, a latência inicial inserida pelo processo de carga do *plugin* está relacionada de forma linear com o tempo de propagação dos pacotes na rede e, portanto, não trará acrescimos que inviabilizem a utilização da metodologia.

5.2 Troca de dados entre Servidores

A troca de dados entre o servidor WebSec e o servidor HTTP tradicional via *socket* , traz grande versatilidade ao sistema, possibilitando que apenas um servidor WebSec sirva a todo um domínio, desde que este conjunto de máquinas seja considerado seguro por hipótese e, como podemos verificar nas Figuras 5.3 e 5.4, com dados extraídos de [27], a utilização de *sockets* não inclui perdas de desempenho significativas, trazendo até mesmo, em alguns casos, melhorias nos tempos de troca de mensagens.

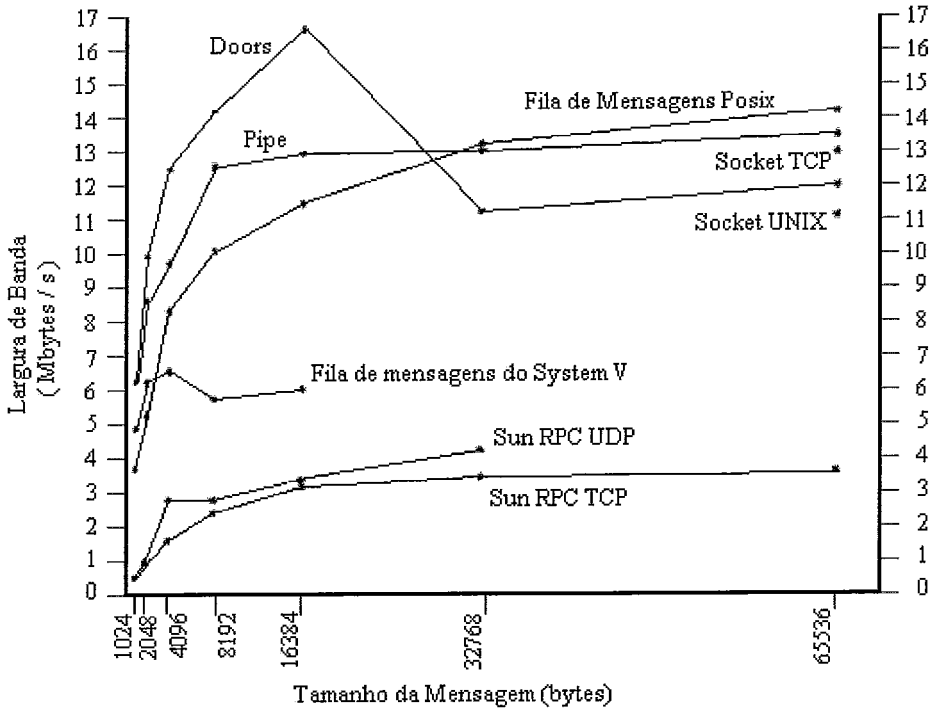


Figura 5.3: Desempenho de diversas formas de troca de mensagens (Solaris 2.6)

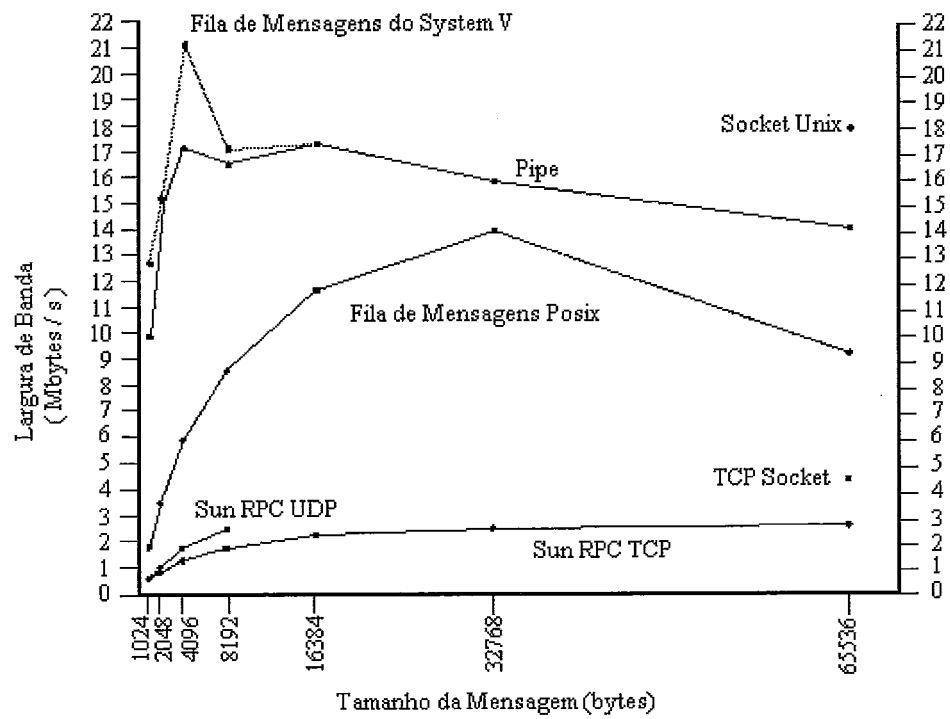


Figura 5.4: Desempenho de diversas formas de troca de mensagens (Digital Unix 4.0B)

Tendo em vista que as principais implementações do SSL são feitas tendo a parte responsável pela criptografia como um módulo a parte do servidor, não consideramos que, para efeitos práticos, exista diminuição do desempenho decorrente da troca de mensagens entre os servidores.

5.3 Resultados Esperados

Consideramos, portanto, que o desempenho da metodologia proposta pode ser influenciado principalmente por dois aspectos:

1 – Aumento do tempo de latência na conexão devido à necessidade de carga do *plugin* que antes não se fazia necessária;

2 – Aumento do desempenho geral do processo devido a maior eficiência do *blowfish* em comparação com os demais algoritmos de criptografia utilizados.

Tendo isto em mente, passamos a testes de desempenho realizados com o protótipo desenvolvido cujos códigos se encontram como anexos A e B.

5.4 Ambiente de teste

Realizamos os testes com dois micros pessoais padrão IBM-PC ligados por meio de um cabo coaxial fino padrão BNC e dois cartões de rede padrão ETHERNET tipo NE2000, de 10 Mbits/s.

Como servidor foi utilizado um *Notebook* IBM "Thinkpad" 390, com processador Celeron (Mendocino) de 333 MHz (363,72 "bogomips"), executando sistema operacional Conectiva Linux versão 5.0, com kernel versão 2.2.14, ambiente gráfico KDE 1.1.2 e servidor de HTTP Apache 1.3.12 e Open SSL 0.9.4. A máquina possui 96 Mbytes de memória instalada, sendo que no momento dos testes, com os servidores carregados, dispúnhamos de 11.246 Kbytes de memória livre.

O programa cliente foi executado em um AMD K6II com clock de 450 MHz, 64 Mbytes de memória RAM, sistema operacional Microsoft Windows ME versão 4.90.3000, com 72% dos recursos livres, após todos os softwares carregados. Repetimos o teste para os Navegadores Explorer 5.50.4134.0100IC e Netscape Comunicator 4.78 (versão em inglês).

Para efetuarmos a medição, foram criadas páginas HTML, que podem ser observadas no anexo C, simulando a transmissão de um extrato de conta bancária. Na página consta uma figura no formato .jpeg e um arquivo *extrato.sec* contendo o extrato propriamente dito, inserido com o marcador *<embed>* e solicitado através da porta 500 (atendida pelo *WebSec*).

Para medirmos o tempo de carga total da página, utilizamos um código *javascript* no início da seção *BODY* que armazena a hora do sistema, e associamos ao evento *ONLOAD,* que ocorre quando a página termina de ser carregada, um outro script que subtrai a hora atual do sistema da hora inicialmente armazenada, resultando em um valor proporcional ao tempo de carga total da página, em milisegundos, do ponto de vista do cliente.

Transmitimos arquivos de 100, 250, 500, 750, 1.000, 1.250, 1.500, 1.750, 2.000, 2.250, 2.500, 2.750 e 3.000 Kbytes 10 vezes cada um e observamos os seguintes resultados.

5.5 Resultados

Na Tabela 5.1 podemos observar os tempos de transmissão de arquivos texto com tamanhos de 100 a 3.000 Kbytes utilizando os navegadores Microsoft Explorer e Netscape Navigator, cada um com o WebSec e com o SSL.

Nas Figuras 5.5 e 5.6 podemos observar representações gráficas destes dados. Na Figura 5.5, estão representados os tempos de transmissão utilizando o Netscape com o WebSec e com o SSL, e o Explorer utilizando o WebSec. Separamos os dados relativos ao desempenho do Explorer utilizando o SSL na Figura 5.6 pois, neste caso, os tempos de transmissão apresentaram evolução muito diferente das demais situações.

Enquanto as curvas da Figura 5.5 apresentam evolução praticamente linear com o aumento do tamanho do arquivo transferido, a Figura 5.6 mostra dados que crescem de forma aproximadamente quadrática, conforme demonstramos graficamente, traçando a curva correspondente à função polinomial $\mathbf{y = 8569\ x^2 - 29224\ x + 29915}$.

Volume Transfe-rido (Kbytes)	Explorer + WebSec		Netscape + WebSecS		Explorer + SSL		Explorer + SSL	
	Média (10 exp)	Desvio Padrão	Média (10 exp)	Desvio Padrão	Média (10 exp)	Desvio Padrão	Média (10 exp)	Desvio Padrão
100	554	88	410	120	999	22	351	27
250	892	77	687	96	5.718	115	713	8
500	1.417	67	1.330	192	24.156	850	1.242	61
750	1.896	76	1.941	198	56.077	1.475	1.800	73
1.000	2.709	448	2.495	180	101.057	2.798	2.318	34
1.250	3.401	279	3.306	382	166.72	4.423	2.800	37
1.500	4.102	545	3.829	419	244.332	5.797	3.450	355
1.750	4.517	188	4.480	434	353.166	14.423	3.968	43
2.000	5.357	704	4.405	176	452.622	9.085	4.568	178
2.250	6.360	346	5.613	171	577.505	42.32	4.955	46
2.500	6.960	797	6.182	112	741.570	6.297	5.615	65
2.750	7.915	846	7.003	162	919.812	12.385	6.186	112
3.000	8.865	938	7.477	124	1104.376	9.756	6.615	65

Tabela 5.1: Resultados das medições realizadas (WS=WebSec)

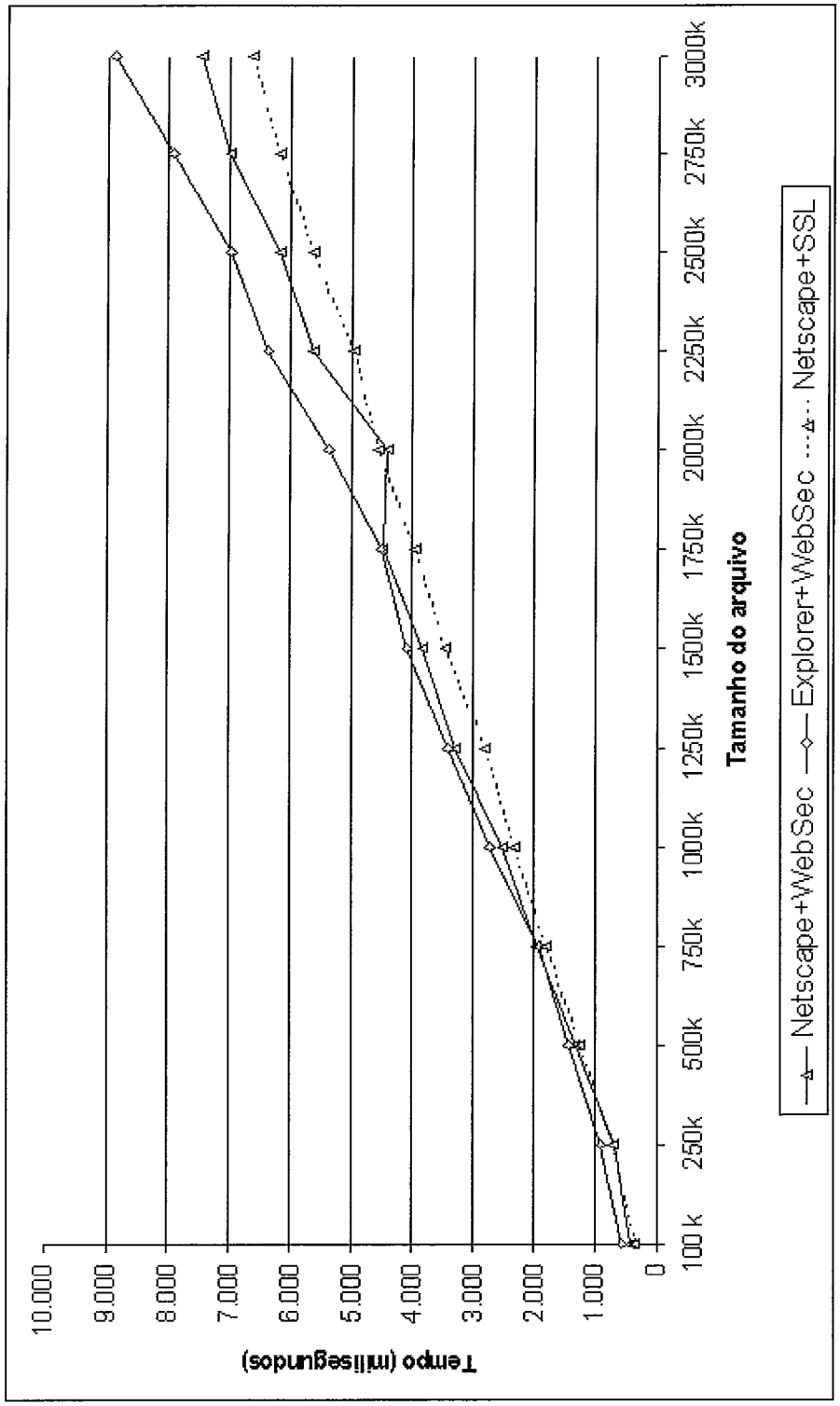


Figura 5.5: Tempos de transmissão de arquivos em diversas plataformas

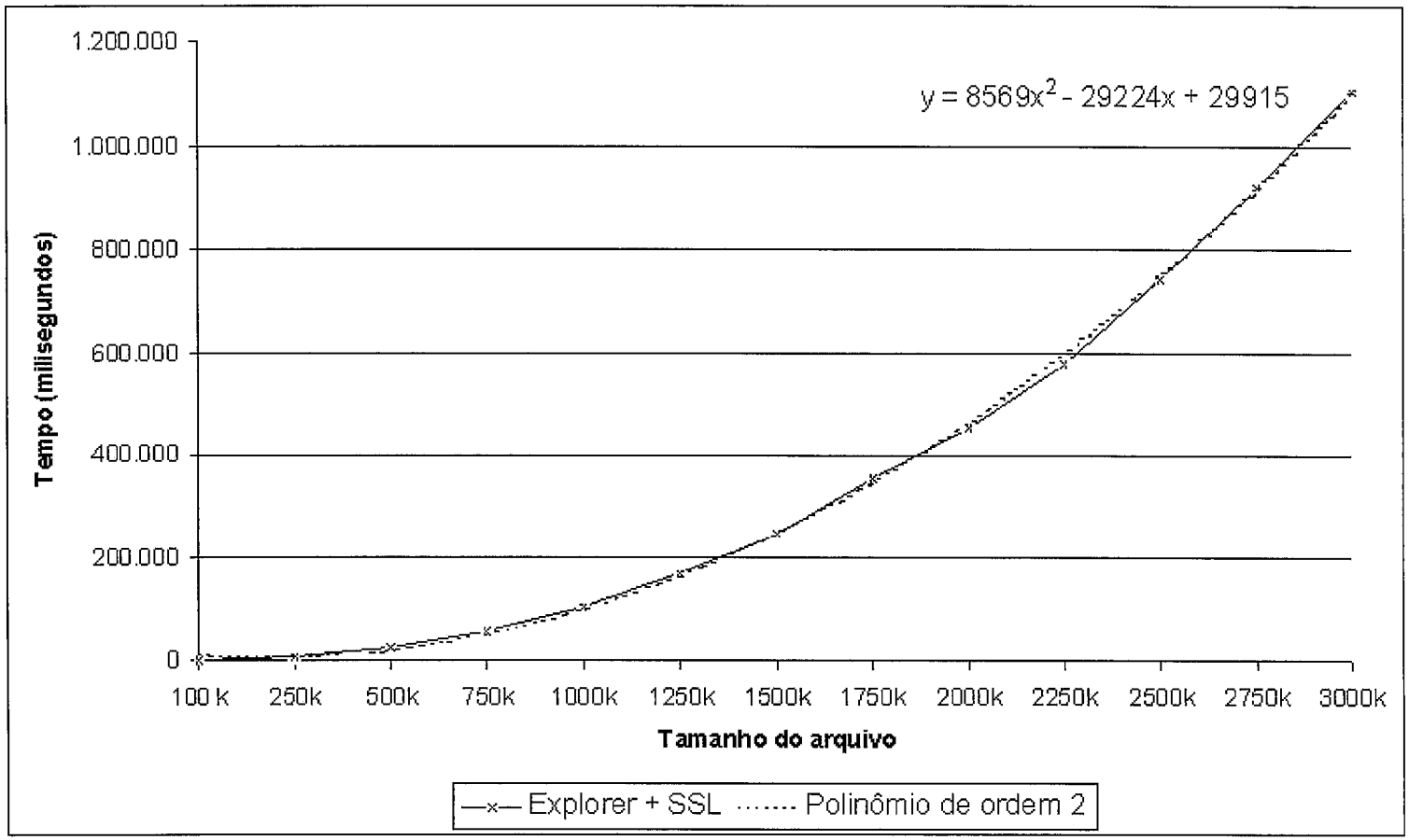


Figura 5.6: Tempo de transmissão de arquivo com Explorer+SSL e Função Quadrática.

Os conjuntos Netscape+WebSec, Netscape+SSL e Explorer+WebSec apresentam desempenho semelhante. O mesmo não acontece como Explorer+SSL, cujo rápido crescimento do tempo de transmissão torna o envio de arquivos grandes praticamente inviável. No entanto esta diferença parece devida, pelo menos em parte, a diferenças nas filosofias de implementação, e não necessariamente a ineficiência pura e simples.

O Explorer interrompe a recepção dos dados para exibir em sua janela o que já foi decifrado até o momento, enquanto o Netscape recebe todo o arquivo, só então exibindo os dados no monitor. Desta forma, do ponto de vista do usuário, o Explorer pode até parecer mais eficiente, pois assim que os primeiros blocos são recebidos, algum resultado, mesmo que parcial, é exibido na janela. Tal característica penaliza o desempenho na transmissão do arquivo como um todo de forma decisiva, como podemos observar nos dados exibidos nas Figura 5.6 e Tabela 5.1.

No caso do Netscape, o ponto de estrangulamento de eficiência parece ter-se movido do processo de criptografia para a transmissão dos dados em si e, como ambos os algoritmos não alteram o volume de dados transmitidos, ou seja, não compactam os dados nem aumentam a redundância de informação neles contidos, os tempos de transmissão pouco se alteram com ou sem a utilização do WebSec. O que não deixa de ser um bom resultado, tendo em vista as considerações anteriores.

5.6 Páginas para demonstração do sistema

Foi construído um pequeno conjunto de páginas HTML, com a finalidade de demonstrar a utilização do sistema. Estas páginas encontram-se localizadas no servidor do Colégio Técnico Universitário da Universidade Federal de Juiz de Fora e no Laboratório de Redes de Alta velocidade da COPPE/UFRJ nos endereços www.ctu.ufjf.br/wstst/ e websec.ravel.ufrj.br, respectivamente.

Sua aparência pode ser verificada na Figura 5.7.

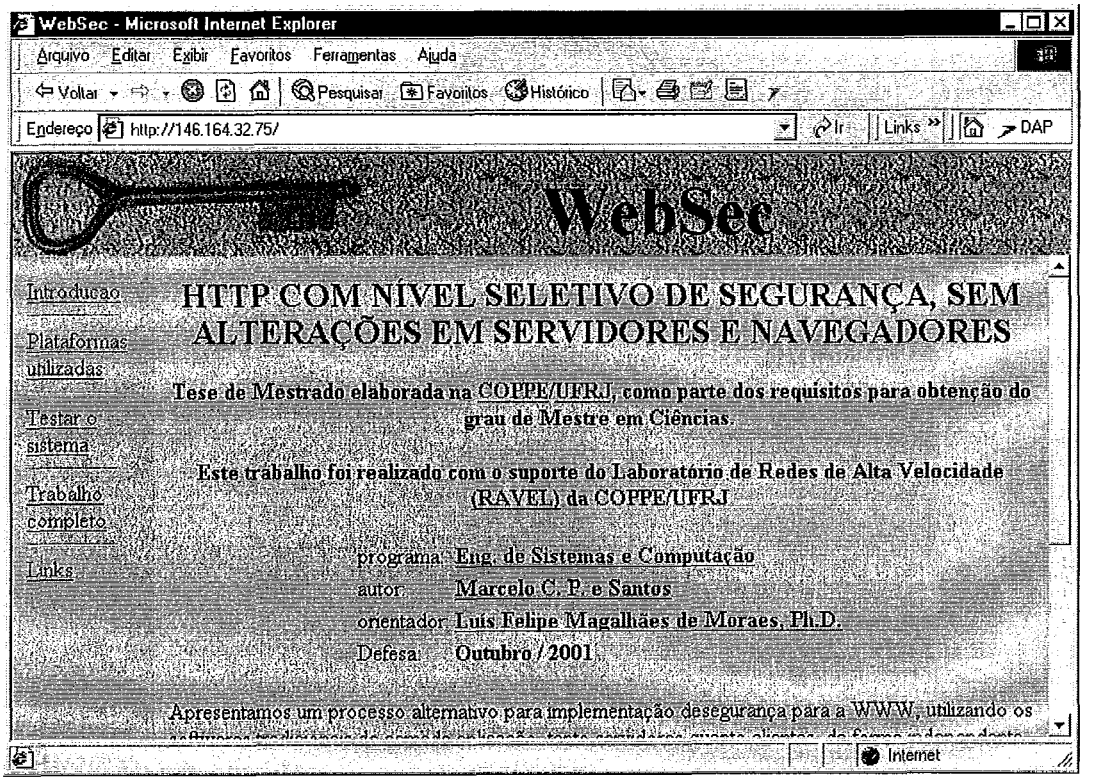


Figura 5.7: Página de demonstração do sistema WebSec.

No *link* testar o sistema, o usuário pode tentar a recepção de um pequeno arquivo por meio do WebSec sem possuir o *plugin*, quando o *browser* emitirá uma mensagem informando a impossibilidade de exibição dos dados solicitados. A seguir, a transferência e instalação do *plugim* pode ser feita e a transmissão segura pode ser observada pelo usuário.

6 Conclusões e Sugestões para Trabalhos Futuros

A segurança para a internet vem se mostrando como um dos maiores empecilhos à ampla utilização da grande rede mundial para fins comerciais, que poderia trazer ganhos de produtividade nem sequer imaginados pela comunidade mundial.

Certamente tal utilização ocorrerá, e será como um catalisador para o processo de globalização por que passa a humanidade atualmente, e tanta angústia tem trazido a países desenvolvidos, que dirá a países em desenvolvimento como o Brasil, que ainda não dispõem de uma estrutura de produção de mercadorias e, mais importante, tecnologias, para que possa fazer frente a potências atuais. Certamente este é um assunto a ser ainda amplamente discutido por economistas, sociólogos, antropologistas e líderes políticos de uma forma geral.

De qualquer forma, segurança é um assunto que tem de ser tratado e estudado. Até hoje, o conhecimento sobre segurança tem residido principalmente em uma sub-cultura *hacker*, que possui uma estrutura incrivelmente diferente, virtual e ao mesmo tempo real, com suas fronteiras baseadas não em coordenadas geográficas, mas em áreas de atuação, com seus valores próprios, baseados em prestígio e poder, mas um poder diferente do usual, proporcional à capacidade de utilização de recursos computacionais disponíveis na internet, que depende diretamente do nível de conhecimento dos protocolos e programas utilizados em máquinas que criam este universo paralelo que é a internet.

Consideramos que o meio acadêmico tem o dever de produzir conhecimento formal sobre o assunto, de forma a assumir o papel de principal depositário do conhecimento também nesta área, para que este saber possa ser utilizado de forma mais controlada, exclusivamente para o bem da comunidade mundial, e a melhoria da qualidade de vida do homem na terra.

A presente proposta desenvolve um método para troca de dados segura tatravés da Internet, por meio de um servidor independente que quando solicitado levanta arquivo junto a ao servidor HTTP tradicional e o passa ao cliente criptografado.

Junto ao cliente, um *plugin* recebe os dados e, os mostra ao usuário, após a decriptografia.

Desta forma o WebSec provê segurança independente de fornecedores e empresas, cujos interesses nem sempre são totalmente revelados. Com o sistema podemos, com custo de treinamento zero para o usuário da internet, e mínimo para o criador das páginas HTML, implementar a troca criptografada de dados, com tamanho de chave e algoritmo tão seguros quanto se queira. Tal possibilidade pode ser utilizada em comércio eletrônico ou em aplicações cujo nível de segurança necessário seja até mesmo maior, como na transmissão segura de segredos industriais, comerciais ou militares. Neste caso o nível de segurança pode ser aumentado, tornando indisponível o *plugin* através da internet, instalando-o manualmente em cada máquina cliente. O nível de segurança pode ser ainda maior, se a troca de chaves não for feita pela rede, e sim da única forma de transmissão totalmente segura: passada pessoalmente entre as partes interessadas na comunicação.

Podemos sugerir, a eventuais interessados em dar prosseguimento a esta linha de pesquisa, que realize comparações qualitativas e quantitativas entre o WebSec, como aqui foi definido, e idéia semelhante, implementada com um *applet* Java no lugar do *plugin*. Conforme discutido no trabalho, tal implementação tem vantagens e desvantagens em relação à adotada em nosso protótipo e, certamente, existirão aplicações onde o *applet* seja preferível ao *plugin*. Deve-se considerar, inclusive, a possibilidade de trabalharmos com um servidor único, que atenda a ambas as implementações do cliente.

De posse de um cliente implementado como uma classe Java, devemos efetuar testes, de forma a quantificar a queda de desempenho decorrente da menor eficiência da linguagem, da transmissão do *byte code* a cada vez que uma conexão for necessária, e outros fatores de comparação que certamente surgiriam no decorrer do trabalho.

Temos notícia de empresas comerciais que desenvolveram soluções neste sentido, notadamente bancos, implementando conexões seguras utilizando classes Java. Tais empresas relutam em divulgar informações sobre suas arquiteturas de segurança e, certamente, não gostariam de ver uma descrição detalhada das mesmas amplamente

divulgadas em um trabalho de cunho acadêmico, de livre circulação por definição. Daí outra motivação para que a Universidade desenvolva sua solução, para fins de pesquisa e desenvolvimento.

A fim de tornar o protótipo desenvolvido um produto comercial, é necessário que se implemente a exibição de mais tipos de dados no cliente, que atualmente só esta apto a exibir texto. Imagens e sons dariam maior versatilidade ao sistema, ampliando a gama de aplicações que poderia atender, caso viesse a ser utilizado comercialmente, como uma opção de arquitetura de segurança pelos criadores de páginas para a WWW.

7 Referencias

[1] Anônimo, *Segurança Máxima*, Campus, 2000.

[2] Atkinsons, Randall, "Security Architecture for I P", *IETF Request for Comment # 2401*, novembro de 1998.

[3] Bryant, Bill, *Designing an Authentication System: A Dialogue in Four Scenes*, http://web.mit.edu/kerberos/www/dialogue.html, 1998.

[4] Campos, Eduardo, Information Week, nº 26, 21 de julho de 1999.

[5] Comer, Douglas E., *Computer Networks and Internets*, Prentice Hall, 1997.

[6] Comer, Douglas E., *Internetworking with TCP/IP*, Prentice Hall, 1988.

[7] Dutton, Geoffrey, "Computing Operating Environments Coevolve", Relatório do IDC # 22347, http://www.idc.com/itforecaster/ itf20000808.stm;

[8] Ferguson, Daniel Senie, "Network Ingress Filtering:Defeating Denial of Service Attacks wich Employ IP Source Address Spoofing",*IETF Request for Comment # 2267*, janeiro 1998.

[9] *Final Report of the President's Concil on Year 2000 Conversion*, Governo dos EUA, 29 de março de 2000, http://www.y2k.gov/docs/lastrep3.htm.

[10] Freier, Alan O., Philip Karlton, Paul C Kocher; "The SSL protocol version 3.0", *Internet Draft*, novembro de 1996;

[11] Galyon, Eric, "C++ versus Java Performance", *Site da Colorado State University – Computer Science Department,* http://www.cs.colostate.edu/~cs154/PerfComp/

[12] Kaufman, Charlie, Radia Perlman, Mike Speciner; *Network Security*; Prentice Hall, 1995.

[13] Koops, Bert-Jaap, "Crypto Law Survey", *homepage,* outubro de 2000; http://cwis.kub.nl/~frw/people/koops/lawsurvey.htm;

[14] Kowalenko, Kathy, "Sun security expert analyzes Internet deniel-of-service attacks", *The Institute,* abril 200; volume 24, número 4.

[15] Negin, Michael , Thomas A. Chmielewski et al., "An Iris Biometric System for Public and Personal Use", *Computer*, fevereiro 2000; volume 33 número 2; pp. 70-75;

[16] Oliphant, Zan, *Programming Netscape Plug-Ins*, Sams.net Publishing, 1996.

[17] Pankanti, Sharah, Rud M. Bolle, Anil Jain; "Biometrics: The Future of Identification", *Computer,* fevereiro 2000, volume 33, número 2, pp.46-49;

[18] Phil, Basavaraj Patil, David B. Johnson e Charles Perkins, *Mobility Support in IPv6*; www.ietf.org\drafts\Draft-ietf-mobileip-ipv6-09.txt, 22 de outubro de 1999.

[19] Rescorla, Eric , Allan M. Schiffman;"The Secure Hypertext Transfer Protocol", *IETF Request for Comment # 2660*, agosto de 1999;

[20] Schneier, B., *Fast Software Encryption, Cambridge Security Workshop Proceedings,* Dezembro/1993, Springer-Verlag, 1994, pp. 191-204;

[21] Schneier, Bruce, "Fast Software Encryption", *Cambridge Security Workshop Proceedings,* dezembro 1993, Springer-Verlag, 1994, pp. 191-204;

[22] Seminerio, Maria, "FBI Warms 'Eletronic Pearl Harbor' Possible", *ZDNET*, 25 de março de 1998, http://www.scri.fsu.edu.

[23] Shafer, Steven L. e Alan R. Simon, *Network Security*; Academic Press; 1994.

[24] Silva, Carlos Roberto Damasceno, "Linux e Networking", *Revista do Linux*, Outubro de 2000;

[25] Stallings, William Stallings, "IP Security", *The Internet Protocol Journal*; março 2000; volume 3 número 1;

[26] Stevens, W. Richard; *Network Programming, Interprocess Cmmunications*; Prentice Hall, 1998.

[27] Tanenbaum, Andrew S., *Computer Networks,* 3ª ed., Prentice Hall, 1996.

[28] Wiener, Michael, Ronald L. Rivest, Whitfield Diffie, Bruce Schneier, et al., *Minimal Key Lenghts for Symetric Ciphers to Provide Adequate commercial Security,* http://www.counterpane.com/keylength.html, janeiro de 1996.

Anexo A - Código fonte do servidor WebSec

```
/*--------------------------------------
 * WebSecd
 * Recebe uma solicitacao de um cliente http,
 * recupera os dados solicitados, criptografa
 * e os encaminha ao solicitante.
 *--------------------------------------
 */

#include <sys/types.h>
#include <sys/signal.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "blowfish.h"
#include "crypt.h"

#ifndef INADDR_NONE
#define INADDR_NONE 0xffffffff
#endif

#define TAM_BLOCO    8  /* tamaho do bloco a ser
                         * criptografado BLOWFISH
                         * =64bits (8*8)
                         */
#define QLEN         5 /* tamanho máximo da fila de
                        * conexões
                        */
#define N           16

u_short htons();
u_long inet_addr();
int reaper();


/*------------------------------------------------------
 *  Valores originais de S e P para o Blowfish
 *------------------------------------------------------
 */
static const unsigned long ORIG_P[16 + 2] = {
        0x243F6A88L, 0x85A308D3L, 0x13198A2EL, 0x03707344L,
        0xA4093822L, 0x299F31D0L, 0x082EFA98L, 0xEC4E6C89L,
```

```
        0x452821E6L, 0x38D01377L, 0xBE5466CFL, 0x34E90C6CL,
        0xC0AC29B7L, 0xC97C50DDL, 0x3F84D5B5L, 0xB5470917L,
        0x9216D5D9L, 0x8979FB1BL
};

static const unsigned long ORIG_S[4][256] = {
    {   0xD1310BA6L, 0x98DFB5ACL, 0x2FFD72DBL, 0xD01ADFB7L,
        0xB8E1AFEDL, 0x6A267E96L, 0xBA7C9045L, 0xF12C7F99L,
        0x24A19947L, 0xB3916CF7L, 0x0801F2E2L, 0x858EFC16L,
        0x636920D8L, 0x71574E69L, 0xA458FEA3L, 0xF4933D7EL,
        0x0D95748FL, 0x728EB658L, 0x718BCD58L, 0x82154AEEL,
        0x7B54A41DL, 0xC25A59B5L, 0x9C30D539L, 0x2AF26013L,
        0xC5D1B023L, 0x286085F0L, 0xCA417918L, 0xB8DB38EFL,
        0x8E79DCB0L, 0x603A180EL, 0x6C9E0E8BL, 0xB01E8A3EL,
        0xD71577C1L, 0xBD314B27L, 0x78AF2FDAL, 0x55605C60L,
        0xE65525F3L, 0xAA55AB94L, 0x57489862L, 0x63E81440L,
        0x55CA396AL, 0x2AAB10B6L, 0xB4CC5C34L, 0x1141E8CEL,
        0xA15486AFL, 0x7C72E993L, 0xB3EE1411L, 0x636FBC2AL,
        0x2BA9C55DL, 0x741831F6L, 0xCE5C3E16L, 0x9B87931EL,
        0xAFD6BA33L, 0x6C24CF5CL, 0x7A325381L, 0x28958677L,
        0x3B8F4898L, 0x6B4BB9AFL, 0xC4BFE81BL, 0x66282193L,
        0x61D809CCL, 0xFB21A991L, 0x487CAC60L, 0x5DEC8032L,
        0xEF845D5DL, 0xE98575B1L, 0xDC262302L, 0xEB651B88L,
        0x23893E81L, 0xD396ACC5L, 0x0F6D6FF3L, 0x83F44239L,
        0x2E0B4482L, 0xA4842004L, 0x69C8F04AL, 0x9E1F9B5EL,
        0x21C66842L, 0xF6E96C9AL, 0x670C9C61L, 0xABD388F0L,
        0x6A51A0D2L, 0xD8542F68L, 0x960FA728L, 0xAB5133A3L,
        0x6EEF0B6CL, 0x137A3BE4L, 0xBA3BF050L, 0x7EFB2A98L,
        0xA1F1651DL, 0x39AF0176L, 0x66CA593EL, 0x82430E88L,
        0x8CEE8619L, 0x456F9FB4L, 0x7D84A5C3L, 0x3B8B5EBEL,
        0xE06F75D8L, 0x85C12073L, 0x401A449FL, 0x56C16AA6L,
        0x4ED3AA62L, 0x363F7706L, 0x1BFEDF72L, 0x429B023DL,
        0x37D0D724L, 0xD00A1248L, 0xDB0FEAD3L, 0x49F1C09BL,
        0x075372C9L, 0x80991B7BL, 0x25D479D8L, 0xF6E8DEF7L,
        0xE3FE501AL, 0xB6794C3BL, 0x976CE0BDL, 0x04C006BAL,
        0xC1A94FB6L, 0x409F60C4L, 0x5E5C9EC2L, 0x196A2463L,
        0x68FB6FAFL, 0x3E6C53B5L, 0x1339B2EBL, 0x3B52EC6FL,
        0x6DFC511FL, 0x9B30952CL, 0xCC814544L, 0xAF5EBD09L,
        0xBEE3D004L, 0xDE334AFDL, 0x660F2807L, 0x192E4BB3L,
        0xC0CBA857L, 0x45C8740FL, 0xD20B5F39L, 0xB9D3FBDBL,
        0x5579C0BDL, 0x1A60320AL, 0xD6A100C6L, 0x402C7279L,
        0x679F25FEL, 0xFB1FA3CCL, 0x8EA5E9F8L, 0xDB3222F8L,
        0x3C7516DFL, 0xFD616B15L, 0x2F501EC8L, 0xAD0552ABL,
        0x323DB5FAL, 0xFD238760L, 0x53317B48L, 0x3E00DF82L,
        0x9E5C57BBL, 0xCA6F8CA0L, 0x1A87562EL, 0xDF1769DBL,
        0xD542A8F6L, 0x287EFFC3L, 0xAC6732C6L, 0x8C4F5573L,
        0x695B27B0L, 0xBBCA58C8L, 0xE1FFA35DL, 0xB8F011A0L,
        0x10FA3D98L, 0xFD2183B8L, 0x4AFCB56CL, 0x2DD1D35BL,
        0x9A53E479L, 0xB6F84565L, 0xD28E49BCL, 0x4BFB9790L,
        0xE1DDF2DAL, 0xA4CB7E33L, 0x62FB1341L, 0xCEE4C6E8L,
        0xEF20CADAL, 0x36774C01L, 0xD07E9EFEL, 0x2BF11FB4L,
```

```
        0x95DBDA4DL, 0xAE909198L, 0xEAAD8E71L, 0x6B93D5A0L,
        0xD08ED1D0L, 0xAFC725E0L, 0x8E3C5B2FL, 0x8E7594B7L,
        0x8FF6E2FBL, 0xF2122B64L, 0x8888B812L, 0x900DF01CL,
        0x4FAD5EA0L, 0x688FC31CL, 0xD1CFF191L, 0xB3A8C1ADL,
        0x2F2F2218L, 0xBE0E1777L, 0xEA752DFEL, 0x8B021FA1L,
        0xE5A0CC0FL, 0xB56F74E8L, 0x18ACF3D6L, 0xCE89E299L,
        0xB4A84FE0L, 0xFD13E0B7L, 0x7CC43B81L, 0xD2ADA8D9L,
        0x165FA266L, 0x80957705L, 0x93CC7314L, 0x211A1477L,
        0xE6AD2065L, 0x77B5FA86L, 0xC75442F5L, 0xFB9D35CFL,
        0xEBCDAF0CL, 0x7B3E89A0L, 0xD6411BD3L, 0xAE1E7E49L,
        0x00250E2DL, 0x2071B35EL, 0x226800BBL, 0x57B8E0AFL,
        0x2464369BL, 0xF009B91EL, 0x5563911DL, 0x59DFA6AAL,
        0x78C14389L, 0xD95A537FL, 0x207D5BA2L, 0x02E5B9C5L,
        0x83260376L, 0x6295CFA9L, 0x11C81968L, 0x4E734A41L,
        0xB3472DCAL, 0x7B14A94AL, 0x1B510052L, 0x9A532915L,
        0xD60F573FL, 0xBC9BC6E4L, 0x2B60A476L, 0x81E67400L,
        0x08BA6FB5L, 0x571BE91FL, 0xF296EC6BL, 0x2A0DD915L,
        0xB6636521L, 0xE7B9F9B6L, 0xFF34052EL, 0xC5855664L,
        0x53B02D5DL, 0xA99F8FA1L, 0x08BA4799L, 0x6E85076AL
},
    {   0x4B7A70E9L, 0xB5B32944L, 0xDB75092EL, 0xC4192623L,
        0xAD6EA6B0L, 0x49A7DF7DL, 0x9CEE60B8L, 0x8FEDB266L,
        0xECAA8C71L, 0x699A17FFL, 0x5664526CL, 0xC2B19EE1L,
        0x193602A5L, 0x75094C29L, 0xA0591340L, 0xE4183A3EL,
        0x3F54989AL, 0x5B429D65L, 0x6B8FE4D6L, 0x99F73FD6L,
        0xA1D29C07L, 0xEFE830F5L, 0x4D2D38E6L, 0xF0255DC1L,
        0x4CDD2086L, 0x8470EB26L, 0x6382E9C6L, 0x021ECC5EL,
        0x09686B3FL, 0x3EBAEFC9L, 0x3C971814L, 0x6B6A70A1L,
        0x687F3584L, 0x52A0E286L, 0xB79C5305L, 0xAA500737L,
        0x3E07841CL, 0x7FDEAE5CL, 0x8E7D44ECL, 0x5716F2B8L,
        0xB03ADA37L, 0xF0500C0DL, 0xF01C1F04L, 0x0200B3FFL,
        0xAE0CF51AL, 0x3CB574B2L, 0x25837A58L, 0xDC0921BDL,
        0xD19113F9L, 0x7CA92FF6L, 0x94324773L, 0x22F54701L,
        0x3AE5E581L, 0x37C2DADCL, 0xC8B57634L, 0x9AF3DDA7L,
        0xA9446146L, 0x0FD0030EL, 0xECC8C73EL, 0xA4751E41L,
        0xE238CD99L, 0x3BEA0E2FL, 0x3280BBA1L, 0x183EB331L,
        0x4E548B38L, 0x4F6DB908L, 0x6F420D03L, 0xF60A04BFL,
        0x2CB81290L, 0x24977C79L, 0x5679B072L, 0xBCAF89AFL,
        0xDE9A771FL, 0xD9930810L, 0xB38BAE12L, 0xDCCF3F2EL,
        0x5512721FL, 0x2E6B7124L, 0x501ADDE6L, 0x9F84CD87L,
        0x7A584718L, 0x7408DA17L, 0xBC9F9ABCL, 0xE94B7D8CL,
        0xEC7AEC3AL, 0xDB851DFAL, 0x63094366L, 0xC464C3D2L,
        0xEF1C1847L, 0x3215D908L, 0xDD433B37L, 0x24C2BA16L,
        0x12A14D43L, 0x2A65C451L, 0x50940002L, 0x133AE4DDL,
        0x71DFF89EL, 0x10314E55L, 0x81AC77D6L, 0x5F11199BL,
        0x043556F1L, 0xD7A3C76BL, 0x3C11183BL, 0x5924A509L,
        0xF28FE6EDL, 0x97F1FBFAL, 0x9EBABF2CL, 0x1E153C6EL,
        0x86E34570L, 0xEAE96FB1L, 0x860E5E0AL, 0x5A3E2AB3L,
        0x771FE71CL, 0x4E3D06FAL, 0x2965DCB9L, 0x99E71D0FL,
        0x803E89D6L, 0x5266C825L, 0x2E4CC978L, 0x9C10B36AL,
        0xC6150EBAL, 0x94E2EA78L, 0xA5FC3C53L, 0x1E0A2DF4L,
```

```
        0xF2F74EA7L, 0x361D2B3DL, 0x1939260FL, 0x19C27960L,
        0x5223A708L, 0xF71312B6L, 0xEBADFE6EL, 0xEAC31F66L,
        0xE3BC4595L, 0xA67BC883L, 0xB17F37D1L, 0x018CFF28L,
        0xC332DDEFL, 0xBE6C5AA5L, 0x65582185L, 0x68AB9802L,
        0xEECEA50FL, 0xDB2F953BL, 0x2AEF7DADL, 0x5B6E2F84L,
        0x1521B628L, 0x29076170L, 0xECDD4775L, 0x619F1510L,
        0x13CCA830L, 0xEB61BD96L, 0x0334FE1EL, 0xAA0363CFL,
        0xB5735C90L, 0x4C70A239L, 0xD59E9E0BL, 0xCBAADE14L,
        0xEECC86BCL, 0x60622CA7L, 0x9CAB5CABL, 0xB2F3846EL,
        0x648B1EAFL, 0x19BDF0CAL, 0xA02369B9L, 0x655ABB50L,
        0x40685A32L, 0x3C2AB4B3L, 0x319EE9D5L, 0xC021B8F7L,
        0x9B540B19L, 0x875FA099L, 0x95F7997EL, 0x623D7DA8L,
        0xF837889AL, 0x97E32D77L, 0x11ED935FL, 0x16681281L,
        0x0E358829L, 0xC7E61FD6L, 0x96DEDFA1L, 0x7858BA99L,
        0x57F584A5L, 0x1B227263L, 0x9B83C3FFL, 0x1AC24696L,
        0xCDB30AEBL, 0x532E3054L, 0x8FD948E4L, 0x6DBC3128L,
        0x58EBF2EFL, 0x34C6FFEAL, 0xFE28ED61L, 0xEE7C3C73L,
        0x5D4A14D9L, 0xE864B7E3L, 0x42105D14L, 0x203E13E0L,
        0x45EEE2B6L, 0xA3AAABEAL, 0xDB6C4F15L, 0xFACB4FD0L,
        0xC742F442L, 0xEF6ABBB5L, 0x654F3B1DL, 0x41CD2105L,
        0xD81E799EL, 0x86854DC7L, 0xE44B476AL, 0x3D816250L,
        0xCF62A1F2L, 0x5B8D2646L, 0xFC8883A0L, 0xC1C7B6A3L,
        0x7F1524C3L, 0x69CB7492L, 0x47848A0BL, 0x5692B285L,
        0x095BBF00L, 0xAD19489DL, 0x1462B174L, 0x23820E00L,
        0x58428D2AL, 0x0C55F5EAL, 0x1DADF43EL, 0x233F7061L,
        0x3372F092L, 0x8D937E41L, 0xD65FECF1L, 0x6C223BDBL,
        0x7CDE3759L, 0xCBEE7460L, 0x4085F2A7L, 0xCE77326EL,
        0xA6078084L, 0x19F8509EL, 0xE8EFD855L, 0x61D99735L,
        0xA969A7AAL, 0xC50C06C2L, 0x5A04ABFCL, 0x800BCADCL,
        0x9E447A2EL, 0xC3453484L, 0xFDD56705L, 0x0E1E9EC9L,
        0xDB73DBD3L, 0x105588CDL, 0x675FDA79L, 0xE3674340L,
        0xC5C43465L, 0x713E38D8L, 0x3D28F89EL, 0xF16DFF20L,
        0x153E21E7L, 0x8FB03D4AL, 0xE6E39F2BL, 0xDB83ADF7L
},
    {   0xE93D5A68L, 0x948140F7L, 0xF64C261CL, 0x94692934L,
        0x411520F7L, 0x7602D4F7L, 0xBCF46B2EL, 0xD4A20068L,
        0xD4082471L, 0x3320F46AL, 0x43B7D4B7L, 0x500061AFL,
        0x1E39F62EL, 0x97244546L, 0x14214F74L, 0xBF8B8840L,
        0x4D95FC1DL, 0x96B591AFL, 0x70F4DDD3L, 0x66A02F45L,
        0xBFBC09ECL, 0x03BD9785L, 0x7FAC6DD0L, 0x31CB8504L,
        0x96EB27B3L, 0x55FD3941L, 0xDA2547E6L, 0xABCA0A9AL,
        0x28507825L, 0x530429F4L, 0x0A2C86DAL, 0xE9B66DFBL,
        0x68DC1462L, 0xD7486900L, 0x680EC0A4L, 0x27A18DEEL,
        0x4F3FFEA2L, 0xE887AD8CL, 0xB58CE006L, 0x7AF4D6B6L,
        0xAACE1E7CL, 0xD3375FECL, 0xCE78A399L, 0x406B2A42L,
        0x20FE9E35L, 0xD9F385B9L, 0xEE39D7ABL, 0x3B124E8BL,
        0x1DC9FAF7L, 0x4B6D1856L, 0x26A36631L, 0xEAE397B2L,
        0x3A6EFA74L, 0xDD5B4332L, 0x6841E7F7L, 0xCA7820FBL,
        0xFB0AF54EL, 0xD8FEB397L, 0x454056ACL, 0xBA489527L,
        0x55533A3AL, 0x20838D87L, 0xFE6BA9B7L, 0xD096954BL,
        0x55A867BCL, 0xA1159A58L, 0xCCA92963L, 0x99E1DB33L,
```

```
        0xA62A4A56L, 0x3F3125F9L, 0x5EF47E1CL, 0x9029317CL,
        0xFDF8E802L, 0x04272F70L, 0x80BB155CL, 0x05282CE3L,
        0x95C11548L, 0xE4C66D22L, 0x48C1133FL, 0xC70F86DCL,
        0x07F9C9EEL, 0x41041F0FL, 0x404779A4L, 0x5D886E17L,
        0x325F51EBL, 0xD59BC0D1L, 0xF2BCC18FL, 0x41113564L,
        0x257B7834L, 0x602A9C60L, 0xDFF8E8A3L, 0x1F636C1BL,
        0x0E12B4C2L, 0x02E1329EL, 0xAF664FD1L, 0xCAD18115L,
        0x6B2395E0L, 0x333E92E1L, 0x3B240B62L, 0xEEBEB922L,
        0x85B2A20EL, 0xE6BA0D99L, 0xDE720C8CL, 0x2DA2F728L,
        0xD0127845L, 0x95B794FDL, 0x647D0862L, 0xE7CCF5F0L,
        0x5449A36FL, 0x877D48FAL, 0xC39DFD27L, 0xF33E8D1EL,
        0x0A476341L, 0x992EFF74L, 0x3A6F6EABL, 0xF4F8FD37L,
        0xA812DC60L, 0xA1EBDDF8L, 0x991BE14CL, 0xDB6E6B0DL,
        0xC67B5510L, 0x6D672C37L, 0x2765D43BL, 0xDCD0E804L,
        0xF1290DC7L, 0xCC00FFA3L, 0xB5390F92L, 0x690FED0BL,
        0x667B9FFBL, 0xCEDB7D9CL, 0xA091CF0BL, 0xD9155EA3L,
        0xBB132F88L, 0x515BAD24L, 0x7B9479BFL, 0x763BD6EBL,
        0x37392EB3L, 0xCC115979L, 0x8026E297L, 0xF42E312DL,
        0x6842ADA7L, 0xC66A2B3BL, 0x12754CCCL, 0x782EF11CL,
        0x6A124237L, 0xB79251E7L, 0x06A1BBE6L, 0x4BFB6350L,
        0x1A6B1018L, 0x11CAEDFAL, 0x3D25BDD8L, 0xE2E1C3C9L,
        0x44421659L, 0x0A121386L, 0xD90CEC6EL, 0xD5ABEA2AL,
        0x64AF674EL, 0xDA86A85FL, 0xBEBFE988L, 0x64E4C3FEL,
        0x9DBC8057L, 0xF0F7C086L, 0x60787BF8L, 0x6003604DL,
        0xD1FD8346L, 0xF6381FB0L, 0x7745AE04L, 0xD736FCCCL,
        0x83426B33L, 0xF01EAB71L, 0xB0804187L, 0x3C005E5FL,
        0x77A057BEL, 0xBDE8AE24L, 0x55464299L, 0xBF582E61L,
        0x4E58F48FL, 0xF2DDFDA2L, 0xF474EF38L, 0x8789BDC2L,
        0x5366F9C3L, 0xC8B38E74L, 0xB475F255L, 0x46FCD9B9L,
        0x7AEB2661L, 0x8B1DDF84L, 0x846A0E79L, 0x915F95E2L,
        0x466E598EL, 0x20B45770L, 0x8CD55591L, 0xC902DE4CL,
        0xB90BACE1L, 0xBB8205D0L, 0x11A86248L, 0x7574A99EL,
        0xB77F19B6L, 0xE0A9DC09L, 0x662D09A1L, 0xC4324633L,
        0xE85A1F02L, 0x09F0BE8CL, 0x4A99A025L, 0x1D6EFE10L,
        0x1AB93D1DL, 0x0BA5A4DFL, 0xA186F20FL, 0x2868F169L,
        0xDCB7DA83L, 0x573906FEL, 0xA1E2CE9BL, 0x4FCD7F52L,
        0x50115E01L, 0xA70683FAL, 0xA002B5C4L, 0x0DE6D027L,
        0x9AF88C27L, 0x773F8641L, 0xC3604C06L, 0x61A806B5L,
        0xF0177A28L, 0xC0F586E0L, 0x006058AAL, 0x30DC7D62L,
        0x11E69ED7L, 0x2338EA63L, 0x53C2DD94L, 0xC2C21634L,
        0xBBCBEE56L, 0x90BCB6DEL, 0xEBFC7DA1L, 0xCE591D76L,
        0x6F05E409L, 0x4B7C0188L, 0x39720A3DL, 0x7C927C24L,
        0x86E3725FL, 0x724D9DB9L, 0x1AC15BB4L, 0xD39EB8FCL,
        0xED545578L, 0x08FCA5B5L, 0xD83D7CD3L, 0x4DAD0FC4L,
        0x1E50EF5EL, 0xB161E6F8L, 0xA28514D9L, 0x6C51133CL,
        0x6FD5C7E7L, 0x56E14EC4L, 0x362ABFCEL, 0xDDC6C837L,
        0xD79A3234L, 0x92638212L, 0x670EFA8EL, 0x406000E0L
},
    {   0x3A39CE37L, 0xD3FAF5CFL, 0xABC27737L, 0x5AC52D1BL,
        0x5CB0679EL, 0x4FA33742L, 0xD3822740L, 0x99BC9BBEL,
        0xD5118E9DL, 0xBF0F7315L, 0xD62D1C7EL, 0xC700C47BL,
```

```
0xB78C1B6BL, 0x21A19045L, 0xB26EB1BEL, 0x6A366EB4L,
0x5748AB2FL, 0xBC946E79L, 0xC6A376D2L, 0x6549C2C8L,
0x530FF8EEL, 0x468DDE7DL, 0xD5730A1DL, 0x4CD04DC6L,
0x2939BBDBL, 0xA9BA4650L, 0xAC9526E8L, 0xBE5EE304L,
0xA1FAD5F0L, 0x6A2D519AL, 0x63EF8CE2L, 0x9A86EE22L,
0xC089C2B8L, 0x43242EF6L, 0xA51E03AAL, 0x9CF2D0A4L,
0x83C061BAL, 0x9BE96A4DL, 0x8FE51550L, 0xBA645BD6L,
0x2826A2F9L, 0xA73A3AE1L, 0x4BA99586L, 0xEF5562E9L,
0xC72FEFD3L, 0xF752F7DAL, 0x3F046F69L, 0x77FA0A59L,
0x80E4A915L, 0x87B08601L, 0x9B09E6ADL, 0x3B3EE593L,
0xE990FD5AL, 0x9E34D797L, 0x2CF0B7D9L, 0x022B8B51L,
0x96D5AC3AL, 0x017DA67DL, 0xD1CF3ED6L, 0x7C7D2D28L,
0x1F9F25CFL, 0xADF2B89BL, 0x5AD6B472L, 0x5A88F54CL,
0xE029AC71L, 0xE019A5E6L, 0x47B0ACFDL, 0xED93FA9BL,
0xE8D3C48DL, 0x283B57CCL, 0xF8D56629L, 0x79132E28L,
0x785F0191L, 0xED756055L, 0xF7960E44L, 0xE3D35E8CL,
0x15056DD4L, 0x88F46DBAL, 0x03A16125L, 0x0564F0BDL,
0xC3EB9E15L, 0x3C9057A2L, 0x97271AECL, 0xA93A072AL,
0x1B3F6D9BL, 0x1E6321F5L, 0xF59C66FBL, 0x26DCF319L,
0x7533D928L, 0xB155FDF5L, 0x03563482L, 0x8ABA3CBBL,
0x28517711L, 0xC20AD9F8L, 0xABCC5167L, 0xCCAD925FL,
0x4DE81751L, 0x3830DC8EL, 0x379D5862L, 0x9320F991L,
0xEA7A90C2L, 0xFB3E7BCEL, 0x5121CE64L, 0x774FBE32L,
0xA8B6E37EL, 0xC3293D46L, 0x48DE5369L, 0x6413E680L,
0xA2AE0810L, 0xDD6DB224L, 0x69852DFDL, 0x09072166L,
0xB39A460AL, 0x6445C0DDL, 0x586CDECFL, 0x1C20C8AEL,
0x5BBEF7DDL, 0x1B588D40L, 0xCCD2017FL, 0x6BB4E3BBL,
0xDDA26A7EL, 0x3A59FF45L, 0x3E350A44L, 0xBCB4CDD5L,
0x72EACEA8L, 0xFA6484BBL, 0x8D6612AEL, 0xBF3C6F47L,
0xD29BE463L, 0x542F5D9EL, 0xAEC2771BL, 0xF64E6370L,
0x740E0D8DL, 0xE75B1357L, 0xF8721671L, 0xAF537D5DL,
0x4040CB08L, 0x4EB4E2CCL, 0x34D2466AL, 0x0115AF84L,
0xE1B00428L, 0x95983A1DL, 0x06B89FB4L, 0xCE6EA048L,
0x6F3F3B82L, 0x3520AB82L, 0x011A1D4BL, 0x277227F8L,
0x611560B1L, 0xE7933FDCL, 0xBB3A792BL, 0x344525BDL,
0xA08839E1L, 0x51CE794BL, 0x2F32C9B7L, 0xA01FBAC9L,
0xE01CC87EL, 0xBCC7D1F6L, 0xCF0111C3L, 0xA1E8AAC7L,
0x1A908749L, 0xD44FBD9AL, 0xD0DADECBL, 0xD50ADA38L,
0x0339C32AL, 0xC6913667L, 0x8DF9317CL, 0xE0B12B4FL,
0xF79E59B7L, 0x43F5BB3AL, 0xF2D519FFL, 0x27D9459CL,
0xBF97222CL, 0x15E6FC2AL, 0x0F91FC71L, 0x9B941525L,
0xFAE59361L, 0xCEB69CEBL, 0xC2A86459L, 0x12BAA8D1L,
0xB6C1075EL, 0xE3056A0CL, 0x10D25065L, 0xCB03A442L,
0xE0EC6E0EL, 0x1698DB3BL, 0x4C98A0BEL, 0x3278E964L,
0x9F1F9532L, 0xE0D392DFL, 0xD3A0342BL, 0x8971F21EL,
0x1B0A7441L, 0x4BA3348CL, 0xC5BE7120L, 0xC37632D8L,
0xDF359F8DL, 0x9B992F2EL, 0xE60B6F47L, 0x0FE3F11DL,
0xE54CDA54L, 0x1EDAD891L, 0xCE6279CFL, 0xCD3E7E6FL,
0x1618B166L, 0xFD2C1D05L, 0x848FD2C5L, 0xF6FB2299L,
0xF523F357L, 0xA6327623L, 0x93A83531L, 0x56CCCD02L,
0xACF08162L, 0x5A75EBB5L, 0x6E163697L, 0x88D273CCL,
```

```
        0xDE966292L, 0x81B949D0L, 0x4C50901BL, 0x71C65614L,
        0xE6C6C7BDL, 0x327A140AL, 0x45E1D006L, 0xC3F27B9AL,
        0xC9AA53FDL, 0x62A80F00L, 0xBB25BFE2L, 0x35BDD2F6L,
        0x71126905L, 0xB2040222L, 0xB6CBCF7CL, 0xCD769C2BL,
        0x53113EC0L, 0x1640E3D3L, 0x38ABBD60L, 0x2547ADF0L,
        0xBA38209CL, 0xF746CE76L, 0x77AFA1C5L, 0x20756060L,
        0x85CBFE4EL, 0x8AE88DD8L, 0x7AAAF9B0L, 0x4CF9AA7EL,
        0x1948C25CL, 0x02FB8A8CL, 0x01C36AE4L, 0xD6EBE1F9L,
        0x90D4F869L, 0xA65CDEA0L, 0x3F09252DL, 0xC208E69FL,
        0xB74E6132L, 0xCE77E25BL, 0x578FDFE3L, 0x3AC372E6L
}
};

/*------------------------------------------------
 *  main
 *------------------------------------------------
 */
unsigned char buf[TAM_BLOCO+1];  /* BUFFER principal  */

/*----------------------------------------
 *   Servidor concorrente para o WebSec
 *   Atende na porta 500
 *----------------------------------------
 */
int
main()
{
   char     *host = "localhost"; /* nome do servidor  */
   char     *retorno;

   struct   sockaddr_in fsin;    /* endereço do cliente
   int      alen;                /* comprimento do endereço
                                  * de cliente  */
   int      msock;      /* soquete do servidor master  */
   int      ssock;      /* soquete do servidor escravo  */

   msock = passivesock500("tcp", QLEN);

   (void) signal (SIGCHLD, reaper);

   while (1) {
      alen =sizeof(fsin);
      ssock=accept(msock, (struct sockaddr *)&fsin, &alen);
      if (ssock < 0) {
         printf("Erro na aceitação da conexão \n");
         exit(2);
         }
      switch (fork()) {
         case 0:    /*  filho   */
            (void) close(msock);
            exit(WebSec(ssock));
```

```
            default:    /*  Pai  */
               (void) close(ssock);
               break;

            case -1:
               printf("Falha na bifurcação dos processos\n");
               exit(3);
            }
         }

   exit(0);
}

/*-----------------------------------------------------
 *  WebSecd - módulo principal.
 *  1- Recupera dados e passa, criptografado ao cliente
 *  2- Recebe dados cripografados do cliente e encaminha
 *     ao servidor, apos decifrar.
 *-----------------------------------------------------
 */
int
WebSec(cli)
int cli;  // handler para o cliente
{
   int ser;       // handler para o servidor
   int n=1, m=1;  // Numero de caracteres lidos por um read

   BLOWFISH_CTX ctxCif,ctxDec;   // Estrutura de controle
                                 // do Blowfish para CIFrar
                                 //e DECifrar
   unsigned long E=0,D=0;  // 32 bits a esquerda e a
                           // direita do bloco de 64 a
                           // criptografar

   int aux,aux1;

   unsigned char GET[1024]="";  // conterá a linha do GET
                                // de uma mensagem ao
                                // servidor

   int iniCifrado=0,fimCifrado=0,fimGET=0;
                            // controles para passagem de
                            // mensagens ao servidor

   unsigned char cabecalho[32][128];
                        // contera o cabeçalho de dados
                        // destinados ao cliente

   int fimCabecalho=0; //  para passar dados aos clientes
```

```
int linha=0, coluna=0; // nr de linhas/letras de
                       //cabecalho recebidas do servidor

int aler,lido,aenviar,enviado;
                      // tamanho do arquivo a transmitir

FILE * fd;  //  descritor de arquivo para o certificado

char chaveCifrada[128]="",chave[128]="";
char chaveSecreta[128]="";

CRYPT_ENVELOPE rsaEnvelope;
CRYPT_CONTEXT rsaContext;

n = read(cli, buf, TAM_BLOCO);

/*
 *  Se o comando HTTP nao for um SGET, devolve-lo ao
 *  cliente e sair
 */
if(memcmp(buf,"SGET",4)) {
   strcpy(GET,"Content-Type:application/x-websec\n");
   strcat(GET,buf);
   while (n==TAM_BLOCO) {
      n = read(cli, buf, TAM_BLOCO);
      strcat(GET,buf);
   }
   write(cli,GET,strlen(GET));
   return;
}

/*
 * Se for SGET
 */
//  copia o primeiro bloco retirando o "S" do "SGET" e
//  termina de receber a requisicao
for(aux=1;aux<strlen(buf);aux++) GET[aux-1]=buf[aux];
n=TAM_BLOCO;
while (n==TAM_BLOCO) {
   n = read(cli, buf, TAM_BLOCO);
   strcat(GET,buf);
}

//  Transmite ao cliente:
//     1-Se exige certificado (S ou N)
//     2-Tamanho da chave (3 caracteres)
//     3-Certificado (lido de /home/httpd/html/snakeoil-
//       rsa.crt

write(cli,"N128",4);
```

```
// Envia certificado ao cliente
fd = fopen("/home/httpd/html/snakeoil-rsa.crt","r");
n = fread(buf,64,1,fd);
while (n>0) {
   write(cli,buf,n);
   n = fread(buf,64,1,fd);
}

/*
 *  Recebe e decifra a chave simétrica
 */
//  Cria envelope para decifrar a chave simétrica
cryptCreateEnvelope( &rsaEnvelope,
                     CRYPT_UNUSED,
                     CRYPT_FORMAT_AUTO);
//  Cria contexto de criptografia adequado (RSA)
cryptCreateContext( &rsaContext,
                    CRYPT_UNUSED,
                    CRYPT_ALGO_RSA);
//  Le chave privada do servidor do disco
fd = fopen("/home/httpd/html/server.key","r");
n = fread(buf,64,1,fd);
while (n>0) {
   strcat(chaveSecreta,buf);
   n = fread(buf,64,1,fd);
}
//  Inclui chave secreta lida no contexto
cryptSetAttributeString( rsaContext,
                         CRYPT_CTXINFO_KEY,
                         chaveSecreta,
                         strlen(chaveSecreta));
//  Inclui contexto com chave secreta no envelope
cryptSetAttribute( rsaEnvelope,
                   CRYPT_ENVINFO_SESSIONKEY,
                   rsaContext);
cryptDestroyContext( rsaContext );
//  recebe chave cifrada com RSA
n = 0;
while( n < 128 ) n += read(cli, chaveCifrada, 128);
//  Inclui chave simétrica a decifrar no envelope
cryptSetAttribute( rsaEnvelope,
                   CRYPT_ENVINFO_DATASIZE,
                   strlen(chaveCifrada));
cryptPushData( rsaEnvelope,
               chaveCifrada,
               strlen(chaveCifrada), NULL);
cryptPushData( rsaEnvelope,
               NULL,
               0,
               NULL );
```

```
   //  Retira chave simétrica decifrada
   cryptPopData( rsaEnvelope,
                 chave,
                 128,
                 NULL);
   //  Destroi envelope
   cryptDestroyEnvelope( rsaEnvelope );

   switch (fork()) {

/*=======================================================*/
      case 0: /*filho-le do cliente e passa ao servidor */

         /*  Inicializa Blowfish com chave de 128 bits  */
         Blowfish_Init (&ctxDec, chave, 16);

         // Inclui dados já recebidos retirando
         // o 'S' do 'SGET'
         for(aux=0;aux<strlen(buf);aux++)
            buf[aux]=buf[aux+1];
         strcpy(GET,buf);  // copia dados recebidos
                           // para var GET

         //  le restante da solicitacao do cliente
         while (n>0) {

            n = read(cli, buf, TAM_BLOCO);

            //  conecta ao servidor tradicional
            ser = connectsock( "localhost", "http", "tcp");

            strcat(GET,buf);  // copia dados recebidos
                              // para var GET

            // Determina inicio e fim da area de variaveis
            // que necessitam ser decifradas.
            while(!fimGET) {
               m = read(cli, buf, TAM_BLOCO);
               strcat(GET,buf);

if(iniCifrado==0)for(aux=0;aux<strlen(GET);aux++)
if(GET[aux]=='=') iniCifrado=aux+1;
               if((iniCifrado>0)&&(fimCifrado==0)) {
                  // comecou cifrado, tem de detectar fim
                  for(aux=iniCifrado;
                      aux<(strlen(GET));
                      aux++) {
                     if ((GET[aux  ]=='/')&&
                         (GET[aux+1]==' ')&&
                         (GET[aux+2]=='H')&&
                         (GET[aux+3]=='T')) {
```

```
                        fimCifrado=aux;
                    }
                }
            }
            for(aux=0;aux<m;aux++)
                if(buf[aux]==(char)10)   fimGET=1;
        }

        // Decifra area de variaveis
        aux = iniCifrado;
        while(aux<fimCifrado) {

            //  Carrega dados recebidos em E e D
                        D =(int)GET[aux+7];
            D=D<<8;     D+=(int)GET[aux+6];
            D=D<<8;     D+=(int)GET[aux+5];
            D=D<<8;     D+=(int)GET[aux+4];
                        E =(int)GET[aux+3];
            E=E<<8;     E+=(int)GET[aux+2];
            E=E<<8;     E+=(int)GET[aux+1];
            E=E<<8;     E+=(int)GET[aux+0];

           //  encripta bloco de 64 bits
           Blowfish_Decrypt(&ctxDec, &E, &D);

           //  retorna com dados decifados para o buffer
                        GET[aux+0]=(char) E;
            E=E>>8;     GET[aux+1]=(char) E;
            E=E>>8;     GET[aux+2]=(char) E;
            E=E>>8;     GET[aux+3]=(char) E;
                        GET[aux+4]=(char) D;
            D=D>>8;     GET[aux+5]=(char) D;
            D=D>>8;     GET[aux+6]=(char) D;
            D=D>>8;     GET[aux+7]=(char) D;

            aux+=8;
        }

        // Envia dados decifrados ao servidor
        // tradicional
        write(ser,GET,strlen(GET));

    }
    break;


/*=======================================================*/
    default:  /* Pai-le do servidor e passa ao cliente */

        /*  Inicializa Blowfish com chave de 128 bits  */
        Blowfish_Init (&ctxCif, chave, 16);
```

```
//   conecta ao servidor tradicional
ser = connectsock( "localhost", "http", "tcp");


while (n>0) {

   //  le cabecalho
   fimCabecalho=0;
   linha=0;
   sprintf(cabecalho[0],"");  //limpa primeira
                              //linha de cabecalho

   while(!fimCabecalho) {

      n = read(ser, buf, TAM_BLOCO);

      for( aux=0; aux<n; aux++) {
         cabecalho[linha][coluna]=buf[aux];
         coluna++;
         if( buf[aux] == (char)10 ) {
            if( strlen(cabecalho[linha]) < 3 )
               fimCabecalho=1;  //fim do cabecalho
            linha++;  // proxima linha
            coluna=0;
            sprintf(cabecalho[linha],"");
                                    //zera string
         }
      }
   }

   //  transmite cabecalho linha a linha
   for(aux=0;aux<linha;aux++)
       write(cli,
       cabecalho[aux],
       strlen(cabecalho[aux]));

   // TRANSMITE DADOS
   lido=0;
   enviado=0;
   while(lido < aler) {

      if((aler-lido)>TAM_BLOCO)
         n=read(ser, buf, TAM_BLOCO);
      else n = read(ser, buf, (aler-lido));

      if(n<TAM_BLOCO) lido=aler;

      lido+=n;

      // cifra
```

```
                if(n < TAM_BLOCO) { //completar com caracter
                                    //NULL (0 decimal)
                   for(aux=n;aux<TAM_BLOCO;aux++)
                      buf[aux]=(char)0;
                }

                /*  Carrega dados do buffer em E e D */
                          D =(int)buf[7];
                D=D<<8;   D+=(int)buf[6];
                D=D<<8;   D+=(int)buf[5];
                D=D<<8;   D+=(int)buf[4];
                          E =(int)buf[3];
                E=E<<8;   E+=(int)buf[2];
                E=E<<8;   E+=(int)buf[1];
                E=E<<8;   E+=(int)buf[0];

               /*  encripta bloco de 64 bits  */
               Blowfish_Encrypt(&ctxCif, &E, &D);

               /* retorna com dados criptografados
                * para o buffer
                */
                          buf[0]=(char) E;
                E=E>>8;   buf[1]=(char) E;
                E=E>>8;   buf[2]=(char) E;
                E=E>>8;   buf[3]=(char) E;
                          buf[4]=(char) D;
                D=D>>8;   buf[5]=(char) D;
                D=D>>8;   buf[6]=(char) D;
                D=D>>8;   buf[7]=(char) D;


                /* Envia dados criptografados ao cliente  */
                write(cli,buf,TAM_BLOCO);
                enviado+=TAM_BLOCO;

            }
         }
      case -1:
         printf("Falha na bifurcação dos processos \n");
         exit(3);
      }
   return 0;

}


/*-------------------------------------------------------
 * passivesock500 - aloca e conecta um soquete a porta 500
 *-------------------------------------------------------
 */
```

```
int
passivesock500(protocol, qlen)
char *protocol; /* "tcp" ou "udp"  */
int qlen;       /* tamanho maximo da fila de requisições
                 * para o soquete
                 */
{
   struct servent  *pse;    /* ponteiro para o registro
                             * de informações do serviço
                             */
   struct protoent *ppe;    /* ponteiro para o registro
                             * de informações do protocolo
                             */
   struct sockaddr_in sin; /* endereço do requisitante
                             * da conexao
                             */
   int    s, type, erro;    /*descritor e tipo do soquete */

   bzero((char *)&sin, sizeof(sin));
   sin.sin_family = AF_INET;
   sin.sin_addr.s_addr=INADDR_ANY;

   /*  mapeia o serviço a porta  */
   sin.sin_port = htons(500);

   /*  mapeia o nome do protocolo em seu número  */
   if ((ppe=getprotobyname(protocol))==0){
      printf("Não encontro o nome do protocolo\n");
      exit(100);
      }

   /*  decide o tipo do soquete com base no protocolo  */
   if (strcmp(protocol,"udp")==0)
      type=SOCK_DGRAM;
   else
      type=SOCK_STREAM;

   /* ALOCA O SOQUETE  */
   s = socket(PF_INET, type, ppe->p_proto);
   if (s<0) {
      printf("não consigo alocar o soquete\n");
      exit(100);
      }

   /* Conecta o soquete  */
   if ( (erro=bind(s,
                   (struct sockaddr *)&sin,
                   sizeof(sin))) < 0) {
      printf("prob. na conexão da porta. Erro %d\n", erro);
      exit(100);
      }
```

```
   if (type==SOCK_STREAM && listen(s, qlen)<0) {
      printf("não consigo escutar a porta\n");
      exit(100);
      }

   return s;
}


/*----------------------------------------
 * connectsock - aloca e conecta um soquete
 *----------------------------------------
 */
int
connectsock( host, service, protocol)
char *host;     /* servidor para o serviço  */
char *service;  /* serviço associado com a porta desejada*/
char *protocol; /* "tcp" ou "udp"  */
{
   struct hostent     *phe; /* ponteiro para o registro
                            * de informações do host
                            */
   struct servent     *pse; /* ponteiro para o registro
                            * de informações do serviço
                            */
   struct protoent    *ppe; /* ponteiro para o registro
                            * de informações do protocolo
                            */
   struct sockaddr_in sin;  /* endereço internet  */
   int    s, type;          /* descritor/tipo do soquete*/

   bzero((char *)&sin, sizeof(sin));
   sin.sin_family = AF_INET;

   /*  mapeia o serviço a porta  */
   if (pse = getservbyname(service, protocol))
      sin.sin_port = pse->s_port;
   else
      if((sin.sin_port=htons((u_short)atoi(service)))==0){
      printf("Serviço desconhecido\n");
      exit(100);
      }

   /*  resolve endereço do host (descobre IP)  */
   if (phe=gethostbyname(host))
      bcopy(phe->h_addr,
           (char *)&sin.sin_addr,
           phe->h_length);
   else if((sin.sin_addr.s_addr=inet_addr(host)) ==
                                          INADDR_NONE){
      printf("Host desconhecido\n");
```

```
        exit(100);
        }

    /*  mapeia o nome do protocolo em seu número  */
    if ((ppe=getprotobyname(protocol))==0){
        printf("Protocolo desconhecido\n");
        exit(100);
        }

    /*  decide o tipo do soquete com base no protocolo  */
    if (strcmp(protocol,"udp")==0)
        type=SOCK_DGRAM;
    else
        type=SOCK_STREAM;

    /* ALOCA O SOQUETE  */
    s = socket(PF_INET, type, ppe->p_proto);
    if (s<0) {
        printf("não consigo alocar o soquete\n");
        exit(100);
        }

    /* Conecta o soquete  */
    if (connect(s, (struct sockaddr *)&sin, sizeof(sin))<0)
{
        printf("não consigo conectar \n");
        exit(100);
        }

    return s;
}


/*---------------------------
 * reaper - limpa os escravos
 *---------------------------
 */
int
reaper()
{
    union wait status;

    while (wait3(&status,
                 WNOHANG,
                (struct rusage *)0) >= 0);
}

/*------------------------------------------------
 * "mang function" do BLOWFISH
 *------------------------------------------------
 */
```

```
unsigned long F(BLOWFISH_CTX *ctx, unsigned long x) {
   unsigned short a, b, c, d;
   unsigned long  y;

   d = x & 0x00FF;
   x >>= 8;
   c = x & 0x00FF;
   x >>= 8;
   b = x & 0x00FF;
   x >>= 8;
   a = x & 0x00FF;
   y = ctx->S[0][a] + ctx->S[1][b];
   y = y ^ ctx->S[2][c];
   y = y + ctx->S[3][d];

   return y;
}

/*---------------------------------------------------
 * ENCRIPTA
 * ctx - chaves geradas por Blowfish_Init
 * xl  - 32 bits mais significativos do bloco a
 *       criptografar
 * xr  - 32 bits menos significativos do bloco a
 *       criptografar
 * Dados criptografados ficam nos endereços apontados por
 * xl e xr.
 *---------------------------------------------------
 */
void
Blowfish_Encrypt(BLOWFISH_CTX *ctx, unsigned long *xl,
unsigned long *xr)
{
  unsigned long  Xl; /* armazenamento temporário para xl */
  unsigned long  Xr; /* armazenamento temporário para xr */
  unsigned long  temp;
  short       i;

  Xl = *xl;
  Xr = *xr;

  for (i = 0; i < N; ++i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;
    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }
  temp = Xl;
  Xl = Xr;
  Xr = temp;
```

```
  Xr = Xr ^ ctx->P[N];
  Xl = Xl ^ ctx->P[N + 1];

  *xl = Xl;
  *xr = Xr;
}

/*-------------------------------------------------------------
 * DECRIPTA
 * ctx - chaves geradas por Blowfish_Init
 * xl  - 32 bits mais significativos do bloco a
 *       criptografado
 * xr  - 32 bits menos significativos do bloco a
 *       criptografado
 * Dados criptografados ficam nos endereços apontados por
 *       xl e xr.
 *-------------------------------------------------------------
 */
void
Blowfish_Decrypt(BLOWFISH_CTX *ctx, unsigned long *xl,
unsigned long *xr)
{
  unsigned long  Xl;
  unsigned long  Xr;
  unsigned long  temp;
  short       i;

  Xl = *xl;
  Xr = *xr;

  for (i = N + 1; i > 1; --i) {
    Xl = Xl ^ ctx->P[i];
    Xr = F(ctx, Xl) ^ Xr;

    /* Troca Xl e Xr */
    temp = Xl;
    Xl = Xr;
    Xr = temp;
  }

  /* Troca Xl e Xr */
  temp = Xl;
  Xl = Xr;
  Xr = temp;

  Xr = Xr ^ ctx->P[1];
  Xl = Xl ^ ctx->P[0];

  *xl = Xl;
  *xr = Xr;
```

```
}

/*-------------------------------------------------------------
 * INICIALIZA BLOWFISH
 * ctx  -   estrutura contendo matrizes S e P a serem
 *          alteradas com
 *          base na chave secreta
 * key  -   chave secreta
 * keylen - Tamanho da chave
 * Dados criptografados ficam nos endereços apontados por
 * xl e xr.
 *-------------------------------------------------------------
 */

void
Blowfish_Init(BLOWFISH_CTX *ctx,
              unsigned char *key,
              int keyLen) {

   int i, j, k;
   unsigned long data, datal, datar;

   /*  Carrega S original  */
   for (i = 0; i < 4; i++) {
      for (j = 0; j < 256; j++)
         ctx->S[i][j] = ORIG_S[i][j];
   }

   /* Calcula P com base em P original e chave  */
   j = 0;
   for (i = 0; i < N + 2; ++i) {
      data = 0x00000000;
      for (k = 0; k < 4; ++k) {
         data = (data << 8) | key[j];
         j = j + 1;
         if (j >= keyLen)
            j = 0;
      }
      ctx->P[i] = ORIG_P[i] ^ data;
   }
   datal = 0x00000000;
   datar = 0x00000000;
   for (i = 0; i < N + 2; i += 2) {
      Blowfish_Encrypt(ctx, &datal, &datar);
      ctx->P[i] = datal;
      ctx->P[i + 1] = datar;
   }
   for (i = 0; i < 4; ++i) {
      for (j = 0; j < 256; j += 2) {
         Blowfish_Encrypt(ctx, &datal, &datar);
```

```
            ctx->S[i][j] = datal;
            ctx->S[i][j + 1] = datar;
        }
    }
}
```

Anexo B - Código Fonte do *plugin*

Observação: A listagem abaixo é do arquivo "WinTemp.cpp", onde se encontram as funções definidas pelo usuário na estrutura criada pelo Plugin SDK da netscape. Portanto, não se trata do sistema completo, e sim apenas das rotinas criadas para o desenvolvimento do protótipo.

```
/* -*- Mode: C; tab-width: 4; -*- */
/*********************************************************
 * WinTemp.c
 *********************************************************
 * Netscape Navigator Windows Plugin Template
 *********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <io.h>
#include <time.h>
#include <string.h>
#include <winsock.h>
#include "cryptlib.h"
#include "npapi.h"
#include "blowfish.h"

#include <windows.h>

LRESULT CALLBACK PluginWindowProc( HWND hWnd,
                                   UINT Msg,
                                   WPARAM wParam,
                                   LPARAM lParam);
const char* gInstanceLookupString = "instance->pdata";


typedef struct _PluginInstance
{
     NPWindow*            fWindow;
     uint16               fMode;

     HWND                 fhWnd;
     WNDPROC              fDefaultWindowProc;
} PluginInstance;

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Variáveis Globais
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int tipo=1;  // 0-embed 1-janela própria 2-form.
```

```
char Linha[2048][512];

bool CriarControles=false;
char NomeMetodo[50], NomeCampo[128][128];
int NrLinha=0,JaRec=0,NrControles=0;
HWND hjanBotao,hjanEdit;

CBlowFish BFRecebe,BFEnvia;

NPP inst;
NPStream* fluxo;

unsigned char chave[2048]; //  chave simetrica

int nrbl=0;

SOCKET connectsock(char host[128]);

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_Initialize:
 * Provides global initialization for a plug-in, and
 * returns an error value.
 *
 * This function is called once when a plug-in is loaded,
 * before the first instance is created.
 * You should allocate any memory or resources
 * shared by all instances of your plug-in at this time.
 * After the last instance has been deleted,
 * NPP_Shutdown will be called, where you can release any
 * memory or resources allocated by NPP_Initialize.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
NPError
NPP_Initialize(void)
{
    return NPERR_NO_ERROR;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_GetJavaClass:
 * New in Netscape Navigator 3.0.
 *
 * NPP_GetJavaClass is called during initialization to ask
 * your plugin what its associated Java class is.
 * If you don't have one, just return NULL.
 * Otherwise, use the javah-generated "use_" function to
 * both initialize your class and return it. If you can't
 * find your class, an error will be signalled by
 * "use_" and will cause the Navigator to
 * complain to the user.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
jref
```

```
NPP_GetJavaClass(void)
{
     return NULL;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_Shutdown:
 * Provides global deinitialization for a plug-in.
 *
 * This function is called once after the last instance of
 * your plug-in is destroyed.
 * Use this function to release any memory or resources
 * shared across all instances of your plug-in.
 * You should be a good citizen and declare that
 * you're not using your java class any more.
 * This allows java to unload it, freeing up memory.
 * ++++++++++++++++++++++++++++++++++++++++++++++++++*/
void
NPP_Shutdown(void)
{

}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_New:
 * Creates a new instance of a plug-in and returns an error
 * value.
 *
 * NPP_New creates a new instance of your plug-in with MIME
 * type specified by pluginType.
 * The parameter mode is NP_EMBED if the instance was
 * created by an EMBED tag, or NP_FULL if the instance was
 * created by a separate file.
 * You can allocate any instance-specific private data in
 * instance->pdata at this time.
 * The NPP pointer is valid until the instance is
 * Destroyed.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
NPError
NPP_New(NPMIMEType pluginType,
     NPP instance,
     uint16 mode,
     int16 argc,
     char* argn[],
     char* argv[],
     NPSavedData* saved)
{
     NPError result = NPERR_NO_ERROR;
     PluginInstance* This;

     if (instance == NULL) {
```

```
        return NPERR_INVALID_INSTANCE_ERROR;
    }
    instance->pdata=NPN_MemAlloc(sizeof(PluginInstance));
    This = (PluginInstance*) instance->pdata;
    if (This == NULL) {
        return NPERR_OUT_OF_MEMORY_ERROR;
    }
    /* mode is NP_EMBED, NP_FULL, or NP_BACKGROUND (see
     * npapi.h)
     */
    This->fWindow = NULL;
    This->fMode = mode;

    This->fhWnd = NULL;
    This->fDefaultWindowProc = NULL;

    /* PLUGIN DEVELOPERS:
     *   Initialize fields of your plugin
     *   instance data here.  If the NPSavedData is non-
     *   NULL, you can use that data (returned by you from
     *   NPP_Destroy to set up the new plugin instance).
     */

    //guarda handler da instancia como global
    inst=instance;

    //  determina parametro form
    int aux;
    for(aux=0;aux<argc;aux++)
        if(((argn[aux][0]=='f')||(argn[aux][0]=='F'))&&
           ((argn[aux][1]=='o')||(argn[aux][1]=='O'))&&
           ((argn[aux][2]=='r')||(argn[aux][2]=='R'))&&
           ((argn[aux][3]=='m')||(argn[aux][3]=='M')) )
           if((argv[aux][0]=='t')||(argv[aux][0]=='T')) {
               tipo = 2;   // é form
               CriarControles = true;
           } else {
            tipo = 0;   // é embed
               CriarControles = false;
           }

    // se nao houver param FORM, fica tipo 1 (nova janela)

    return result;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_Destroy:
 * Deletes a specific instance of a plug-in and returns an
 * error value. NPP_Destroy is called when a plug-in
 * instance is deleted, typically because the
```

```
 * user has left the page containing the instance, closed
 * the window, or quit the application. You should delete
 * any private instance-specific information stored in
 * instance->pdata. If the instance being deleted is the
 * last instance created by your plug-in, NPP_Shutdown will
 * subsequently be called, where you can delete any data
 * allocated in NPP_Initialize to be shared by all your
 * plug-in's instances. Note that you should not perform
 * any graphics operations in NPP_Destroy as the instance's
 * window is no longer guaranteed to be valid.
 *+++++++++++++++++++++++++++++++++++++++++++++++++++*/
NPError
NPP_Destroy(NPP instance, NPSavedData** save)
{
     PluginInstance* This;

     if (instance == NULL)
          return NPERR_INVALID_INSTANCE_ERROR;

     This = (PluginInstance*) instance->pdata;
     if( This->fWindow != NULL ) {
          /* If we have a window, clean it up. */
          SetWindowLong( This->fhWnd,
                         GWL_WNDPROC,
                         (LONG)This->fDefaultWindowProc);
          This->fDefaultWindowProc = NULL;
          This->fhWnd = NULL;
     }

     return NPERR_NO_ERROR;

/* PLUGIN DEVELOPERS:
 *   If desired, call NP_MemAlloc to create a
 *   NPSavedDate structure containing any state information
 *   that you want restored if this plugin instance is
 *   later recreated.
 */

     if (This != NULL) {
          NPN_MemFree(instance->pdata);
          instance->pdata = NULL;
     }

     return NPERR_NO_ERROR;
}

/*+++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_SetWindow:
 * Sets the window in which a plug-in draws, and
 * returns an error value.
 * NPP_SetWindow informs the plug-in instance specified by
```

```
 * instance of thethe window denoted by window in which the
 * instance draws. This NPWindow pointer is valid for the
 * life
 * of the instance, or until NPP_SetWindow is called again
 * with a different value. Subsequent calls to
 * NPP_SetWindow for a given instance typically indicate
 * that the window has been resized. If either window or
 * window->window are NULL, the plug-in must not perform
 * any additional graphics operations on the window and
 * should free any resources associated with the window.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
NPError
NPP_SetWindow(NPP instance, NPWindow* window)
{
     NPError result = NPERR_NO_ERROR;
     PluginInstance* This;

     if (instance == NULL)
          return NPERR_INVALID_INSTANCE_ERROR;

     This = (PluginInstance*) instance->pdata;

     /*
      * PLUGIN DEVELOPERS:
      *   Before setting window to point to the
      *   new window, you may wish to compare the new
      *   window info to the previous window
      *   (if any) to note window size changes, etc.
      */
     if( This->fWindow != NULL )
        /* If we already have a window, clean
         * it up before trying to subclass
         * the new window.
         */
     {
     if((window==NULL)||( window->window == NULL ) ) {
     /* There is now no window to use. get rid of the old
      * one and exit. */
          SetWindowLong( This->fhWnd,
                         GWL_WNDPROC,
                         (LONG)This->fDefaultWindowProc);
               This->fDefaultWindowProc = NULL;
               This->fhWnd = NULL;
               This->fWindow=window;
               return NPERR_NO_ERROR;
          }

     else if ( This->fhWnd == (HWND) window->window ) {
          /* The new window is the same as the old one.
           * Redraw and get out. */
               InvalidateRect( This->fhWnd, NULL, TRUE );
```

```
            UpdateWindow( This->fhWnd );
            This->fWindow=window;
            return NPERR_NO_ERROR;
        }
        else {
            /* Clean up the old window, so that we can
             * subclass the new one later. */

             SetWindowLong( This->fhWnd,
                            GWL_WNDPROC,
                           (LONG)This->fDefaultWindowProc);
            This->fDefaultWindowProc = NULL;
            This->fhWnd = NULL;
        }
    }
    else if((window == NULL)||(window->window==NULL ) ) {
        /* We can just get out of here if there is no
         * current window and there is no new window to
         * use. */
            This->fWindow=window;
        return NPERR_NO_ERROR;
    }

    /* At this point, we will subclass
     * window->window so that we can begin drawing and
     * receiving window messages. */
    This->fDefaultWindowProc =
    (WNDPROC)SetWindowLong( (HWND)window->window,
                            GWL_WNDPROC,
                            (LONG)PluginWindowProc);
    This->fhWnd = (HWND) window->window;
    SetProp( This->fhWnd,
             gInstanceLookupString,
             (HANDLE)This);

    InvalidateRect( This->fhWnd, NULL, TRUE );
    UpdateWindow( This->fhWnd );

    This->fWindow = window;

    return result;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_NewStream:
 * Notifies an instance of a new data stream and returns an
 * error value.
 *
 * NPP_NewStream notifies the instance denoted by instance
 * of the creation of a new stream specifed by stream. The
 * NPStream* pointer is valid until the stream is
```

```
 * destroyed. The MIME type of the stream is provided by
 * the parameter type.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
NPError
NPP_NewStream(NPP instance,
           NPMIMEType type,
           NPStream *stream,
           NPBool seekable,
           uint16 *stype)
{
     PluginInstance* This;

     if (instance == NULL)
          return NPERR_INVALID_INSTANCE_ERROR;

     This = (PluginInstance*) instance->pdata;

     if(tipo==1) NPN_NewStream(inst,
                               "text/html",
                               "_blank",
                               &fluxo);

     return NPERR_NO_ERROR;
}


/* PLUGIN DEVELOPERS:
 *   These next 2 functions are directly relevant in a
 * plug-in which handles the data in a streaming manner.
 * If you want zero bytes because no buffer space is YET
 * available, return 0. As long as the stream has not been
 * written to the plugin, Navigator will continue trying to
 * send bytes.  If the plugin doesn't want them, just
 * return some large number from NPP_WriteReady(), and
 * ignore them in NPP_Write().  For a NP_ASFILE stream,
 * they are still called but can safely be ignored using
 * this strategy.
 */

int32 STREAMBUFSIZE = 1024;
    /* If we are reading from a  file in NPAsFile mode so
     * we can take any size stream in our write call (since
     * we ignore it) */

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_WriteReady:
 * Returns the maximum number of bytes that an instance is
 * prepared to accept from the stream.
 * NPP_WriteReady determines the maximum number of bytes
 * that the instance will consume from the stream in a
 * subsequent call NPP_Write. This function allows Netscape
```

```
 * to only send as much data to the instance as the
 * instance is capable of handling at a time, allowing more
 * efficient use of resources within both Netscape and the
 * plug-in.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
int32
NPP_WriteReady(NPP instance, NPStream *stream)
{
     PluginInstance* This;
     if (instance != NULL)
          This = (PluginInstance*) instance->pdata;

     /* Number of bytes ready to accept in NPP_Write() */
     return STREAMBUFSIZE;
}


/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_Write:
 * Delivers data from a stream and returns the number of
 * bytes written.
 *
 * NPP_Write is called after a call to NPP_NewStream in
 * which the plug-in requested a normal-mode stream, in
 * which the data in the stream is delivered progressively
 * over a series of calls to NPP_WriteReady and NPP_Write.
 * The function delivers a buffer buf of len bytes of data
 * from the stream identified by stream to the instance.
 * The parameter offset is the logical position of
 * buf from the beginning of the data in the stream.
 *
 * The function returns the number of bytes written
 * (consumed by the instance).
 * A negative return value causes an error on the stream,
 * which will subsequently be destroyed via a call to
 * NPP_DestroyStream.
 *
 * Note that a plug-in must consume at least as many bytes
 * as it indicated in the preceeding NPP_WriteReady call.
 * All data consumed must be either processed immediately
 * or copied to memory allocated by the plug-in: the buf
 * parameter
 * is not persistent.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
int32
NPP_Write(NPP instance,
          NPStream *stream,
          int32 offset,
          int32 len,
          void *buffer)
{
```

```
     if (instance != NULL) {
          PluginInstance* This = (PluginInstance*)
instance->pdata;
     }

     static char GET[1024]="";
     int aux;
     static bool campo=false, metodo=true,
                 reqIniCompleta=false;
     static int PosLinha=0,jaRec=0;
     char mensagem[128];

     BYTE* p;
     p=(BYTE*)buffer;

    // copia mensagem recebida para var GET
    for (aux=0;aux<len;aux++) GET[jaRec+aux]=*(p+aux);
     jaRec=jaRec+len;
     // verifica se terminou a mensagem inicial
    for (aux=0;aux<len;aux++)
          if ( (*(p+aux+0) == (char)13)&&
               (*(p+aux+1) == (char)10)&&
               (*(p+aux+2) == (char)13)&&
               (*(p+aux+3) == (char)10) )
                    reqIniCompleta=true;

     // se nao terminou bloco inicial,
     // retorna para pegar mais dados
     if (!reqIniCompleta) return len;

     /*  PREPARA REQUISIÃO PARA O WEBSECD
      *  Retira tipo MIME colocado para carregar plugin
      *  e transforma o "GET" em "SGET"
      */
    GET[0]='S';
    for(aux=34;aux<strlen(GET);aux++) GET[aux-33]=GET[aux];
    GET[aux-33]='\0';

    /*
     * Descobre host cuja conexão é desejada
     */
     int posIni=0,posFim=0;
     char host[128];
     aux=0;
     while (posFim==0) {
          if ((GET[aux+0]=='H')&&
              (GET[aux+1]=='o')&&
              (GET[aux+2]=='s')&&
              (GET[aux+3]=='t')  ) {
           aux+=6;
           posIni=aux;
```

```
      }
      if ((posIni>0)&&(GET[aux]==':')) posFim=aux;
      aux++;
 }
 for(aux=posIni;aux<posFim;aux++)
                  host[aux-posIni]=GET[aux];
 host[posFim-posIni]='\0';

/*
 *  Abre conexão com host e envia GET
 */
 SOCKET ser;
 int n=1;
 char sn_tam_cert[2048]="", cert[2048], buf[1024]="";
 char cTamChave[4];
 int tamChave,num;

 ser = connectsock(host);
 n = send(ser,GET,strlen(GET),0);

 /* Receber :
  *    1-S/N indicando necessidade de envio de
  *      certificado do cliente
  *    2-Tamanho da chave simetrica com 3 caracteres
  *    3-Certificado
  */
 n = recv(ser,buf,64,0);
 while (n>0) {
   strncat(sn_tam_cert,buf,n);
    n = recv(ser,buf,64,0);
 }

 if(sn_tam_cert[0]=='S') {
    // Enviar certificado do cliente
    FILE* certCli;
    certCli = fopen("certcli.crt","r");
    n=1;
    n=fread(buf,64,1,fd);
    while (n>0) {
      send(ser,buf,n,0);
      n=fread(buf,64,1,fd);
    }
 }

 // copia tamanho da chave para cTamChave
for(aux=1;aux<4;aux++)
   cTamChave[aux-1]=sn_tam_cert[aux];
cTamChave[3]='\0';
tamChave=atoi(cTamChave);

// Retira os 4 primeiros caracteres recebidos,
```

```
// que nao fazem parte do certificado copiando
// para a var cert.
for(aux=4;aux<strlen(sn_tam_cert);aux++)
   cert[aux-4]=sn_tam_cert[aux];

/*
 *   Verificar o certificado
 */

CRYPT_CERTIFICATE cryptCert, cryptCertCA;
                      // objeto certificado da cryptlib

char certCA[2048]="";// Conterá certificado do CA

FILE* FDcertCA;       // Arquivo com certificado do CA

// inicializa cyptLib
cryptInit();

// Le certificado da CA
FDcertCA = fopen("certCA.crt","r");
n=1;
n=fread(buf,64,1,FDcertCA);
while (n>0) {
   strcat(certCA,buf);
    n=fread(buf,64,1,FDcertCA);
}
cryptImportCert(certCA,
                strlen(certCA),
                CRYPT_UNUSED,
                &cryptCertCA);

 // importa certificado do servidor
cryptImportCert(cert,
                strlen(cert),
                CRYPT_UNUSED,
                &cryptCert);

 //  checa certificado do servidor
 if(cryptCheckCert(cryptCert,
                   cryptCertCA) == CRYPT_ERROR_INVALID)
      exit(-1);

 // Sortear chave privada
 srand((unsigned)time( NULL ) );
for( aux = 0; aux<(tamChave/8); aux++ ) {
   num = rand();
   while(num>128) num -= 128;
        // mantem chave no ASCII padrão
  chave[aux]=(char)num;
 }
```

```
chave[tamChave/8]='\0';

//  Inicializa Blowfish com chave privada sorteada
BFRecebe.Initialize(chave,tamChave);
BFEnvia.Initialize(chave,tamChave);

/*
 * Cifrar chave simétrica com RSA e enviar ao servidor
 */
CRYPT_ENVELOPE rsaEnvelope;
char chaveCifrada[1024];

cryptCreateEnvelope( &rsaEnvelope,
                     CRYPT_UNUSED,
                     CRYPT_FORMAT_CRYPTLIB);
 //  Adiciona chave pública contida no certificado
 cryptSetAttribute( rsaEnvelope,
                    CRYPT_ENVINFO_PUBLICKEY,
                    cryptCert);
 //  Passa chave para ser envelopada (cifrada)
 cryptSetAttribute( rsaEnvelope,
                    CRYPT_ENVINFO_DATASIZE,
                    tamChave);
 cryptPushData( rsaEnvelope,
                chave,
                tamChave,
                NULL);
 //  Chama pushdata sem dados para fechar o processo
 // (requerido pela cryptlib)
cryptPushData( rsaEnvelope,
               NULL,
               0,
               NULL);
 //  Retira dados criptografados
 cryptPopData( rsaEnvelope,
               chaveCifrada,
               1024,
               NULL);
 //  Destroi envelope
 cryptDestroyEnvelope(rsaEnvelope);

 //  Envia ao servidor
 n = send(ser,chaveCifrada, strlen(chaveCifrada), 0);

 // Iniciar recepção de dados decifrando com blowfish

 if(tipo==0) {  // EMBED

      // DECIFRA
      BFRecebe.Decode(p,p,(DWORD) len);
```

```
     for(aux=0; aux<len; aux++) {

if((*(p+aux)!=(char)0)&&(*(p+aux)!=(char)13)){
               if(*(p+aux)==(char)10) {
                    if(NrLinha<1024) NrLinha++;
                   PosLinha=0;

               } else {
                    Linha[NrLinha][PosLinha]=*(p+aux);
                    if(PosLinha<128) PosLinha++;

               }
          }
     }
} else if(tipo==1) {  //  Janela Própria

     // retirar cabecalho ???

     // DECIFRA
     BFRecebe.Decode(p,p,(DWORD) len);

     char MyData[9]="        ";
     for(aux=0; aux<len; aux++) MyData[aux]=*(p+aux);
     NPN_Write(inst, fluxo, strlen(MyData), MyData);

} else if(tipo==2){   // FORM
     for(aux=0; aux<len; aux++) {
          if(metodo) {

if((*(p+aux)!=(char)0)&&(*(p+aux)!=(char)13)){
                    if(*(p+aux)==(char)10) {
                         metodo=false;
                         PosLinha=0;
                    } else {

NomeMetodo[PosLinha]=*(p+aux);
                         PosLinha++;
                    }
               }
          } else {

if((*(p+aux)!=(char)0)&&(*(p+aux)!=(char)13)){
                    if(*(p+aux)==(char)10) {
                         NrLinha++;
                         PosLinha=0;
                         campo=false;
                    } else {
                         if(*(p+aux)==','){
                              campo=true;
                              NrControles++;
                              PosLinha=0;
```

```
                                }else{
                                     if(campo)

      NomeCampo[NrLinha][PosLinha]=*(p+aux);
                                     else

      Linha[NrLinha][PosLinha]=*(p+aux);
                                     PosLinha++;
                                }
                           }
                      }

                }
           }
      }

     JaRec+=len;

     return len;          /* The number of bytes accepted */
}

/*------------------------------------
 * Cria e conecta um soquete 'a um host
 *------------------------------------*/
SOCKET
connectsock (char host[128])
   {
   WORD   wVersionRequested; /* versão da API solicitada */
   WSADATA wsadata;          /* Windows Sockets API data */
   SOCKET sock;              /* socket descriptor */
   struct hostent FAR *hp;   /* ptr to host info struct */
   struct servent FAR *sp; /* ptr to service info struct */
   struct sockaddr_in server;/* socket address and port */
   int    stat;   /* valor de retorno de função (status) */
   char   mensagem[128];     /* mensagem ao usuário  */

   /*-----------------------------------------------------*/
   /* Start up the Windows Sockets API.                   */
   /* We are looking for an API version 1.1               */
   /* implementation.                                     */
   /*-----------------------------------------------------*/

   wVersionRequested = 0x0101;
   stat = WSAStartup( wVersionRequested, &wsadata );
   if (stat != 0){
      fwrite("No usable WINSOCK.DLL found\n",40,1,fd);
      exit( 1 );
   }
   else if (LOBYTE( wsadata.wVersion) != 1  &&
            HIBYTE( wsadata.wVersion) != 1)     {
      fwrite("WINSOCK.DLL nao suporta vrs 1.1\n",40,1,fd);
```

```
      WSACleanup();
      exit( 1 );
   }

   /*---------------------------------------------*/
   /* Create the socket                           */
   /*---------------------------------------------*/

   sock = socket( AF_INET, SOCK_STREAM, 0 );
   if (sock == INVALID_SOCKET) {
      fwrite("Nao consegui criar o soaquete\n",40,1,fd);
      WSACleanup();
      exit( 1 );
   }
   /*---------------------------------------------*/
   /* Connect (and bind) the socket to the service  */
   /* and provider.                               */
   /*---------------------------------------------*/

   hp = gethostbyname( host );
   if (hp == NULL) {
      strcpy(mensagem, "gethostbyname falhou:");
      strcat(mensagem,host);
      strcat(mensagem,".\n");
      fwrite(mensagem,strlen(mensagem),1,fd);
      WSACleanup();
      exit( 1 );
   }

   memset( &server, 0, sizeof(server) );
   memcpy( &server.sin_addr, hp->h_addr, hp->h_length );
   server.sin_family = hp->h_addrtype;

   server.sin_port = htons( atoi("500") );

   stat = connect( sock,
                  (const struct sockaddr FAR *)&server,
                   sizeof(server) );
   if (stat != 0){
      fwrite("Nao consegui conectar\n",40,1,fd);
      WSACleanup();
      exit( 1 );
   }
   return sock;
}


/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_DestroyStream:
 * Indicates the closure and deletion of a stream,
 * and returns an error value.
```

```
 *
 * The NPP_DestroyStream function is called when the
 * stream identified by stream for the plug-in instance
 * denoted by instance will be destroyed. You
 * should delete any private data allocated in
 * stream->pdata at this time.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
NPError
NPP_DestroyStream(NPP instance,
                  NPStream *stream,
                  NPError reason)
{
     PluginInstance* This;

     if (instance == NULL)
          return NPERR_INVALID_INSTANCE_ERROR;
     This = (PluginInstance*) instance->pdata;

     NPN_DestroyStream(inst, fluxo, NPRES_DONE);

     return NPERR_NO_ERROR;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_StreamAsFile:
 * Provides a local file name for the data from a stream.
 *
 * NPP_StreamAsFile provides the instance with a full
 * path to a local file, identified by fname, for the
 * stream specified by stream. NPP_StreamAsFile is
 * called as a result of the plug-in requesting mode
 * NP_ASFILEONLY or NP_ASFILE in a previous call to
 * NPP_NewStream. If an error occurs while
 * retrieving the data or writing the file, fname may
 * be NULL.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
void
NPP_StreamAsFile(NPP instance,
                 NPStream *stream,
                 const char* fname)
{
     PluginInstance* This;
     if (instance != NULL)
          This = (PluginInstance*) instance->pdata;
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_Print:
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
void
NPP_Print(NPP instance, NPPrint* printInfo)
```

```
{
    if(printInfo == NULL)
        return;

    if (instance != NULL) {
        PluginInstance* This =
                        (PluginInstance*) instance->pdata;

        if (printInfo->mode == NP_FULL) {
            /*
             * PLUGIN DEVELOPERS:
             *  If your plugin would like to take over
             *  printing completely when it is in full-
             *  screen mode,set printInfo->pluginPrinted
             *  to TRUE and print your
             *  plugin as you see fit.  If your plugin
             *  wants Netscape to handle printing
             *  in this case,set printInfo->pluginPrinted
             *  to FALSE (the default) and do nothing.
             *  If you do want to handle printing
             *  yourself, printOne is true if the print
             *  button (as opposed to the print menu) was
             *  clicked. On the Macintosh, platformPrint
             *  is a THPrint; on Windows, platformPrint
             *  is a structure (defined in npapi.h)
             *  containing the printer name, port,etc.
             */

            void* platformPrint =
            printInfo->print.fullPrint.platformPrint;

            NPBool printOne =
                 printInfo->print.fullPrint.printOne;

            /* Do the default*/
            printInfo->print.fullPrint.
            pluginPrinted = FALSE;
        }
        else {    /* If not fullscreen, we must be
                   * embedded */
            /*
             * PLUGIN DEVELOPERS:
             * If your plugin is embedded, or is full-
             * screen but you returned false in
             * pluginPrinted above, NPP_Print
             * will be called with mode == NP_EMBED.
             * The NPWindow in the printInfo gives
             * the location and dimensions of
             * the embedded plugin on the printed page.
             * On the Macintosh, platformPrint is the
             * printer port; on Windows, platformPrint
```

```
                * is the handle to the printing
                * device context.
                */

                NPWindow* printWindow =
                     &(printInfo->print.embedPrint.window);
                void* platformPrint =
                printInfo->print.embedPrint.platformPrint;
          }
     }
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_URLNotify:
 * Notifies the instance of the completion of a URL
 * request.
 *
 * NPP_URLNotify is called when Netscape completes a
 * NPN_GetURLNotify or NPN_PostURLNotify request, to
 * inform the plug-in that the request, identified by
 * url, has completed for the reason specified by reason.
 * The most common reason code is NPRES_DONE, indicating
 * simply that the request completed normally. Other
 * possible reason codes are NPRES_USER_BREAK,
 * indicating that the request was halted due to a user
 * action (for example, clicking the "Stop" button), and
 * NPRES_NETWORK_ERR, indicating that the request could not
 * be completed (for example, because the URL could not be
 * found). The complete list of reason codes is found in
 * npapi.h.
 *
 * The parameter notifyData is the same plug-in-private
 * value passed as an argument to the corresponding
 * NPN_GetURLNotify or NPN_PostURLNotify
 * call, and can be used by your plug-in to uniquely
 * identify the request.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
void
NPP_URLNotify(NPP instance,
              const char* url,
              NPReason reason,
              void* notifyData)
{
     /* Not used in the Simple plugin. */
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * NPP_HandleEvent:
 * Mac-only, but stub must be present for Windows
 * Delivers a platform-specific event to the instance.
 *
```

```
 * On the Macintosh, event is a pointer to a standard
 * Macintosh EventRecord.
 * All standard event types are passed to the instance as
 * appropriate. In general, return TRUE if you handle the
 * event and FALSE if you ignore the event.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
int16
NPP_HandleEvent(NPP instance, void* event)
{
     return 0;
}
/*****************************************************/

/*++++++++++++++++++++++++++++++++++++++++++++++++++
 * PluginWindowProc
 *
 * Handle the Windows window-event loop.
 ++++++++++++++++++++++++++++++++++++++++++++++++++*/
LRESULT CALLBACK PluginWindowProc( HWND hWnd, UINT Msg,
WPARAM wParam, LPARAM lParam)
{
     PluginInstance* This =
(PluginInstance*) GetProp(hWnd, gInstanceLookupString);
     PAINTSTRUCT paintStruct;
     HDC hdc;

     char TextoEdit[51]="";
     int resto,completar;   // para tornar o texto enviado
                            // multiplo de STREAMBUFSIZE
     char Enviar[100]="";
     BYTE* p;  //  para passar ao BlowFish para cifragem;
     int aux;

     sprintf(Linha[100],"     ");
     for(aux=0;aux<40;aux++) strcat(Linha[100],"     ");

     if(CriarControles) {
          CriarControles=false;

          hjanEdit = CreateWindow ("edit", NULL,
                    WS_CHILD | WS_VISIBLE | WS_BORDER,
                    strlen(Linha[0])*7, 0,
                    200, 20,
                    hWnd, (HMENU) 0,
                    ((LPCREATESTRUCT) lParam) -> hInstance,
                    NULL);

          hjanBotao = CreateWindow ("button", "Enviar",
                    WS_CHILD|WS_VISIBLE|BS_DEFPUSHBUTTON,
                    0, 25,
                    60, 40,
```

```
                    hWnd, (HMENU) 1,
                    ((LPCREATESTRUCT) lParam) -> hInstance,
                    NULL);
     }

     switch( Msg ) {
          case WM_PAINT: {
               hdc = BeginPaint( hWnd, &paintStruct );
               if(tipo==0) {        // EMBED

                    for(aux=0;aux<=NrLinha;aux++)
                         TextOut(hdc, 0, aux*15,
                                 Linha[aux],
                                 strlen(Linha[aux]) );

               }else if(tipo==1) { // Janela própria
                    char Mens[50];
                    sprintf(Mens,"Numero de blocos \
                                   recebidos: %d", nrbl);
                    TextOut(hdc,0,0, "WEBSEC: Dados sendo \
                          exibidos em outra janela.", 45 );
                TextOut(hdc, 0, 15, Mens, strlen(Mens) );
               }else if(tipo==2){       // FORM

                    TextOut(hdc, 0, 0, Linha[0],
                                       strlen(Linha[0]) );

               }
               EndPaint( hWnd, &paintStruct );
               break;
          }
          case WM_DRAWITEM:
          case WM_COMMAND: {
               if((LOWORD(wParam)==1)&&(HIWORD(wParam)==0))
{

                    //  pega valor do campo
                    GetWindowText(hjanEdit,TextoEdit,50);
                resto = strlen(TextoEdit) % STREAMBUFSIZE;
                    if(resto!=0) {
                       completar=STREAMBUFSIZE-resto;
                       for(aux=0;aux<completar;aux++)
                                       strcat(TextoEdit," ");
                }

                    // cifrar TextoEdit
                    BFEnvia.Initialize(chave,16);
                    p=(BYTE*)TextoEdit;
                    BFEnvia.Encode(p,p,(DWORD)
                                       strlen(TextoEdit));
```

```
                    strcat(Enviar,NomeMetodo);
                    strcat(Enviar,"?");
                    strcat(Enviar,NomeCampo[0]);
                    strcat(Enviar,"=");
                    strcat(Enviar,(char*)p);
                    strcat(Enviar,"/\n\n");

                    NPN_GetURL(inst,
                                Enviar,
                                "_blank");

               }

               break;
          }
          default: {
               This->fDefaultWindowProc( hWnd,
                                         Msg,
                                         wParam,
                                         lParam);
          }
     }
     return 0;
}
```

Anexo C - Páginas HTML Utilizada nos Testes de Desempenho

PARA MEDIÇÃO COM O WEBSEC:

```
<script>
function MostraHora()
  {
    HrFim = new Date()
    alert('Tempo de carga da pagina: '+
    (HrFim.getTime()-
     HrIni.getTime())+
     " milisegundos" )
  }
</script>

<html>
  <head>
    <title> Banco Simulado - Extrato </title>
  </head>

  <body bgcolor="#8859988" onLoad="MostraHora()">
    <center>

    <script> var HrIni = new Date() </script>

    <img src="banner.bmp"> <P>
```

```
<embed src="182.169.1.101:500//teste/extrato.sec"
    width=600
    height=300
  form="false">
</center>


</body>
</html>
```

PARA MEDIÇÃO COM O SSL:

Definição dos frames:

```
<script>
function MostraHora()
 {
    HrFim = new Date()
    alert('Clock ao termino da carga da pagina: '+HrFim.getTime() )
 }
</script>


<frameset rows="30%,70%" onLoad="MostraHora()">
  <frame src="http://192.168.1.101/teste/banner.html">
  <frame src="https://192.168.1.101/teste/ext.txt">
</frameset>
```

BANNER.HTML:

```
<body>
<script>
  HrInicio = new Date()
  document.write("Clock no inicio da carga da pagina"+HrInicio.getTime())
```

```
</script>
<img src="banner.jpg">
</body>
```

Anexo D – Sítios especializados em segurança

Os sítios abaixo compõem importante fonte de informação atualizada sobre segurança em redes de computadores e devem ser consultados periodicamente pelos interessados no estudo do assunto.

CERT – www.cert.org - *Computer Emergency Response Team,* mantido pela Universidade de Carnigie Mellon. Possui uma lista de correio eletrônico que mantêm informados seus assinantes sobre novos ataques, vírus e valhas em sistemas.

Attrition – www.attrition.org - Sem fins lucrativas, mantido por internautas a título de *hobby.*

AddSecure – www.addsecure.net - Empresa que produz ferramentas de auditoria e mantêm o "*Journal of Internet Security*".

CounterPane – www.counterpane.com - Empresa que produz ferramentas de segurança e mantem artigos relacionados em seu site.

Security Focus – www.securityfocus.com - Publicação eletrônica especializada.

Multivirtual -www.multivirtual.com.br – Referencia diversas fontes de informações estatísticas sobre a Internet.